Executive Summary of Artificial Intelligence and Emerging Technologies

Prepared by:

Dr. Mejdal Alqahtani

Table of contents

1. Overview on AI
2. Basic math and probabilities
3. Machine learning
    - 3.1. Supervised learning
    - 3.2. Unsupervised learning
4. Deep learning
5. Reinforcement learning
6. Programing languages
    - 6.1. Python language
    - 6.2. R language
    - 6.3. SQL language
7. Business Intelligence
    - 7.1. Tableau
    - 7.2. Power BI
    - 7.3. Data Visualization

8. Emerging Technologies

8.1. Artificial Intelligence (AI)

8.2. Internet of Things (IoT)

8.3. Robotic Process Automation (RPA)

8.4. Cloud Computing

8.5. Quantum Computing

8.6. 3D Printing

8.7. 5th Generation Network (5G)

8.8. Extended Reality (XR)

8.9. Blockchain

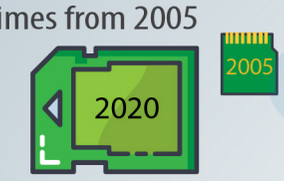
8.10. Cyber Security

1. Overview on AI

BIG DATA

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005



It's esimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data is created each day

6 BILLION PEOPLE
have cell phones

WORLD POPULATION: 7 BILLION

Most companies in the U.S. have at least
100 TERABYTES
[1,00,000 GIGABYTES]
of data stored

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

30 BILLION PIECES OF CONTENT
are shared on Facebook every month

facebook

Variety
DIFFERENT FROMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

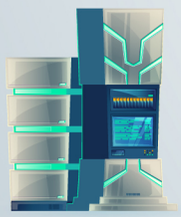
4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month

YouTube

1 IN 3 BUSINESS
don't trust the information they use to make decisions

Morden cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

Velocity
ANALYSIS OF STREAMING DATA



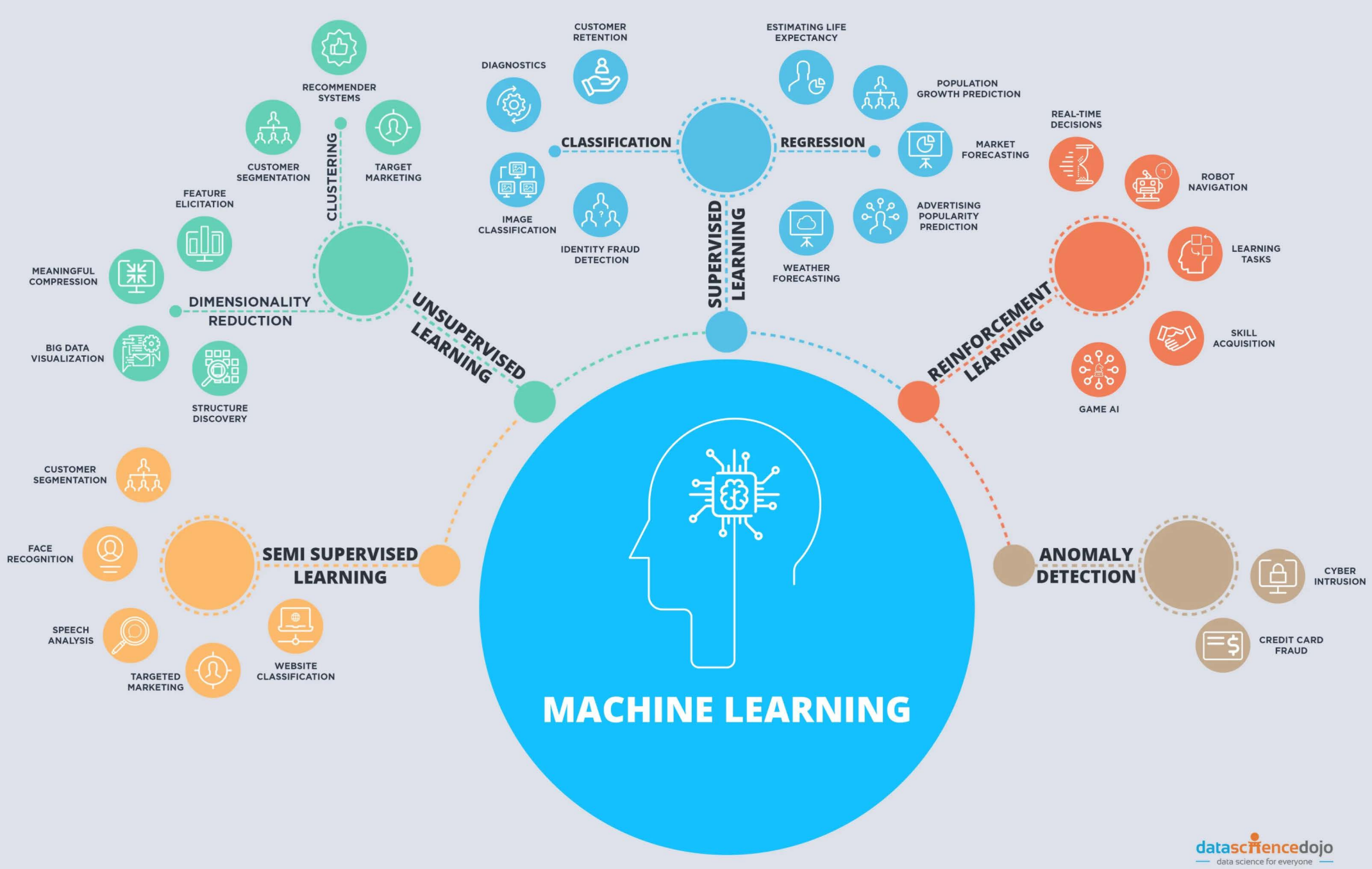
The New York Stock Exchange capture
1 TB OF TRADE INFORMATION
during each trading session

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
- almost 2.5 connections per person on earth

27% OF RESPONDENTS
in one survey were unsure of how much of their data was inaccurate

Veracity
UNCERTAINTY OF DATA

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

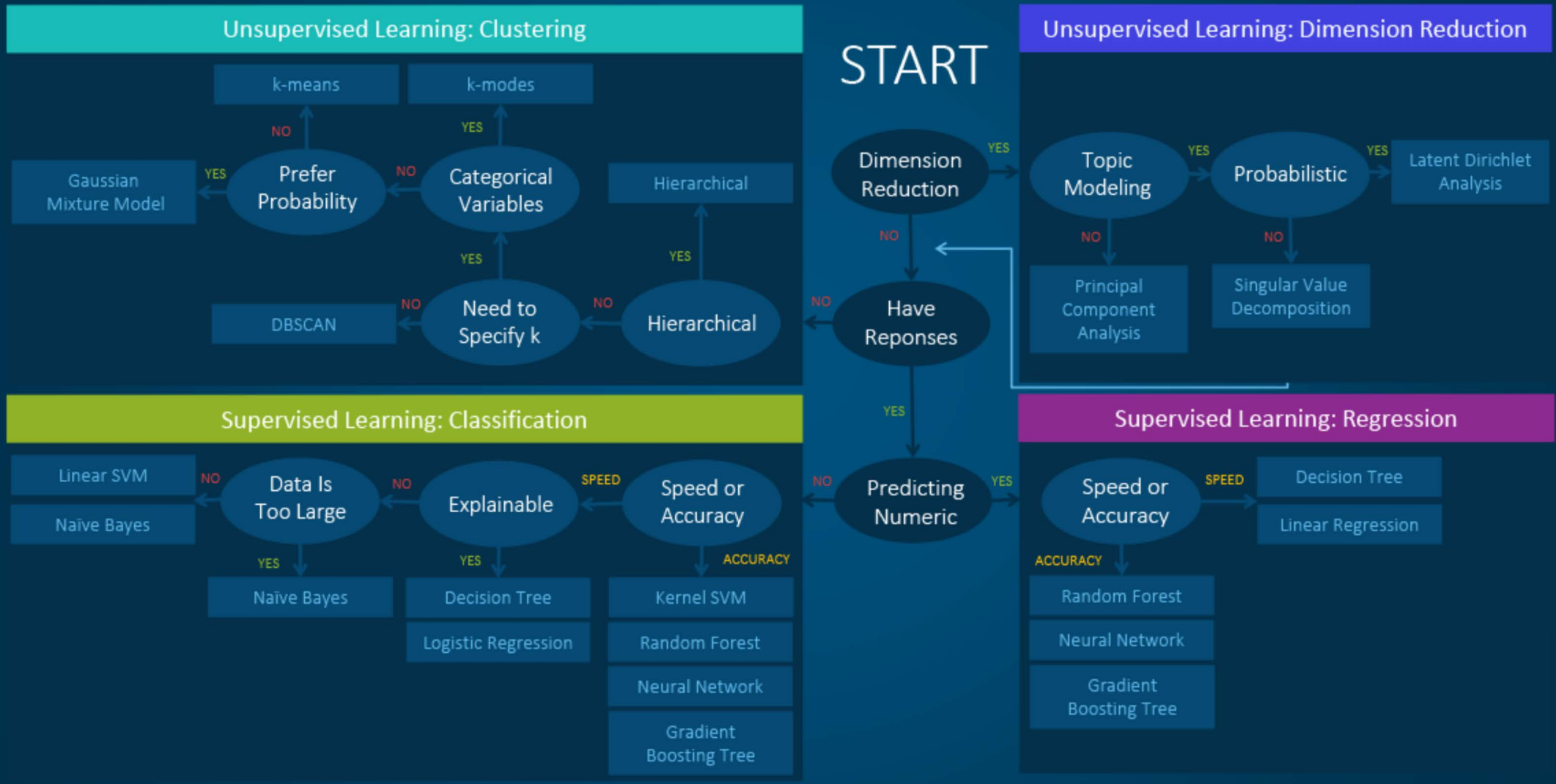


data iku

TOP PREDICTION ALGORITHMS

TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	 Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	X **Sometimes too simple to capture complex relationships between variables.** X **Does poorly with correlated features.**
	 Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	X **Sometimes too simple to capture complex relationships between variables.** X **Does poorly with correlated features.**
Tree-based	 Decision tree	**A series of yes/no rules based on the features,** forming a tree, to match all possible outcomes of a decision.	Easy to understand.	X **Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.**
	 Random Forest	Takes advantage of many decision trees, with rules created from subsamples of features. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	X **Models can get very large.** X **Not easy to understand predictions.**
	 Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	X **A small change in the feature set or training set can create radical changes in the model.** X **Not easy to understand predictions.**
Neural networks	 Neural networks	Interconnected «neurons» that pass messages to each other. Deep learning uses several **layers of neural networks stacked on top of one another.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	X **Very slow to train, because they often have a very complex architecture.** X **Almost impossible to understand predictions.**

Machine Learning Algorithms Cheat Sheet



Microsoft Azure Machine Learning: Algorithm Cheat Sheet

This cheat sheet helps you choose the best Azure Machine Learning Studio algorithm for your predictive analytics solution. Your decision is driven by both the nature of your data and the question you're trying to answer.

START

Finding unusual data points

Discovering structure

Predicting categories

Predicting values

Three or more

Two

ANOMALY DETECTION

- **One-class SVM** — >100 features, aggressive boundary
- **PCA-based anomaly detection** — Fast training

CLUSTERING

- **K-means**

MULTI-CLASS CLASSIFICATION

- Fast training, linear model — **Multiclass logistic regression**
- Accuracy, long training times — **Multiclass neural network**
- Accuracy, fast training — **Multiclass decision forest**
- Accuracy, small memory footprint — **Multiclass decision jungle**
- Depends on the two-class classifier, see notes below — **One-v-all multiclass**

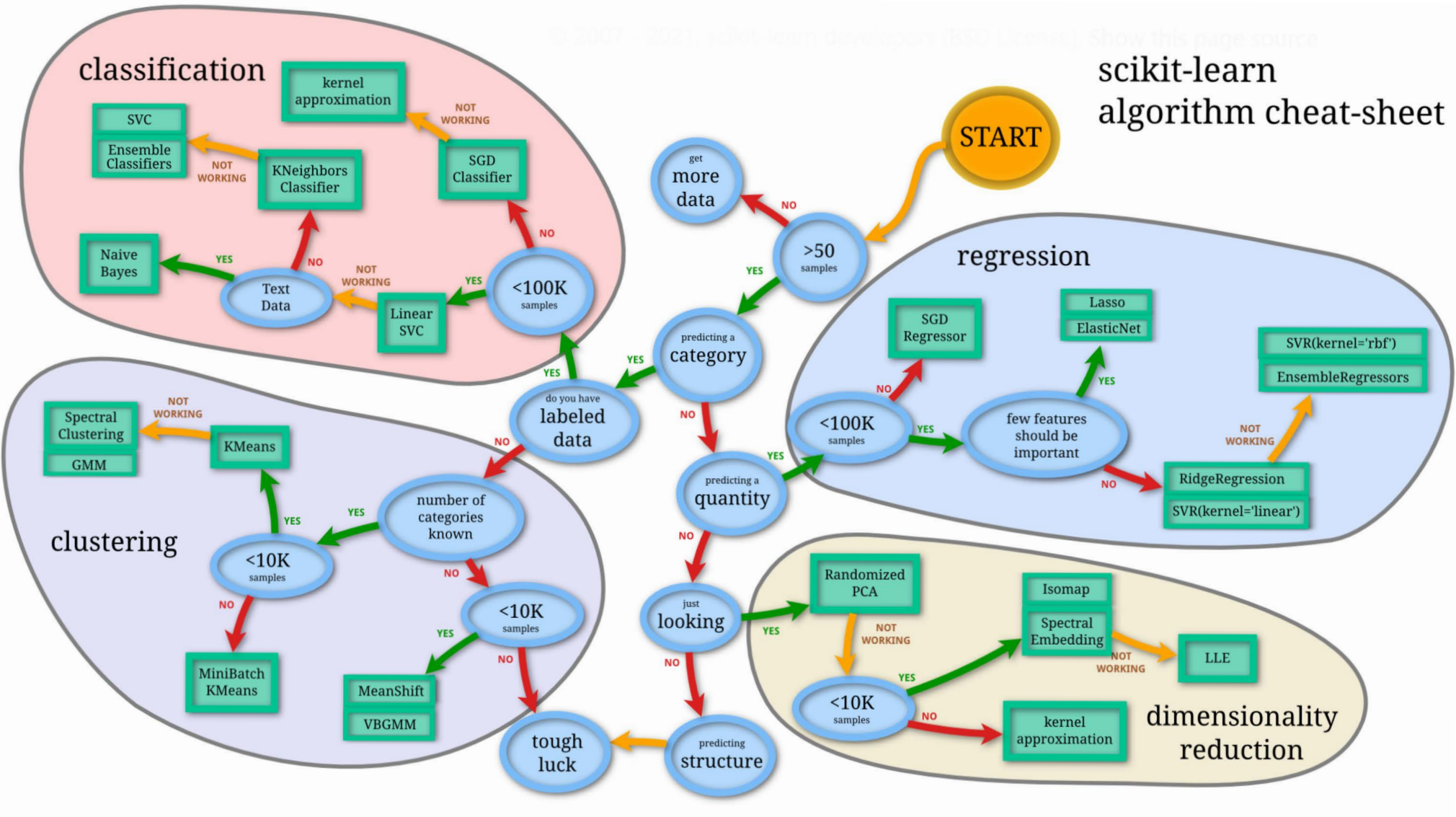
REGRESSION

- **Ordinal regression** — Data in rank ordered categories
- **Poisson regression** — Predicting event counts
- **Fast forest quantile regression** — Predicting a distribution
- **Linear regression** — Fast training, linear model
- **Bayesian linear regression** — Linear model, small data sets
- **Neural network regression** — Accuracy, long training time
- **Decision forest regression** — Accuracy, fast training
- **Boosted decision tree regression** — Accuracy, fast training, large memory footprint

TWO-CLASS CLASSIFICATION

- **Two-class SVM** — >100 features, linear model
- **Two-class averaged perceptron** — Fast training, linear model
- **Two-class logistic regression** — Fast training, linear model
- **Two-class Bayes point machine** — Fast training, linear model

- Accuracy, fast training — **Two-class decision forest**
- Accuracy, fast training, large memory footprint — **Two-class boosted decision tree**
- Accuracy, small memory footprint — **Two-class decision jungle**
- >100 features — **Two-class locally deep SVM**
- Accuracy, long training times — **Two-class neural network**



2. Basic math and probabilities

VIP Refresher: Linear Algebra and Calculus

Afshine Amidi and Shervine Amidi

October 6, 2018

General notations

❒ **Vector** – We note $x \in \mathbb{R}^n$ a vector with n entries, where $x_i \in \mathbb{R}$ is the i^{th} entry:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

❒ **Matrix** – We note $A \in \mathbb{R}^{m \times n}$ a matrix with m rows and n columns, where $A_{i,j} \in \mathbb{R}$ is the entry located in the i^{th} row and j^{th} column:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Remark: the vector x defined above can be viewed as a $n \times 1$ matrix and is more particularly called a column-vector.

❒ **Identity matrix** – The identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix with ones in its diagonal and zero everywhere else:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remark: for all matrices $A \in \mathbb{R}^{n \times n}$, we have $A \times I = I \times A = A$.

❒ **Diagonal matrix** – A diagonal matrix $D \in \mathbb{R}^{n \times n}$ is a square matrix with nonzero values in its diagonal and zero everywhere else:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}$$

Remark: we also note D as $diag(d_1,...,d_n)$.

Matrix operations

❒ **Vector-vector multiplication** – There are two types of vector-vector products:

- inner product: for $x,y \in \mathbb{R}^n$, we have:

$$\boxed{x^T y = \sum_{i=1}^{n} x_i y_i \in \mathbb{R}}$$

- outer product: for $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, we have:

$$\boxed{xy^T = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_n \\ \vdots & & \vdots \\ x_m y_1 & \cdots & x_m y_n \end{pmatrix} \in \mathbb{R}^{m \times n}}$$

❒ **Matrix-vector multiplication** – The product of matrix $A \in \mathbb{R}^{m \times n}$ and vector $x \in \mathbb{R}^n$ is a vector of size $\mathbb{R}^m$, such that:

$$\boxed{Ax = \begin{pmatrix} a_{r,1}^T x \\ \vdots \\ a_{r,m}^T x \end{pmatrix} = \sum_{i=1}^{n} a_{c,i} x_i \in \mathbb{R}^m}$$

where $a_{r,i}^T$ are the vector rows and $a_{c,j}$ are the vector columns of A, and x_i are the entries of x.

❒ **Matrix-matrix multiplication** – The product of matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is a matrix of size $\mathbb{R}^{n \times p}$, such that:

$$\boxed{AB = \begin{pmatrix} a_{r,1}^T b_{c,1} & \cdots & a_{r,1}^T b_{c,p} \\ \vdots & & \vdots \\ a_{r,m}^T b_{c,1} & \cdots & a_{r,m}^T b_{c,p} \end{pmatrix} = \sum_{i=1}^{n} a_{c,i} b_{r,i}^T \in \mathbb{R}^{n \times p}}$$

where $a_{r,i}^T, b_{r,i}^T$ are the vector rows and $a_{c,j}, b_{c,j}$ are the vector columns of A and B respectively.

❒ **Transpose** – The transpose of a matrix $A \in \mathbb{R}^{m \times n}$, noted A^T, is such that its entries are flipped:

$$\boxed{\forall i,j, \quad A_{i,j}^T = A_{j,i}}$$

Remark: for matrices A,B, we have $(AB)^T = B^T A^T$.

❒ **Inverse** – The inverse of an invertible square matrix A is noted A^{-1} and is the only matrix such that:

$$\boxed{AA^{-1} = A^{-1}A = I}$$

Remark: not all square matrices are invertible. Also, for matrices A,B, we have $(AB)^{-1} = B^{-1}A^{-1}$

❒ **Trace** – The trace of a square matrix A, noted $\text{tr}(A)$, is the sum of its diagonal entries:

$$\boxed{\text{tr}(A) = \sum_{i=1}^{n} A_{i,i}}$$

Remark: for matrices A,B, we have $tr(A^T) = tr(A)$ and $tr(AB) = tr(BA)$

❒ **Determinant** – The determinant of a square matrix $A \in \mathbb{R}^{n \times n}$, noted $|A|$ or $\det(A)$ is expressed recursively in terms of $A_{\backslash i,\backslash j}$, which is the matrix A without its i^{th} row and j^{th} column, as follows:

$$\det(A) = |A| = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} |A_{\backslash i, \backslash j}|$$

Remark: A is invertible if and only if $|A| \neq 0$. Also, $|AB| = |A||B|$ and $|A^T| = |A|$.

Matrix properties

❒ **Symmetric decomposition** – A given matrix A can be expressed in terms of its symmetric and antisymmetric parts as follows:

$$A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}}$$

❒ **Norm** – A norm is a function $N : V \longrightarrow [0, +\infty[$ where V is a vector space, and such that for all x,$y \in V$, we have:

- $N(x + y) \leqslant N(x) + N(y)$
- $N(ax) = |a|N(x)$ for a scalar
- if $N(x) = 0$, then $x = 0$

For $x \in V$, the most commonly used norms are summed up in the table below:

Norm	Notation	Definition	Use case
Manhattan, L^1	$\|\|x\|\|_1$	$\sum_{i=1}^{n} \|x_i\|$	LASSO regularization
Euclidean, L^2	$\|\|x\|\|_2$	$\sqrt{\sum_{i=1}^{n} x_i^2}$	Ridge regularization
p-norm, L^p	$\|\|x\|\|_p$	$\left(\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$	Hölder inequality
Infinity, L^∞	$\|\|x\|\|_\infty$	$\max_i \|x_i\|$	Uniform convergence

❒ **Linearly dependence** – A set of vectors is said to be linearly dependent if one of the vectors in the set can be defined as a linear combination of the others.
Remark: if no vector can be written this way, then the vectors are said to be linearly independent.

❒ **Matrix rank** – The rank of a given matrix A is noted $\text{rank}(A)$ and is the dimension of the vector space generated by its columns. This is equivalent to the maximum number of linearly independent columns of A.

❒ **Positive semi-definite matrix** – A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD) and is noted $A \succeq 0$ if we have:

$$A = A^T \quad \text{and} \quad \forall x \in \mathbb{R}^n, \quad x^T A x \geqslant 0$$

Remark: similarly, a matrix A is said to be positive definite, and is noted $A \succ 0$, if it is a PSD matrix which satisfies for all non-zero vector x, $x^T A x > 0$.

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n \times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$Az = \lambda z$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n \times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n \times n}$. By noting $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$, we have:

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

❒ **Singular-value decomposition** – For a given matrix A of dimensions $m \times n$, the singular-value decomposition (SVD) is a factorization technique that guarantees the existence of U $m \times m$ unitary, Σ $m \times n$ diagonal and V $n \times n$ unitary matrices, such that:

$$A = U \Sigma V^T$$

Matrix calculus

❒ **Gradient** – Let $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ be a function and $A \in \mathbb{R}^{m \times n}$ be a matrix. The gradient of f with respect to A is a $m \times n$ matrix, noted $\nabla_A f(A)$, such that:

$$\left(\nabla_A f(A)\right)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}$$

Remark: the gradient of f is only defined when f is a function that returns a scalar.

❒ **Hessian** – Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function and $x \in \mathbb{R}^n$ be a vector. The hessian of f with respect to x is a $n \times n$ symmetric matrix, noted $\nabla_x^2 f(x)$, such that:

$$\left(\nabla_x^2 f(x)\right)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

Remark: the hessian of f is only defined when f is a function that returns a scalar.

❒ **Gradient operations** – For matrices A,B,C, the following gradient properties are worth having in mind:

$$\nabla_A \text{tr}(AB) = B^T \qquad \nabla_{A^T} f(A) = (\nabla_A f(A))^T$$

$$\nabla_A \text{tr}(ABA^T C) = CAB + C^T AB^T \qquad \nabla_A |A| = |A|(A^{-1})^T$$

VIP Refresher: Probabilities and Statistics

Afshine Amidi and Shervine Amidi

August 6, 2018

Introduction to Probability and Combinatorics

❒ **Sample space** – The set of all possible outcomes of an experiment is known as the sample space of the experiment and is denoted by S.

❒ **Event** – Any subset E of the sample space is known as an event. That is, an event is a set consisting of possible outcomes of the experiment. If the outcome of the experiment is contained in E, then we say that E has occurred.

❒ **Axioms of probability** – For each event E, we denote $P(E)$ as the probability of event E occuring. By noting E_1,...,E_n mutually exclusive events, we have the 3 following axioms:

$$(1) \quad \boxed{0 \leqslant P(E) \leqslant 1} \quad (2) \quad \boxed{P(S) = 1} \quad (3) \quad \boxed{P\left(\bigcup_{i=1}^{n} E_i\right) = \sum_{i=1}^{n} P(E_i)}$$

❒ **Permutation** – A permutation is an arrangement of r objects from a pool of n objects, in a given order. The number of such arrangements is given by $P(n,r)$, defined as:

$$\boxed{P(n,r) = \frac{n!}{(n-r)!}}$$

❒ **Combination** – A combination is an arrangement of r objects from a pool of n objects, where the order does not matter. The number of such arrangements is given by $C(n,r)$, defined as:

$$\boxed{C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!}}$$

Remark: we note that for $0 \leqslant r \leqslant n$, we have $P(n,r) \geqslant C(n,r)$.

Conditional Probability

❒ **Bayes' rule** – For events A and B such that $P(B) > 0$, we have:

$$\boxed{P(A|B) = \frac{P(B|A)P(A)}{P(B)}}$$

Remark: we have $P(A \cap B) = P(A)P(B|A) = P(A|B)P(B)$.

❒ **Partition** – Let $\{A_i, i \in [\![1,n]\!]\}$ be such that for all i, $A_i \neq \varnothing$. We say that $\{A_i\}$ is a partition if we have:

$$\boxed{\forall i \neq j, A_i \cap A_j = \emptyset \quad \text{and} \quad \bigcup_{i=1}^{n} A_i = S}$$

Remark: for any event B in the sample space, we have $P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$.

❒ **Extended form of Bayes' rule** – Let $\{A_i, i \in [\![1,n]\!]\}$ be a partition of the sample space. We have:

$$\boxed{P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^{n} P(B|A_i)P(A_i)}}$$

❒ **Independence** – Two events A and B are independent if and only if we have:

$$\boxed{P(A \cap B) = P(A)P(B)}$$

Random Variables

❒ **Random variable** – A random variable, often noted X, is a function that maps every element in a sample space to a real line.

❒ **Cumulative distribution function (CDF)** – The cumulative distribution function F, which is monotonically non-decreasing and is such that $\lim_{x \to -\infty} F(x) = 0$ and $\lim_{x \to +\infty} F(x) = 1$, is defined as:

$$\boxed{F(x) = P(X \leqslant x)}$$

Remark: we have $P(a < X \leqslant B) = F(b) - F(a)$.

❒ **Probability density function (PDF)** – The probability density function f is the probability that X takes on values between two adjacent realizations of the random variable.

❒ **Relationships involving the PDF and CDF** – Here are the important properties to know in the discrete (D) and the continuous (C) cases.

Case	CDF F	PDF f	Properties of PDF
(D)	$F(x) = \sum_{x_i \leqslant x} P(X = x_i)$	$f(x_j) = P(X = x_j)$	$0 \leqslant f(x_j) \leqslant 1$ and $\sum_j f(x_j) = 1$
(C)	$F(x) = \int_{-\infty}^{x} f(y)dy$	$f(x) = \frac{dF}{dx}$	$f(x) \geqslant 0$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$

❒ **Variance** – The variance of a random variable, often noted $\text{Var}(X)$ or σ^2, is a measure of the spread of its distribution function. It is determined as follows:

$$\boxed{\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2}$$

❒ **Standard deviation** – The standard deviation of a random variable, often noted σ, is a measure of the spread of its distribution function which is compatible with the units of the actual random variable. It is determined as follows:

$$\boxed{\sigma = \sqrt{\text{Var}(X)}}$$

❒ **Expectation and Moments of the Distribution** – Here are the expressions of the expected value $E[X]$, generalized expected value $E[g(X)]$, k^{th} moment $E[X^k]$ and characteristic function (ω) for the discrete and continuous cases:

Case	$E[X]$	$E[g(X)]$	$E[X^k]$	(ω)
(D)	$\sum_{i=1}^{n} x_i f(x_i)$	$\sum_{i=1}^{n} g(x_i) f(x_i)$	$\sum_{i=1}^{n} x_i^k f(x_i)$	$\sum_{i=1}^{n} f(x_i) e^{i\omega x_i}$
(C)	$\int_{-\infty}^{+\infty} x f(x) dx$	$\int_{-\infty}^{+\infty} g(x) f(x) dx$	$\int_{-\infty}^{+\infty} x^k f(x) dx$	$\int_{-\infty}^{+\infty} f(x) e^{i\omega x} dx$

Remark: we have $e^{i\omega x} = \cos(\omega x) + i \sin(\omega x)$.

❒ **Revisiting the k^{th} moment** – The k^{th} moment can also be computed with the characteristic function as follows:

$$\boxed{E[X^k] = \frac{1}{i^k} \left[\frac{\partial^k \psi}{\partial \omega^k} \right]_{\omega=0}}$$

❒ **Transformation of random variables** – Let the variables X and Y be linked by some function. By noting f_X and f_Y the distribution function of X and Y respectively, we have:

$$\boxed{f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|}$$

❒ **Leibniz integral rule** – Let g be a function of x and potentially c, and a, b boundaries that may depend on c. We have:

$$\boxed{\frac{\partial}{\partial c} \left(\int_a^b g(x) dx \right) = \frac{\partial b}{\partial c} \cdot g(b) - \frac{\partial a}{\partial c} \cdot g(a) + \int_a^b \frac{\partial g}{\partial c}(x) dx}$$

❒ **Chebyshev's inequality** – Let X be a random variable with expected value μ and standard deviation σ. For $k, \sigma > 0$, we have the following inequality:

$$\boxed{P(|X - \mu| \geqslant k\sigma) \leqslant \frac{1}{k^2}}$$

Jointly Distributed Random Variables

❒ **Conditional density** – The conditional density of X with respect to Y, often noted $f_{X|Y}$, is defined as follows:

$$\boxed{f_{X|Y}(x) = \frac{f_{XY}(x,y)}{f_Y(y)}}$$

❒ **Independence** – Two random variables X and Y are said to be independent if we have:

$$\boxed{f_{XY}(x,y) = f_X(x) f_Y(y)}$$

❒ **Marginal density and cumulative distribution** – From the joint density probability function f_{XY}, we have:

Case	Marginal density	Cumulative function
(D)	$f_X(x_i) = \sum_j f_{XY}(x_i, y_j)$	$F_{XY}(x,y) = \sum_{x_i \leqslant x} \sum_{y_j \leqslant y} f_{XY}(x_i, y_j)$
(C)	$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x,y) dy$	$F_{XY}(x,y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f_{XY}(x', y') dx' dy'$

❒ **Distribution of a sum of independent random variables** – Let $Y = X_1 + ... + X_n$ with $X_1, ..., X_n$ independent. We have:

$$\boxed{\psi_Y(\omega) = \prod_{k=1}^{n} \psi_{X_k}(\omega)}$$

❒ **Covariance** – We define the covariance of two random variables X and Y, that we note σ_{XY}^2 or more commonly $\text{Cov}(X,Y)$, as follows:

$$\boxed{\text{Cov}(X,Y) \triangleq \sigma_{XY}^2 = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y}$$

❒ **Correlation** – By noting σ_X, σ_Y the standard deviations of X and Y, we define the correlation between the random variables X and Y, noted ρ_{XY}, as follows:

$$\boxed{\rho_{XY} = \frac{\sigma_{XY}^2}{\sigma_X \sigma_Y}}$$

Remarks: For any X, Y, we have $\rho_{XY} \in [-1,1]$. If X and Y are independent, then $\rho_{XY} = 0$.

❒ **Main distributions** – Here are the main distributions to have in mind:

Type	Distribution	PDF	(ω)	$E[X]$	$\text{Var}(X)$
(D)	$X \sim \mathcal{B}(n,p)$ Binomial	$P(X=x) = \binom{n}{x} p^x q^{n-x}$ $x \in [\![0,n]\!]$	$(pe^{i\omega} + q)^n$	np	npq
	$X \sim \text{Po}(\mu)$ Poisson	$P(X=x) = \frac{\mu^x}{x!} e^{-\mu}$ $x \in \mathbb{N}$	$e^{\mu(e^{i\omega}-1)}$	μ	μ
(C)	$X \sim \mathcal{U}(a,b)$ Uniform	$f(x) = \frac{1}{b-a}$ $x \in [a,b]$	$\frac{e^{i\omega b} - e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
	$X \sim \mathcal{N}(\mu,\sigma)$ Gaussian	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ $x \in \mathbb{R}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	μ	σ^2
	$X \sim \text{Exp}(\lambda)$ Exponential	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1 - \frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

Parameter estimation

❒ **Random sample** – A random sample is a collection of n random variables $X_1, ..., X_n$ that are independent and identically distributed with X.

❒ **Estimator** – An estimator $\hat{\theta}$ is a function of the data that is used to infer the value of an unknown parameter θ in a statistical model.

❒ **Bias** – The bias of an estimator $\hat{\theta}$ is defined as being the difference between the expected value of the distribution of $\hat{\theta}$ and the true value, i.e.:

$$\boxed{\text{Bias}(\hat{\theta}) = E[\hat{\theta}] - \theta}$$

Remark: an estimator is said to be unbiased when we have $E[\hat{\theta}] = \theta$.

❒ **Sample mean and variance** – The sample mean and the sample variance of a random sample are used to estimate the true mean μ and the true variance σ^2 of a distribution, are noted $\overline{X}$ and s^2 respectively, and are such that:

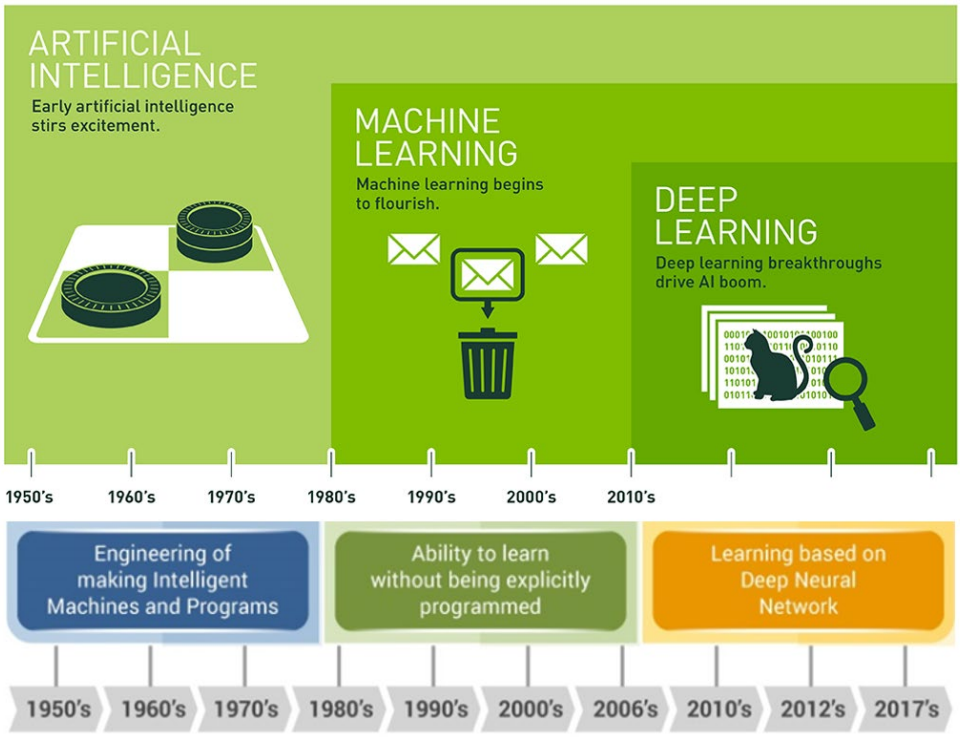
$$\boxed{\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i} \quad \text{and} \quad \boxed{s^2 = \hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2}$$

❒ **Central Limit Theorem** – Let us have a random sample $X_1, ..., X_n$ following a given distribution with mean μ and variance σ^2, then we have:

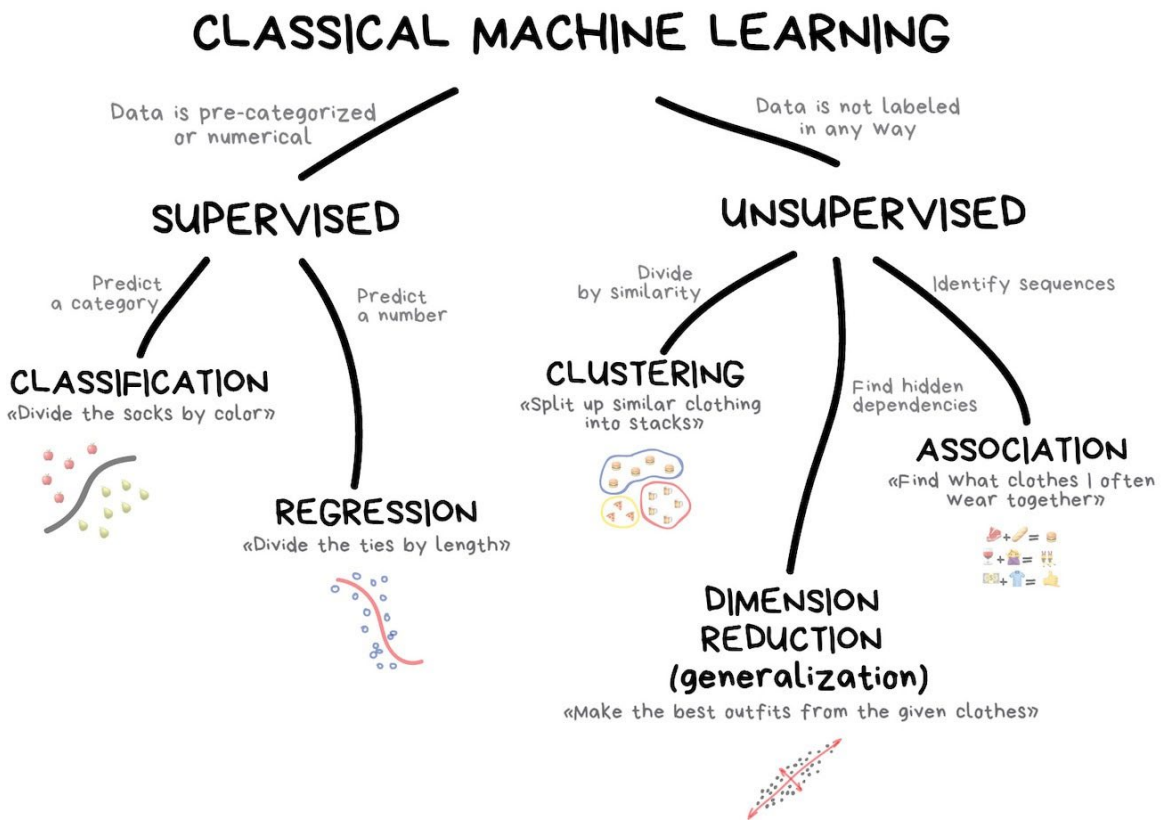
$$\boxed{\overline{X} \underset{n\to+\infty}{\sim} \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)}$$

3. Machine learning

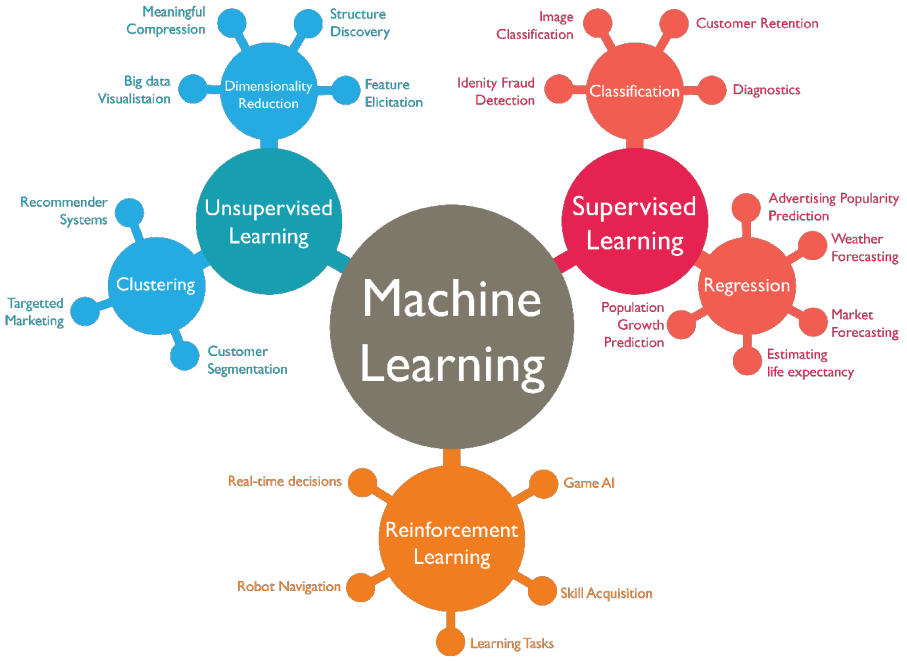
- **History of AI**



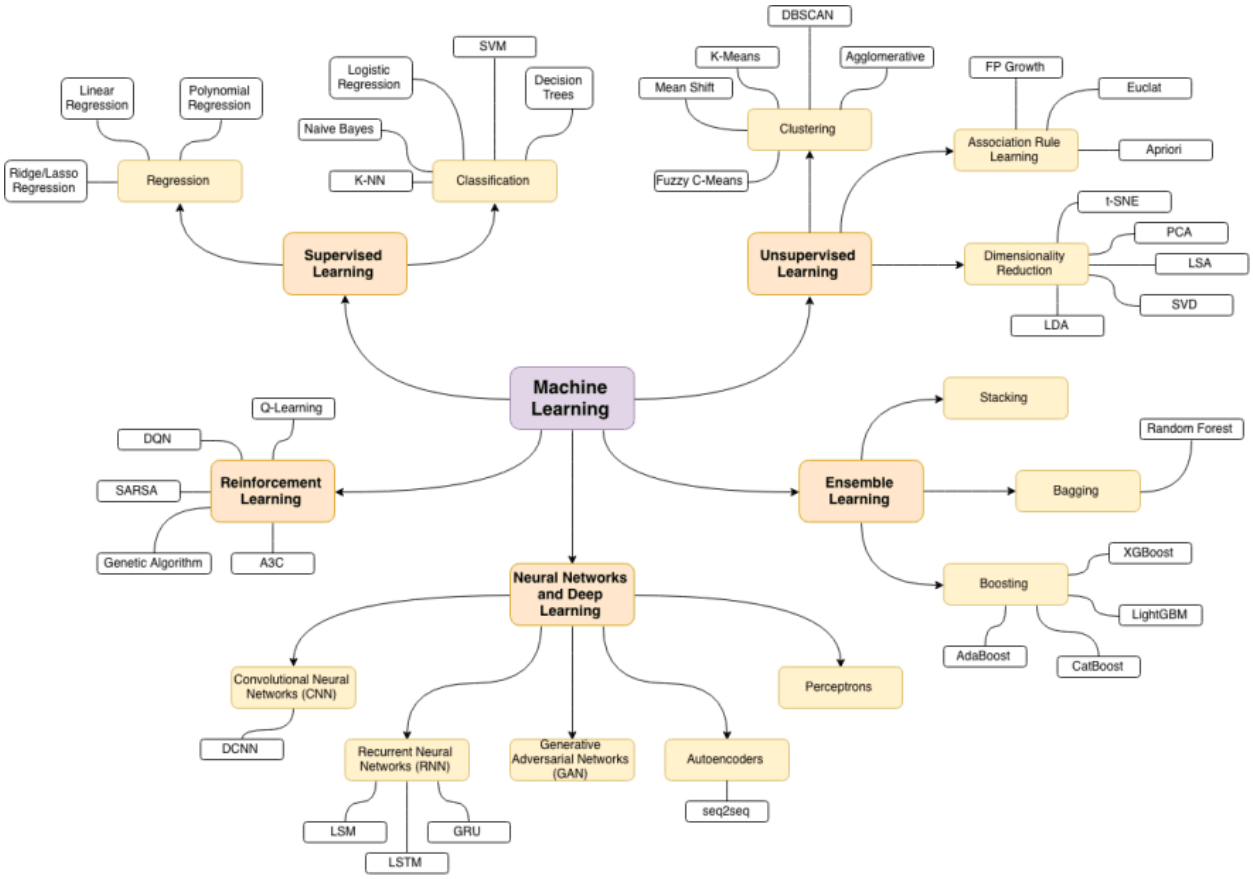
- **Structure of Machine learning**



- **Applications of Machine learning**



- **Top Algorithms of machine and deep learning**



3.1. Supervised learning

VIP Cheatsheet: Supervised Learning

Afshine Amidi and Shervine Amidi

September 9, 2018

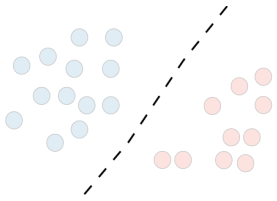
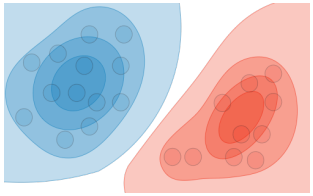
Introduction to Supervised Learning

Given a set of data points $\{x^{(1)},...,x^{(m)}\}$ associated to a set of outcomes $\{y^{(1)},...,y^{(m)}\}$, we want to build a classifier that learns how to predict y from x.

❒ **Type of prediction** – The different types of predictive models are summed up in the table below:

	Regression	**Classifier**
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

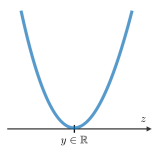
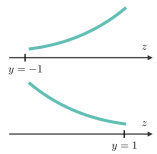
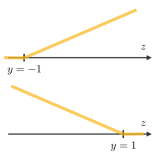
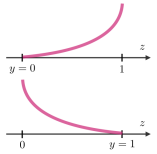
❒ **Type of model** – The different models are summed up in the table below:

	Discriminative model	**Generative model**
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Notations and general concepts

❒ **Hypothesis** – The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$, the model prediction output is $h_\theta(x^{(i)})$.

❒ **Loss function** – A loss function is a function $L:(z,y)\in\mathbb{R}\times Y\longmapsto L(z,y)\in\mathbb{R}$ that takes as inputs the predicted value z corresponding to the real data value y and outputs how different they are. The common loss functions are summed up in the table below:

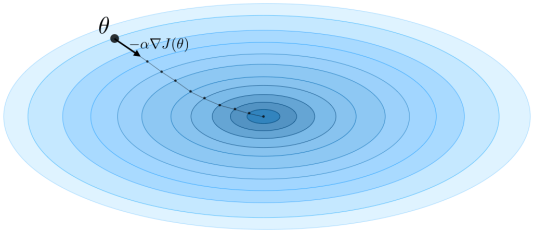
Least squared	**Logistic**	**Hinge**	**Cross-entropy**
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0,1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

❒ **Cost function** – The cost function J is commonly used to assess the performance of a model, and is defined with the loss function L as follows:

$$\boxed{J(\theta)=\sum_{i=1}^{m}L(h_\theta(x^{(i)}),y^{(i)})}$$

❒ **Gradient descent** – By noting $\alpha\in\mathbb{R}$ the learning rate, the update rule for gradient descent is expressed with the learning rate and the cost function J as follows:

$$\boxed{\theta\longleftarrow\theta-\alpha\nabla J(\theta)}$$



Remark: Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.

❒ **Likelihood** – The likelihood of a model $L(\theta)$ given parameters θ is used to find the optimal parameters θ through maximizing the likelihood. In practice, we use the log-likelihood $\ell(\theta)=\log(L(\theta))$ which is easier to optimize. We have:

$$\boxed{\theta^{\text{opt}}=\arg\max_\theta L(\theta)}$$

❒ **Newton's algorithm** – The Newton's algorithm is a numerical method that finds θ such that $\ell'(\theta)=0$. Its update rule is as follows:

$$\boxed{\theta\quad\theta-\frac{\ell'(\theta)}{\ell''(\theta)}}$$

Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:

$$\theta\quad\theta-\left(\nabla^2_\theta\ell(\theta)\right)^{-1}\nabla_\theta\ell(\theta)$$

Linear regression

We assume here that $y|x;\theta \sim \mathcal{N}(\mu,\sigma^2)$

❒ **Normal equations** – By noting X the matrix design, the value of θ that minimizes the cost function is a closed-form solution such that:

$$\boxed{\theta = (X^TX)^{-1}X^Ty}$$

❒ **LMS algorithm** – By noting α the learning rate, the update rule of the Least Mean Squares (LMS) algorithm for a training set of m data points, which is also known as the Widrow-Hoff learning rule, is as follows:

$$\boxed{\forall j, \quad \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^{m} \left[y^{(i)} - h_\theta(x^{(i)})\right] x_j^{(i)}}$$

Remark: the update rule is a particular case of the gradient ascent.

❒ **LWR** – Locally Weighted Regression, also known as LWR, is a variant of linear regression that weights each training example in its cost function by $w^{(i)}(x)$, which is defined with parameter $\tau \in \mathbb{R}$ as:

$$\boxed{w^{(i)}(x) = \exp\left(-\frac{(x^{(i)}-x)^2}{2\tau^2}\right)}$$

Classification and logistic regression

❒ **Sigmoid function** – The sigmoid function g, also known as the logistic function, is defined as follows:

$$\forall z \in \mathbb{R}, \quad \boxed{g(z) = \frac{1}{1+e^{-z}} \in]0,1[}$$

❒ **Logistic regression** – We assume here that $y|x;\theta \sim \text{Bernoulli}(\phi)$. We have the following form:

$$\boxed{\phi = p(y=1|x;\theta) = \frac{1}{1+\exp(-\theta^Tx)} = g(\theta^Tx)}$$

Remark: there is no closed form solution for the case of logistic regressions.

❒ **Softmax regression** – A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes. By convention, we set $\theta_K = 0$, which makes the Bernoulli parameter ϕ_i of each class i equal to:

$$\boxed{\phi_i = \frac{\exp(\theta_i^Tx)}{\sum_{j=1}^{K}\exp(\theta_j^Tx)}}$$

Generalized Linear Models

❒ **Exponential family** – A class of distributions is said to be in the exponential family if it can be written in terms of a natural parameter, also called the canonical parameter or link function, η, a sufficient statistic $T(y)$ and a log-partition function $a(\eta)$ as follows:

$$\boxed{p(y;\eta) = b(y)\exp(\eta T(y) - a(\eta))}$$

Remark: we will often have $T(y) = y$. Also, $\exp(-a(\eta))$ can be seen as a normalization parameter that will make sure that the probabilities sum to one.

Here are the most common exponential distributions summed up in the following table:

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1+\exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^η	$\frac{1}{y!}$
Geometric	$\log(1-\phi)$	y	$\log\left(\frac{e^\eta}{1-e^\eta}\right)$	1

❒ **Assumptions of GLMs** – Generalized Linear Models (GLM) aim at predicting a random variable y as a function fo $x \in \mathbb{R}^{n+1}$ and rely on the following 3 assumptions:

(1) $\boxed{y|x;\theta \sim \text{ExpFamily}(\eta)}$ (2) $\boxed{h_\theta(x) = E[y|x;\theta]}$ (3) $\boxed{\eta = \theta^Tx}$

Remark: ordinary least squares and logistic regression are special cases of generalized linear models.

Support Vector Machines

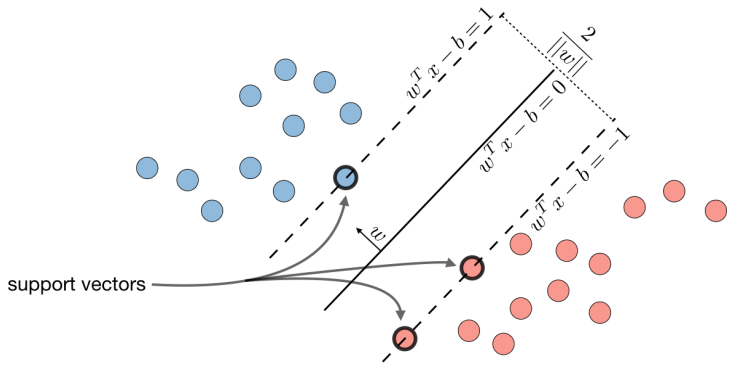
The goal of support vector machines is to find the line that maximizes the minimum distance to the line.

❒ **Optimal margin classifier** – The optimal margin classifier h is such that:

$$\boxed{h(x) = \text{sign}(w^Tx - b)}$$

where $(w,b) \in \mathbb{R}^n \times \mathbb{R}$ is the solution of the following optimization problem:

$$\boxed{\min \frac{1}{2}||w||^2} \quad \text{such that} \quad \boxed{y^{(i)}(w^Tx^{(i)} - b) \geqslant 1}$$



Remark: the line is defined as $\boxed{w^T x - b = 0}$.

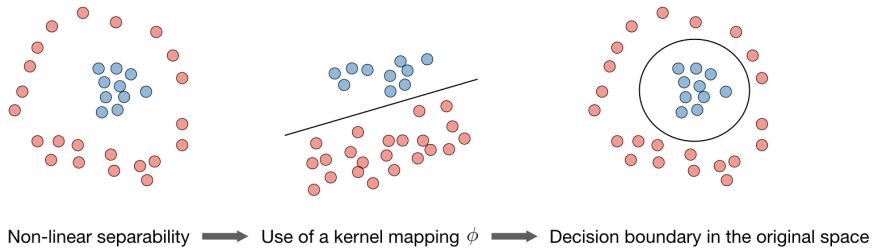
❒ **Hinge loss** – The hinge loss is used in the setting of SVMs and is defined as follows:

$$\boxed{L(z,y) = [1-yz]_+ = \max(0,1-yz)}$$

❒ **Kernel** – Given a feature mapping ϕ, we define the kernel K to be defined as:

$$\boxed{K(x,z) = \phi(x)^T \phi(z)}$$

In practice, the kernel K defined by $K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$ is called the Gaussian kernel and is commonly used.



Remark: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ, which is often very complicated. Instead, only the values $K(x,z)$ are needed.

❒ **Lagrangian** – We define the Lagrangian $\mathcal{L}(w,b)$ as follows:

$$\boxed{\mathcal{L}(w,b) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)}$$

Remark: the coefficients β_i are called the Lagrange multipliers.

Generative Learning

A generative model first tries to learn how the data is generated by estimating $P(x|y)$, which we can then use to estimate $P(y|x)$ by using Bayes' rule.

Gaussian Discriminant Analysis

❒ **Setting** – The Gaussian Discriminant Analysis assumes that y and $x|y=0$ and $x|y=1$ are such that:

$$\boxed{y \sim \text{Bernoulli}(\phi)}$$

$$\boxed{x|y=0 \sim \mathcal{N}(\mu_0,\Sigma)} \quad \text{and} \quad \boxed{x|y=1 \sim \mathcal{N}(\mu_1,\Sigma)}$$

❒ **Estimation** – The following table sums up the estimates that we find when maximizing the likelihood:

$\widehat{\phi}$	$\widehat{\mu_j}$ $(j=0,1)$	$\widehat{\Sigma}$
$\frac{1}{m}\sum_{i=1}^{m} 1_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}}$	$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T$

Naive Bayes

❒ **Assumption** – The Naive Bayes model supposes that the features of each data point are all independent:

$$\boxed{P(x|y) = P(x_1,x_2,...|y) = P(x_1|y)P(x_2|y)... = \prod_{i=1}^{n} P(x_i|y)}$$

❒ **Solutions** – Maximizing the log-likelihood gives the following solutions, with $k \in \{0,1\}$, $l \in [\![1,L]\!]$

$$\boxed{P(y=k) = \frac{1}{m} \times \#\{j|y^{(j)}=k\}} \quad \text{and} \quad \boxed{P(x_i=l|y=k) = \frac{\#\{j|y^{(j)}=k \text{ and } x_i^{(j)}=l\}}{\#\{j|y^{(j)}=k\}}}$$

Remark: Naive Bayes is widely used for text classification and spam detection.

Tree-based and ensemble methods

These methods can be used for both regression and classification problems.

❒ **CART** – Classification and Regression Trees (CART), commonly known as decision trees, can be represented as binary trees. They have the advantage to be very interpretable.

❒ **Random forest** – It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

Remark: random forests are a type of ensemble methods.

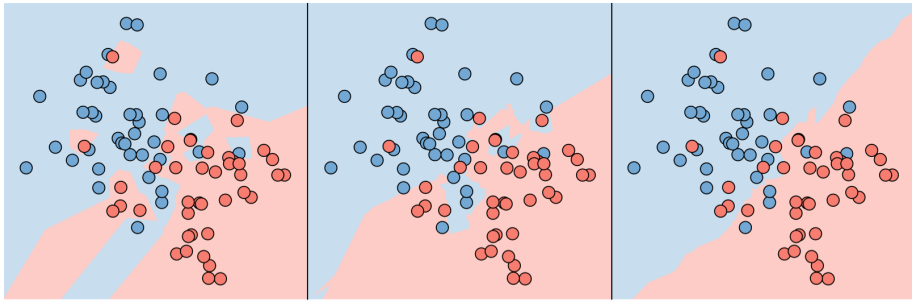
❒ **Boosting** – The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are summed up in the table below:

Adaptive boosting	Gradient boosting
- High weights are put on errors to improve at the next boosting step - Known as Adaboost	- Weak learners trained on remaining errors

Other non-parametric approaches

❒ **k-nearest neighbors** – The k-nearest neighbors algorithm, commonly known as k-NN, is a non-parametric approach where the response of a data point is determined by the nature of its k neighbors from the training set. It can be used in both classification and regression settings.

Remark: The higher the parameter k, the higher the bias, and the lower the parameter k, the higher the variance.

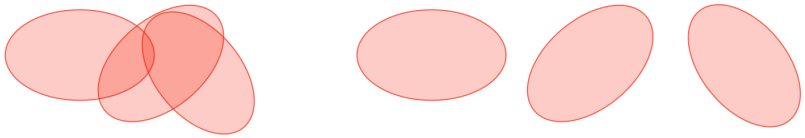


$k = 1$ $k = 3$ $k = 11$

Learning Theory

❒ **Union bound** – Let $A_1, ..., A_k$ be k events. We have:

$$\boxed{P(A_1 \cup ... \cup A_k) \leqslant P(A_1) + ... + P(A_k)}$$



$A_1 \cup A_2 \cup A_3$ A_1 A_2 A_3

❒ **Hoeffding inequality** – Let $Z_1, .., Z_m$ be m iid variables drawn from a Bernoulli distribution of parameter ϕ. Let $\widehat{\phi}$ be their sample mean and $\gamma > 0$ fixed. We have:

$$\boxed{P(|\phi - \widehat{\phi}| > \gamma) \leqslant 2\exp(-2\gamma^2 m)}$$

Remark: this inequality is also known as the Chernoff bound.

❒ **Training error** – For a given classifier h, we define the training error $\widehat{\epsilon}(h)$, also known as the empirical risk or empirical error, to be as follows:

$$\boxed{\widehat{\epsilon}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{\{h(x^{(i)}) \neq y^{(i)}\}}}$$

❒ **Probably Approximately Correct (PAC)** – PAC is a framework under which numerous results on learning theory were proved, and has the following set of assumptions:

- the training and testing sets follow the same distribution
- the training examples are drawn independently

❒ **Shattering** – Given a set $S = \{x^{(1)},...,x^{(d)}\}$, and a set of classifiers $\mathcal{H}$, we say that $\mathcal{H}$ shatters S if for any set of labels $\{y^{(1)}, ..., y^{(d)}\}$, we have:

$$\boxed{\exists h \in \mathcal{H}, \quad \forall i \in [\![1,d]\!], \quad h(x^{(i)}) = y^{(i)}}$$

❒ **Upper bound theorem** – Let $\mathcal{H}$ be a finite hypothesis class such that $|\mathcal{H}| = k$ and let δ and the sample size m be fixed. Then, with probability of at least $1 - \delta$, we have:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + 2\sqrt{\frac{1}{2m}\log\left(\frac{2k}{\delta}\right)}}$$

❒ **VC dimension** – The Vapnik-Chervonenkis (VC) dimension of a given infinite hypothesis class $\mathcal{H}$, noted $\text{VC}(\mathcal{H})$ is the size of the largest set that is shattered by $\mathcal{H}$.

Remark: the VC dimension of $\mathcal{H} = \{$set of linear classifiers in 2 dimensions$\}$ is 3.

❒ **Theorem (Vapnik)** – Let $\mathcal{H}$ be given, with $\text{VC}(\mathcal{H}) = d$ and m the number of training examples. With probability at least $1 - \delta$, we have:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + O\left(\sqrt{\frac{d}{m}\log\left(\frac{m}{d}\right) + \frac{1}{m}\log\left(\frac{1}{\delta}\right)}\right)}$$

VIP Cheatsheet: Machine Learning Tips

Afshine Amidi and Shervine Amidi

September 9, 2018

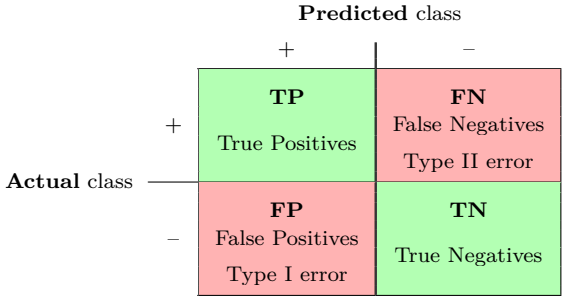
Metrics

Given a set of data points $\{x^{(1)}, ..., x^{(m)}\}$, where each $x^{(i)}$ has n features, associated to a set of outcomes $\{y^{(1)}, ..., y^{(m)}\}$, we want to assess a given classifier that learns how to predict y from x.

Classification

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

❒ **Confusion matrix** – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:



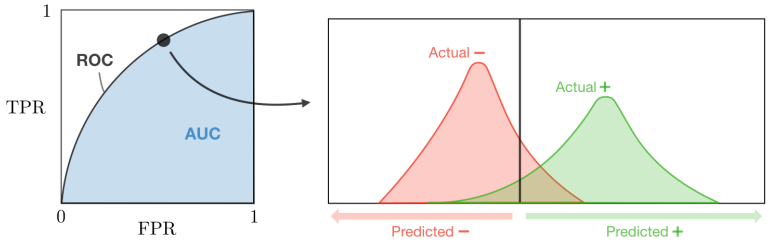
❒ **Main metrics** – The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$	Overall performance of model
Precision	$\frac{\text{TP}}{\text{TP}+\text{FP}}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Coverage of actual positive sample
Specificity	$\frac{\text{TN}}{\text{TN}+\text{FP}}$	Coverage of actual negative sample
F1 score	$\frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}}$	Hybrid metric useful for unbalanced classes

❒ **ROC** – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Recall, sensitivity
False Positive Rate FPR	$\frac{\text{FP}}{\text{TN}+\text{FP}}$	1-specificity

❒ **AUC** – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Regression

❒ **Basic metrics** – Given a regression model f, the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$\text{SS}_{\text{tot}} = \sum_{i=1}^{m}(y_i - \overline{y})^2$	$\text{SS}_{\text{reg}} = \sum_{i=1}^{m}(f(x_i) - \overline{y})^2$	$\text{SS}_{\text{res}} = \sum_{i=1}^{m}(y_i - f(x_i))^2$

❒ **Coefficient of determination** – The coefficient of determination, often noted R^2 or r^2, provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

❒ **Main metrics** – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{\text{SS}_{\text{res}} + 2(n+1)\widehat{\sigma^2}}{m}$	$2\big[(n+2) - \log(L)\big]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

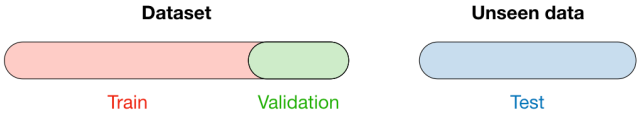
where L is the likelihood and $\widehat{\sigma}^2$ is an estimate of the variance associated with each response.

Model selection

❒ **Vocabulary** – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

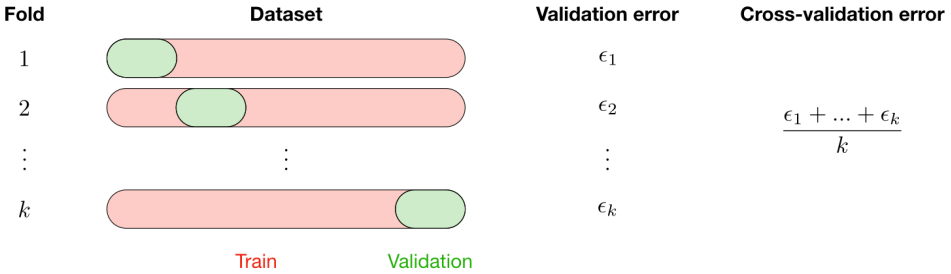
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



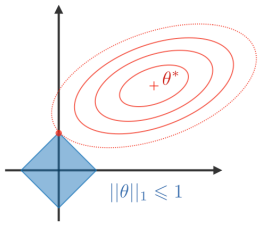
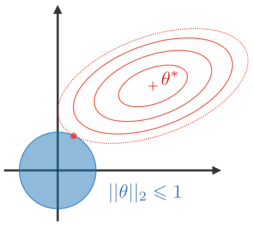
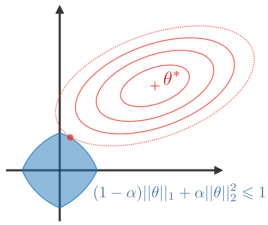
❒ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

k-fold	Leave-p-out
- Training on $k-1$ folds and assessment on the remaining one - Generally $k=5$ or 10	- Training on $n-p$ observations and assessment on the p remaining ones - Case $p=1$ is called leave-one-out

The most commonly used method is called k-fold cross-validation and splits the training data into k folds to validate the model on one fold while training the model on the $k-1$ other folds, all of this k times. The error is then averaged over the k folds and is named cross-validation error.



❒ **Regularization** – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$...+\lambda\|\|\theta\|\|_1$ $\lambda\in\mathbb{R}$	$...+\lambda\|\|\theta\|\|_2^2$ $\lambda\in\mathbb{R}$	$...+\lambda\left[(1-\alpha)\|\|\theta\|\|_1+\alpha\|\|\theta\|\|_2^2\right]$ $\lambda\in\mathbb{R}$, $\alpha\in[0,1]$

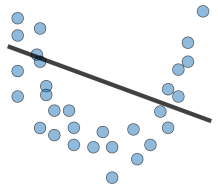
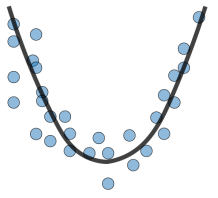
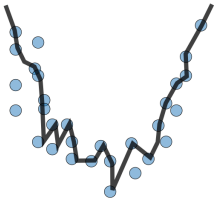
❒ **Model selection** – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

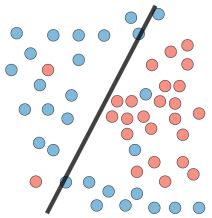
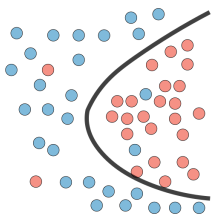
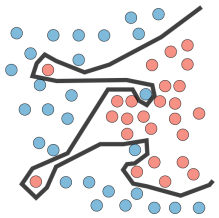
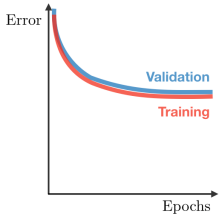
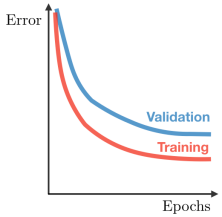
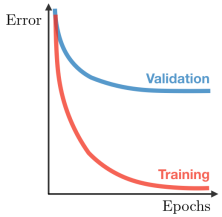
Diagnostics

❒ **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

❒ **Variance** – The variance of a model is the variability of the model prediction for given data points.

❒ **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			

Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

❒ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

❒ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

Cheat Sheet – Regression Analysis

What is Regression Analysis?
Fitting a function f(.) to datapoints y_i=f(x_i) under some error function. Based on the estimated function and error, we have the following types of regression

1. Linear Regression:
Fits a line minimizing the sum of mean-squared error for each datapoint.

$$min_\beta \sum_i \|y_i - f_\beta^{linear}(x_i)\|^2$$
$$f_\beta^{linear}(x_i) = \beta_0 + \beta_1 x_i$$

2. Polynomial Regression:
Fits a polynomial of order k (k+1 unknowns) minimizing the sum of mean-squared error for each datapoint.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta^{poly}(x_i)\|^2$$
$$f_\beta^{poly}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_k x_i^k$$

3. Bayesian Regression:
For each datapoint, fits a gaussian distribution by minimizing the mean-squared error. As the number of data points x_i increases, it converges to point estimates i.e. $n \to \infty, \sigma^2 \to 0$

$$min_\beta \sum_i \|y_i - \mathcal{N}\left(f_\beta(x_i), \sigma^2\right)\|^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$
$$\mathcal{N}\left(\mu, \sigma^2\right) \to \text{Gaussian with mean } \mu \text{ and variance } \sigma^2$$

4. Ridge Regression:
Can fit either a line, or polynomial minimizing the sum of mean-squared error for each datapoint and the weighted L2 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} \beta_j^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$

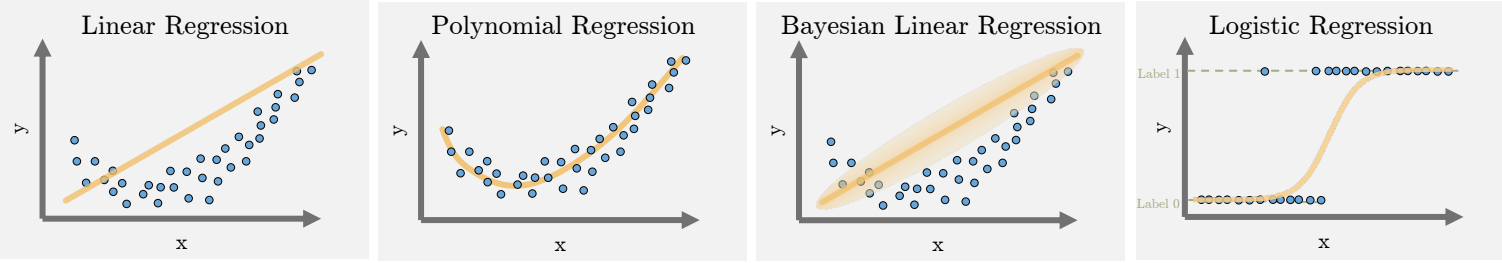
5. LASSO Regression:
Can fit either a line, or polynomial minimizing the the sum of mean-squared error for each datapoint and the weighted L1 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} |\beta_j|$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$

6. Logistic Regression:
Can fit either a line, or polynomial with sigmoid activation minimizing the binary cross-entropy loss for each datapoint. The labels y are binary class labels.

$$min_\beta \sum_i -y_i log\left(\sigma\left(f_\beta(x_i)\right)\right) - (1 - y_i) log\left(1 - \sigma\left(f_\beta(x_i)\right)\right)$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Visual Representation:



Summary:

	What does it fit?	Estimated function	Error Function
Linear	A line in n dimensions	$f_\beta^{linear}(x_i) = \beta_0 + \beta_1 x_i$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$
Polynomial	A polynomial of order k	$f_\beta^{poly}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$
Bayesian Linear	Gaussian distribution for each point	$\mathcal{N}\left(f_\beta(x_i), \sigma^2\right)$	$\sum_i \|y_i - \mathcal{N}\left(f_\beta(x_i), \sigma^2\right)\|^2$
Ridge	Linear/polynomial	$f_\beta^{poly}(x_i)$ or $f_\beta^{linear}(x_i)$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{n} \beta_j^2$
LASSO	Linear/polynomial	$f_\beta^{poly}(x_i)$ or $f_\beta^{linear}(x_i)$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{n} \|\beta_j\|$
Logistic	Linear/polynomial with sigmoid	$\sigma(f_\beta(x_i))$	$min_\beta \sum_i -y_i log\left(\sigma\left(f_\beta(x_i)\right)\right) - (1 - y_i) log\left(1 - \sigma\left(f_\beta(x_i)\right)\right)$

Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

Cheat Sheet – Regularization in ML

What is Regularization in ML?

- Regularization is an approach to address over-fitting in ML.
- Overfitted model fails to generalize estimations on test data
- When the underlying model to be learned is low bias/high variance, or when we have small amount of data, the estimated model is prone to over-fitting.
- Regularization reduces the variance of the model

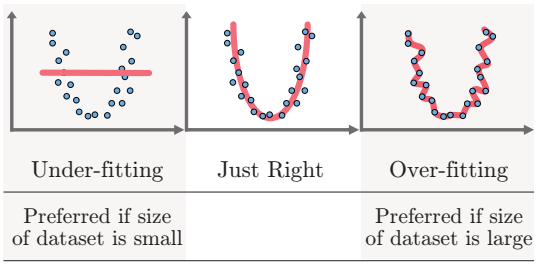


Figure 1. Overfitting

Types of Regularization:

1. Modify the loss function:

- **L2 Regularization:** Prevents the weights from getting too large (defined by L2 norm). Larger the weights, more complex the model is, more chances of overfitting.

$$loss = error(y, \hat{y}) + \lambda \sum_j \beta_j^2 \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

- **L1 Regularization:** Prevents the weights from getting too large (defined by L1 norm). Larger the weights, more complex the model is, more chances of overfitting. L1 regularization introduces sparsity in the weights. It forces more weights to be zero, than reducing the the average magnitude of all weights

$$loss = error(y, \hat{y}) + \lambda \sum_j |\beta_j| \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

- **Entropy:** Used for the models that output probability. Forces the probability distribution towards uniform distribution.

$$loss = error(p, \hat{p}) - \lambda \sum_i \hat{p}_i log(\hat{p}_i) \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

2. Modify data sampling:

- **Data augmentation:** Create more data from available data by randomly cropping, dilating, rotating, adding small amount of noise etc.
- **K-fold Cross-validation:** Divide the data into k groups. Train on (k-1) groups and test on 1 group. Try all k possible combinations.

3. Change training approach:

- **Injecting noise:** Add random noise to the weights when they are being learned. It pushes the model to be relatively insensitive to small variations in the weights, hence regularization
- **Dropout:** Generally used for neural networks. Connections between consecutive layers are randomly dropped based on a dropout-ratio and the remaining network is trained in the current iteration. In the next iteration, another set of random connections are dropped.

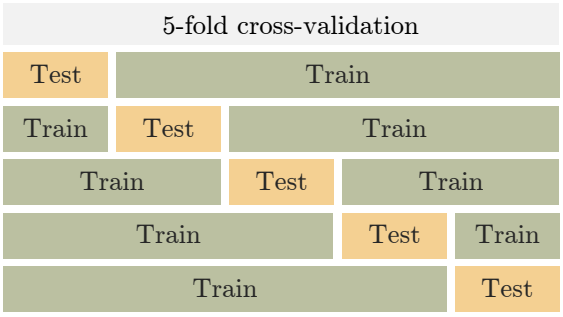


Figure 2. K-fold CV

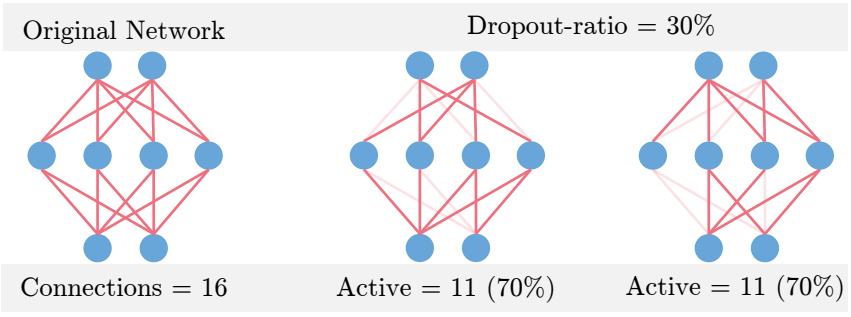


Figure 3. Drop-out

Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

Cheat Sheet – Bias-Variance Tradeoff

What is Bias?

$$bias = \mathbb{E}[f'(x)] - f(x)$$

- Error between average model prediction and ground truth
- The bias of the estimated function tells us the capacity of the underlying model to predict the values

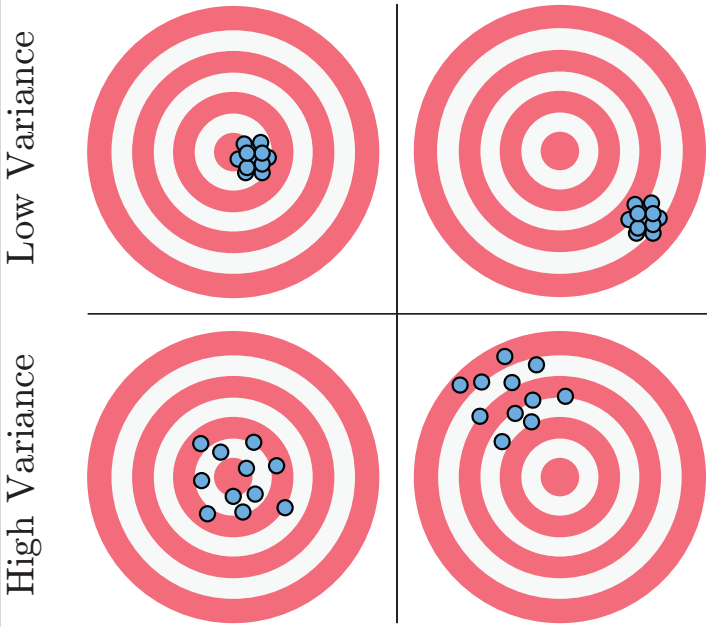
What is Variance?

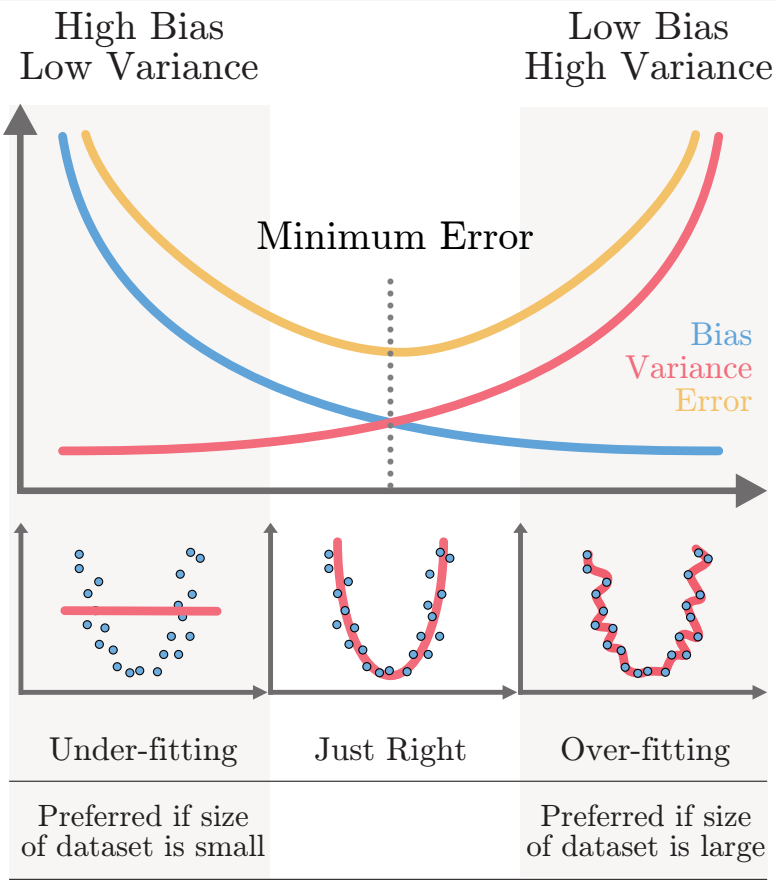
$$variance = \mathbb{E}\left[\left(f'(x) - \mathbb{E}[f'(x)]\right)^2\right]$$

- Average variability in the model prediction for the given dataset
- The variance of the estimated function tells you how much the function can adjust to the change in the dataset

High Bias
- → Overly-simplified Model
- → Under-fitting
- → High error on both test and train data

High Variance
- → Overly-complex Model
- → Over-fitting
- → Low error on train data and high on test
- → Starts modelling the noise in the input





Bias variance Trade-off

- Increasing bias (not always) reduces variance and vice-versa
- Error = bias^2 + variance +irreducible error
- The best model is where the error is reduced.
- Compromise between bias and variance

Cheat Sheet – Bayes Theorem and Classifier

What is Bayes' Theorem?

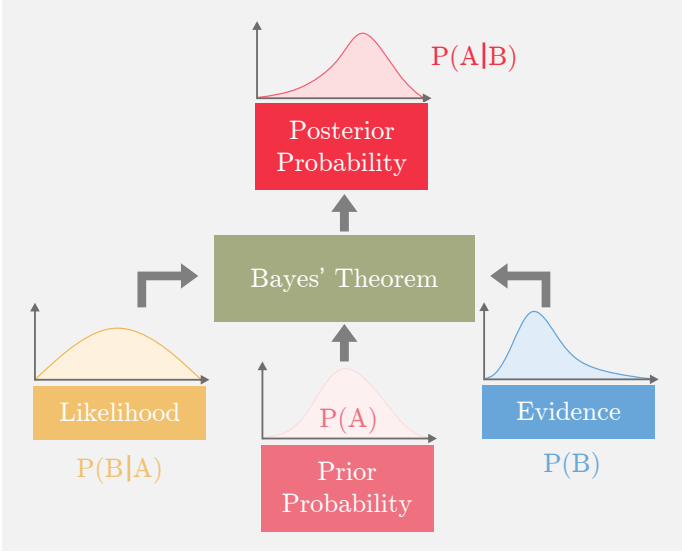
- Describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

$$P(A|B) = \frac{P(B|A)(likelihood) \times P(A)(prior)}{P(B)(evidence)}$$

- How the probability of an event changes when we have knowledge of another event

P(A) ⟶ P(A|B)

↳ Usually, a better estimate than P(A)



Example

- Probability of fire P(F) = 1%
- Probability of smoke P(S) = 10%
- Prob of smoke given there is a fire P(S|F) = 90%
- What is the probability that there is a fire given we see a smoke P(F|S)?

$$P(F|S) = \frac{P(S|F) \times P(F)}{P(S)} = \frac{0.9 \times 0.01}{0.1} = 9\%$$

Maximum Aposteriori Probability (MAP) Estimation

The MAP estimate of the random variable y, given that we have observed iid (x_1, x_2, x_3, …), is given by. We try to accommodate our prior knowledge when estimating.

$$\hat{y}_{MAP} = argmax_y \;\; P(y) \prod_i P(x_i|y)$$

y that maximizes the product of prior and likelihood

Maximum Likelihood Estimation (MLE)

The MAP estimate of the random variable y, given that we have observed iid (x_1, x_2, x_3, …), is given by. We assume we don't have any prior knowledge of the quantity being estimated.

$$\hat{y}_{MLE} = argmax_y \prod_i P(x_i|y)$$

y that maximizes only the likelihood

MLE is a special case of MAP where our prior is uniform (all values are equally likely)

Naïve Bayes' Classifier (Instantiation of MAP as classifier)

Suppose we have two classes, $y=y_1$ and $y=y_2$. Say we have more than one evidence/features (x_1, x_2, x_3, …), using Bayes' theorem

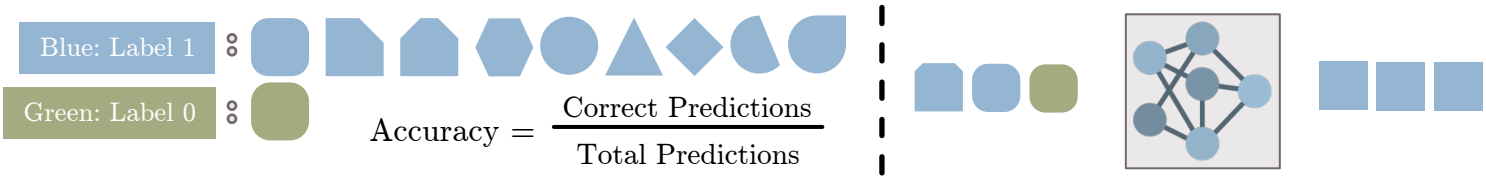
$$P(y|x_1, x_2, x_3, \ldots) = \frac{P(x_1, x_2, x_3, \ldots|y) \times P(y)}{P(x_1, x_2, x_3, \ldots)}$$

Naïve Bayes' theorem assumes the features (x_1, x_2, …) are i.i.d. i.e $P(x_1, x_2, x_3, \ldots|y) = \prod_i P(x_i|y)$

$$P(y|x_1, x_2, x_3, \ldots) = \prod_i P(x_i|y) \frac{P(y)}{P(x_1, x_2, x_3, \ldots)}$$

$$\hat{y} = y_1 \; if \; \frac{P(y_1|x_1, x_2, x_3, \ldots)}{P(y_2|x_1, x_2, x_3, \ldots)} > 1 \;\; else \; \hat{y} = y_2$$

Cheat Sheet – Imbalanced Data in Classification

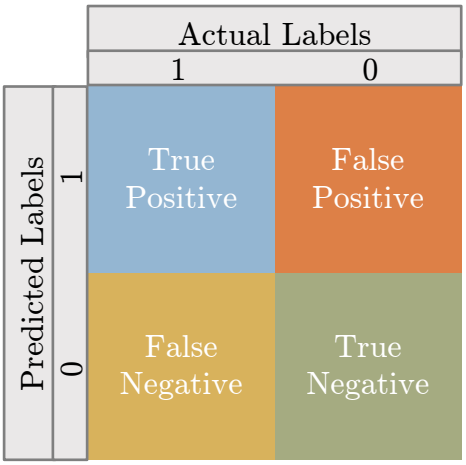


Classifier that always predicts label blue yields prediction accuracy of 90%

Accuracy doesn't always give the correct insight about your trained model

Accuracy: %age correct prediction	Correct prediction over total predictions	One value for entire network
Precision: Exactness of model	From the detected cats, how many were actually cats	Each class/label has a value
Recall: Completeness of model	Correctly detected cats over total cats	Each class/label has a value
F1 Score: Combines Precision/Recall	Harmonic mean of Precision and Recall	Each class/label has a value

Performance metrics associated with Class 1



(Is your prediction correct?) (What did you predict)

True Negative

(Your prediction is correct) (You predicted 0)

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1 score} = 2x \frac{(\text{Prec} \times \text{Rec})}{(\text{Prec} + \text{Rec})}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

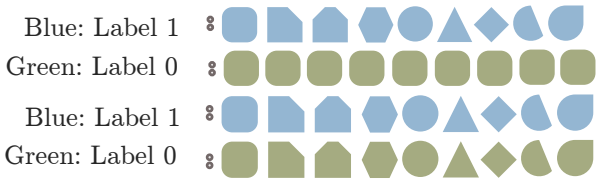
$$\text{False +ve rate} = \frac{FP}{TN + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{Recall, Sensitivity} = \text{True +ve rate} = \frac{TP}{TP + FN}$$

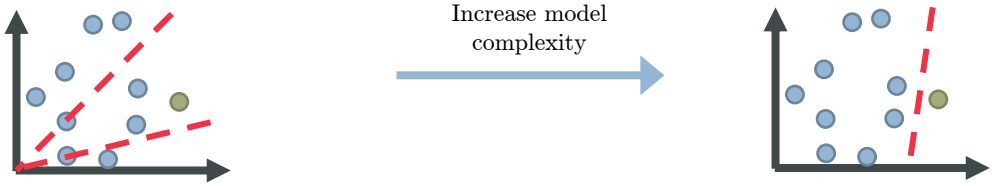
Possible solutions

1. **Data Replication:** Replicate the available data until the number of samples are comparable
2. **Synthetic Data:** Images: Rotate, dilate, crop, add noise to existing input images and create new data
3. **Modified Loss:** Modify the loss to reflect greater error when misclassifying smaller sample set



$$loss = a * loss_{green} + b * loss_{blue} \qquad a > b$$

4. **Change the algorithm:** Increase the model/algorithm complexity so that the two classes are perfectly separable (Con: Overfitting)



No straight line (y=ax) passing through origin can perfectly separate data. **Best solution:** line y=0, predict all labels blue

Straight line (y=ax+**b**) can perfectly separate data. Green class will no longer be predicted as blue

Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

Cheat Sheet – Ensemble Learning in ML

What is Ensemble Learning? Wisdom of the crowd

Combine multiple weak models/learners into one predictive model to reduce bias, variance and/or improve accuracy.

Types of Ensemble Learning: N number of weak learners

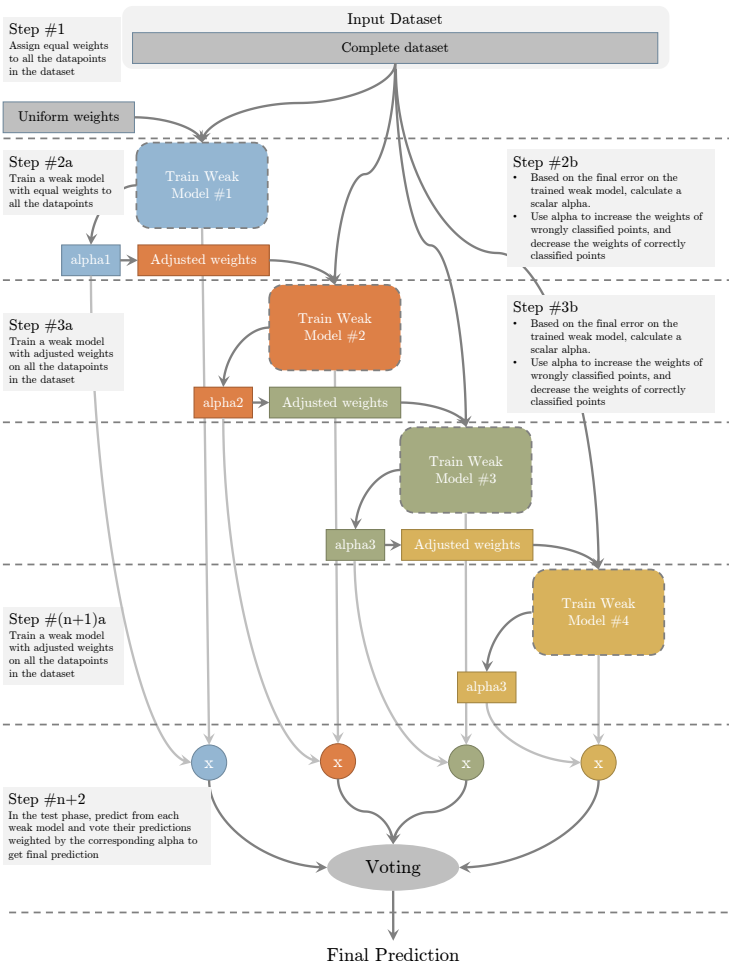
1.Bagging: Trains N different weak models (usually of same types – homogenous) with N non-overlapping subset of the input dataset in parallel. In the test phase, each model is evaluated. The label with the greatest number of predictions is selected as the prediction. Bagging methods reduces variance of the prediction

2.Boosting: Trains N different weak models (usually of same types – homogenous) with the complete dataset in a sequential order. The datapoints wrongly classified with previous weak model is provided more weights to that they can be classified by the next weak leaner properly. In the test phase, each model is evaluated and based on the test error of each weak model, the prediction is weighted for voting. Boosting methods decreases the bias of the prediction.

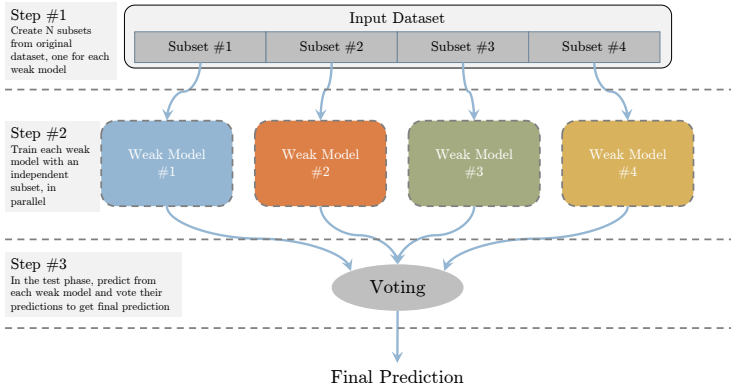
3.Stacking: Trains N different weak models (usually of different types – heterogenous) with one of the two subsets of the dataset in parallel. Once the weak learners are trained, they are used to trained a meta learner to combine their predictions and carry out final prediction using the other subset. In test phase, each model predicts its label, these set of labels are fed to the meta learner which generates the final prediction.

The block diagrams, and comparison table for each of these three methods can be seen below.

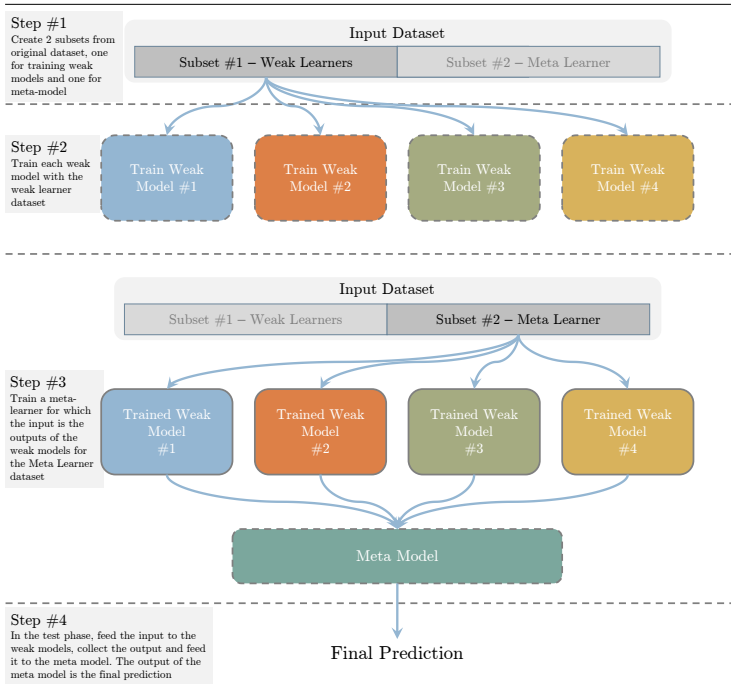
Ensemble Method – Boosting



Ensemble Method – Bagging



Ensemble Method – Stacking



Parameter	Bagging	Boosting	Stacking
Focuses on	Reducing variance	Reducing bias	Improving accuracy
Nature of weak learners is	Homogenous	Homogenous	Heterogenous
Weak learners are aggregated by	Simple voting	Weighted voting	Learned voting (meta-learner)

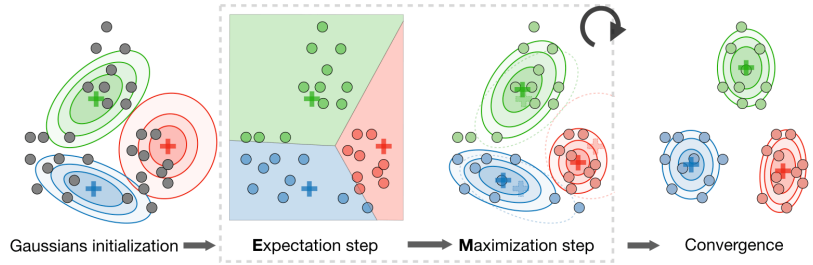
Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

3.2. Unsupervised learning

VIP Cheatsheet: Unsupervised Learning

Afshine Amidi and Shervine Amidi

September 9, 2018



Introduction to Unsupervised Learning

❒ **Motivation** – The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x^{(1)},...,x^{(m)}\}$.

❒ **Jensen's inequality** – Let f be a convex function and X a random variable. We have the following inequality:

$$\boxed{E[f(X)] \geqslant f(E[X])}$$

Expectation-Maximization

❒ **Latent variables** – Latent variables are hidden/unobserved variables that make estimation problems difficult, and are often denoted z. Here are the most common settings where there are latent variables:

Setting	Latent variable z	$x\|z$	Comments
Mixture of k Gaussians	Multinomial(ϕ)	$\mathcal{N}(\mu_j,\Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Factor analysis	$\mathcal{N}(0,I)$	$\mathcal{N}(\mu+\Lambda z,\psi)$	$\mu_j \in \mathbb{R}^n$

❒ **Algorithm** – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step: Evaluate the posterior probability $Q_i(z^{(i)})$ that each data point $x^{(i)}$ came from a particular cluster $z^{(i)}$ as follows:

$$\boxed{Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)};\theta)}$$

- M-step: Use the posterior probabilities $Q_i(z^{(i)})$ as cluster specific weights on data points $x^{(i)}$ to separately re-estimate each cluster model as follows:

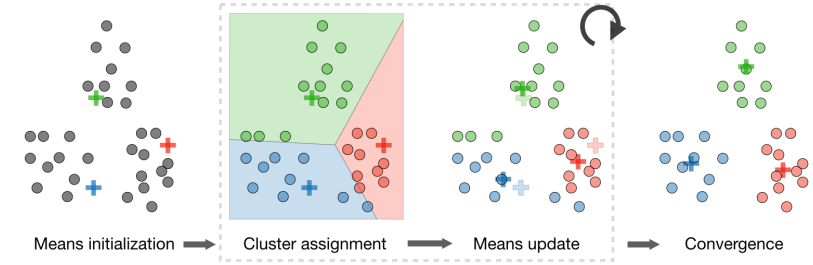
$$\boxed{\theta_i = \underset{\theta}{\text{argmax}} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right) dz^{(i)}}$$

k-means clustering

We note $c^{(i)}$ the cluster of data point i and μ_j the center of cluster j.

❒ **Algorithm** – After randomly initializing the cluster centroids $\mu_1,\mu_2,...,\mu_k \in \mathbb{R}^n$, the k-means algorithm repeats the following step until convergence:

$$\boxed{c^{(i)} = \underset{j}{\arg\min}||x^{(i)}-\mu_j||^2} \quad \text{and} \quad \boxed{\mu_j = \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}}$$



❒ **Distortion function** – In order to see if the algorithm converges, we look at the distortion function defined as follows:

$$\boxed{J(c,\mu) = \sum_{i=1}^m ||x^{(i)} - \mu_{c^{(i)}}||^2}$$

Hierarchical clustering

❒ **Algorithm** – It is a clustering algorithm with an agglomerative hierarchical approach that build nested clusters in a successive manner.

❒ **Types** – There are different sorts of hierarchical clustering algorithms that aims at optimizing different objective functions, which is summed up in the table below:

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

Clustering assessment metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

❒ **Silhouette coefficient** – By noting a and b the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient s for a single sample is defined as follows:

$$\boxed{s = \frac{b-a}{\max(a,b)}}$$

❒ **Calinski-Harabaz index** – By noting k the number of clusters, B_k and W_k the between and within-clustering dispersion matrices respectively defined as

$$B_k = \sum_{j=1}^{k} n_{c^{(i)}}(\mu_{c^{(i)}} - \mu)(\mu_{c^{(i)}} - \mu)^T, \qquad W_k = \sum_{i=1}^{m}(x^{(i)} - \mu_{c^{(i)}})(x^{(i)} - \mu_{c^{(i)}})^T$$

the Calinski-Harabaz index $s(k)$ indicates how well a clustering model defines its clusters, such that the higher the score, the more dense and well separated the clusters are. It is defined as follows:

$$\boxed{s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N-k}{k-1}}$$

Principal component analysis

It is a dimension reduction technique that finds the variance maximizing directions onto which to project the data.

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n\times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$\boxed{Az = \lambda z}$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n\times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n\times n}$. By noting $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$, we have:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

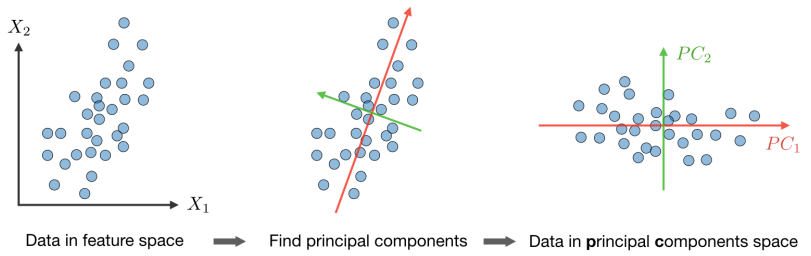
Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A.

❒ **Algorithm** – The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on k dimensions by maximizing the variance of the data as follows:

- Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$\boxed{x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}} \quad \text{where} \quad \boxed{\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}} \quad \text{and} \quad \boxed{\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2}$$

- Step 2: Compute $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T} \in \mathbb{R}^{n\times n}$, which is symmetric with real eigenvalues.
- Step 3: Compute $u_1, ..., u_k \in \mathbb{R}^n$ the k orthogonal principal eigenvectors of Σ, i.e. the orthogonal eigenvectors of the k largest eigenvalues.
- Step 4: Project the data on $\text{span}_{\mathbb{R}}(u_1,...,u_k)$. This procedure maximizes the variance among all k-dimensional spaces.



Independent component analysis

It is a technique meant to find the underlying generating sources.

❒ **Assumptions** – We assume that our data x has been generated by the n-dimensional source vector $s = (s_1,...,s_n)$, where s_i are independent random variables, via a mixing and non-singular matrix A as follows:

$$\boxed{x = As}$$

The goal is to find the unmixing matrix $W = A^{-1}$ by an update rule.

❒ **Bell and Sejnowski ICA algorithm** – This algorithm finds the unmixing matrix W by following the steps below:

- Write the probability of $x = As = W^{-1}s$ as:

$$p(x) = \prod_{i=1}^{n} p_s(w_i^T x) \cdot |W|$$

- Write the log likelihood given our training data $\{x^{(i)}, i \in [\![1,m]\!]\}$ and by noting g the sigmoid function as:

$$l(W) = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} \log\left(g'(w_j^T x^{(i)})\right) + \log|W|\right)$$

Therefore, the stochastic gradient ascent learning rule is such that for each training example $x^{(i)}$, we update W as follows:

$$\boxed{W \longleftarrow W + \alpha \left(\begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)^T} + (W^T)^{-1} \right)}$$

Cheat Sheet – PCA Dimensionality Reduction

What is PCA?

- Based on the dataset find a new set of orthogonal feature vectors in such a way that the data spread is maximum in the direction of the feature vector (or dimension)
- Rates the feature vector in the decreasing order of data spread (or variance)
- The datapoints have maximum variance in the first feature vector, and minimum variance in the last feature vector
- The variance of the datapoints in the direction of feature vector can be termed as a measure of information in that direction.

Steps

1. Standardize the datapoints
2. Find the covariance matrix from the given datapoints
3. Carry out eigen-value decomposition of the covariance matrix
4. Sort the eigenvalues and eigenvectors

$$X_{new} = \frac{X - mean(X)}{std(X)}$$

$$C[i,j] = cov(x_i, x_j)$$

$$C = V\Sigma V^{-1}$$

$$\Sigma_{sort} = sort(\Sigma) \quad V_{sort} = sort(V, \Sigma_{sort})$$

Dimensionality Reduction with PCA

- Keep the first m out of n feature vectors rated by PCA. These m vectors will be the best m vectors preserving the maximum information that could have been preserved with m vectors on the given dataset

Steps:

1. Carry out steps 1-4 from above
2. Keep first m feature vectors from the sorted eigenvector matrix
3. Transform the data for the new basis (feature vectors)
4. The importance of the feature vector is proportional to the magnitude of the eigen value

$$V_{reduced} = V[:, 0:m]$$

$$X_{reduced} = X_{new} \times V_{reduced}$$

Figure 1

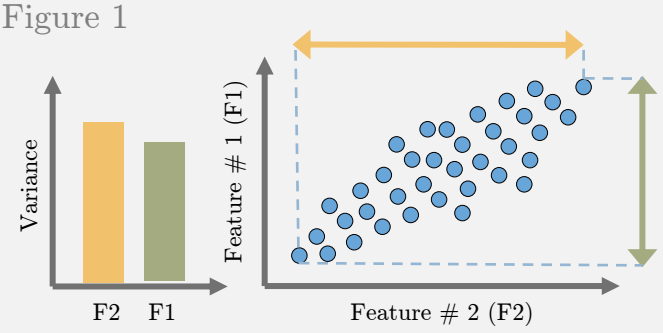


Figure 2

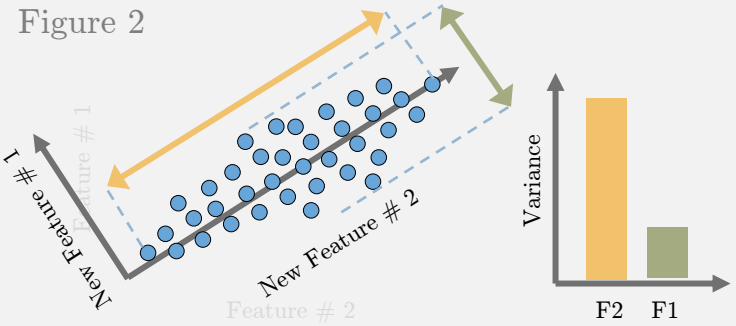


Figure 3

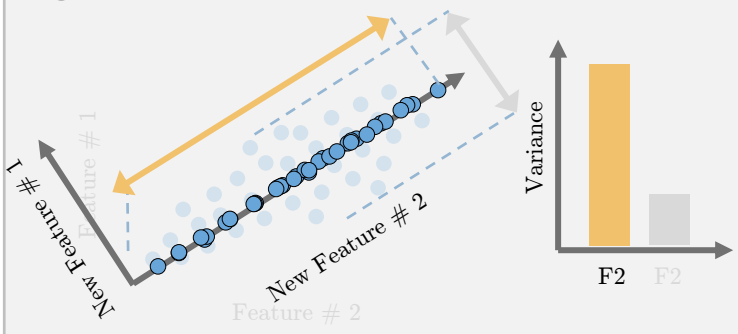


Figure 1: Datapoints with feature vectors as x and y-axis

Figure 2: The cartesian coordinate system is rotated to maximize the standard deviation along any one axis (new feature # 2)

Figure 3: Remove the feature vector with minimum standard deviation of datapoints (new feature # 1) and project the data on new feature # 2

4. Deep learning

VIP Cheatsheet: Deep Learning

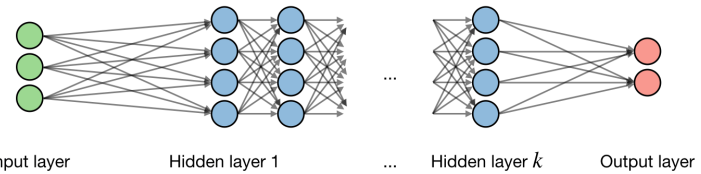
Afshine Amidi and Shervine Amidi

September 15, 2018

Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

❒ **Architecture** – The vocabulary around neural networks architectures is described in the figure below:

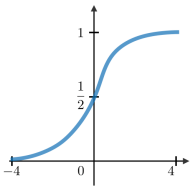
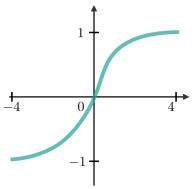
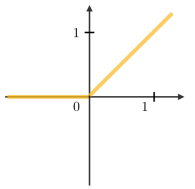
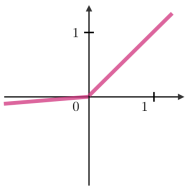


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

where we note w, b, z the weight, bias and output respectively.

❒ **Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

❒ **Cross-entropy loss** – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

❒ **Learning rate** – The learning rate, often noted η, indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

❒ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\boxed{\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}}$$

As a result, the weight is updated as follows:

$$\boxed{w \longleftarrow w - \eta \frac{\partial L(z,y)}{\partial w}}$$

❒ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

❒ **Dropout** – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability p or kept with probability $1-p$.

Convolutional Neural Networks

❒ **Convolutional layer requirement** – By noting W the input volume size, F the size of the convolutional layer neurons, P the amount of zero padding, then the number of neurons N that fit in a given volume is such that:

$$\boxed{N = \frac{W - F + 2P}{S} + 1}$$

❒ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Recurrent Neural Networks

❒ **Types of gates** – Here are the different types of gates that we encounter in a typical recurrent neural network:

Input gate	Forget gate	Output gate	Gate
Write to cell or not?	Erase a cell or not?	Reveal a cell or not?	How much writing?

❒ **LSTM** – A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding 'forget' gates.

Reinforcement Learning and Control

The goal of reinforcement learning is for an agent to learn how to evolve in an environment.

❒ **Markov decision processes** – A Markov decision process (MDP) is a 5-tuple $(S,A,\{P_{sa}\},\gamma,R)$ where:

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $\{P_{sa}\}$ are the state transition probabilities for $s \in \mathcal{S}$ and $a \in \mathcal{A}$
- $\gamma \in [0,1[$ is the discount factor
- $R : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ or $R : \mathcal{S} \longrightarrow \mathbb{R}$ is the reward function that the algorithm wants to maximize

❒ **Policy** – A policy π is a function $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ that maps states to actions.

Remark: we say that we execute a given policy π if given a state s we take the action $a = \pi(s)$.

❒ **Value function** – For a given policy π and a given state s, we define the value function V^π as follows:

$$\boxed{V^\pi(s) = E\Big[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...|s_0 = s,\pi\Big]}$$

❒ **Bellman equation** – The optimal Bellman equations characterizes the value function V^{π^*} of the optimal policy π^*:

$$\boxed{V^{\pi^*}(s) = R(s) + \max_{a\in\mathcal{A}} \gamma \sum_{s'\in S} P_{sa}(s')V^{\pi^*}(s')}$$

Remark: we note that the optimal policy π^* for a given state s is such that:

$$\boxed{\pi^*(s) = \underset{a\in\mathcal{A}}{\text{argmax}} \sum_{s'\in\mathcal{S}} P_{sa}(s')V^*(s')}$$

❒ **Value iteration algorithm** – The value iteration algorithm is in two steps:

- We initialize the value:

$$\boxed{V_0(s) = 0}$$

- We iterate the value based on the values before:

$$\boxed{V_{i+1}(s) = R(s) + \max_{a\in\mathcal{A}} \left[\sum_{s'\in\mathcal{S}} \gamma P_{sa}(s')V_i(s')\right]}$$

❒ **Maximum likelihood estimate** – The maximum likelihood estimates for the state transition probabilities are as follows:

$$\boxed{P_{sa}(s') = \frac{\#\text{times took action } a \text{ in state } s \text{ and got to } s'}{\#\text{times took action } a \text{ in state } s}}$$

❒ **Q-learning** – Q-learning is a model-free estimation of Q, which is done as follows:

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]}$$

Super VIP Cheatsheet: Deep Learning

Afshine Amidi and Shervine Amidi

November 25, 2018

Contents

1 Convolutional Neural Networks **2**
- 1.1 Overview . . . 2
- 1.2 Types of layer . . . 2
- 1.3 Filter hyperparameters . . . 2
- 1.4 Tuning hyperparameters . . . 3
- 1.5 Commonly used activation functions . . . 3
- 1.6 Object detection . . . 4
  - 1.6.1 Face verification and recognition . . . 5
  - 1.6.2 Neural style transfer . . . 5
  - 1.6.3 Architectures using computational tricks . . . 6

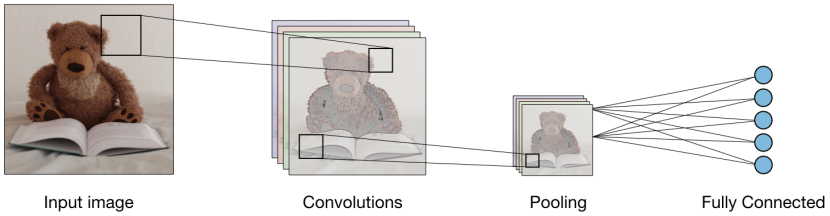
2 Recurrent Neural Networks **7**
- 2.1 Overview . . . 7
- 2.2 Handling long term dependencies . . . 8
- 2.3 Learning word representation . . . 9
  - 2.3.1 Motivation and notations . . . 9
  - 2.3.2 Word embeddings . . . 9
- 2.4 Comparing words . . . 9
- 2.5 Language model . . . 10
- 2.6 Machine translation . . . 10
- 2.7 Attention . . . 10

3 Deep Learning Tips and Tricks **11**
- 3.1 Data processing . . . 11
- 3.2 Training a neural network . . . 12
  - 3.2.1 Definitions . . . 12
  - 3.2.2 Finding optimal weights . . . 12
- 3.3 Parameter tuning . . . 12
  - 3.3.1 Weights initialization . . . 12
  - 3.3.2 Optimizing convergence . . . 12
- 3.4 Regularization . . . 13
- 3.5 Good practices . . . 13

1 Convolutional Neural Networks

1.1 Overview

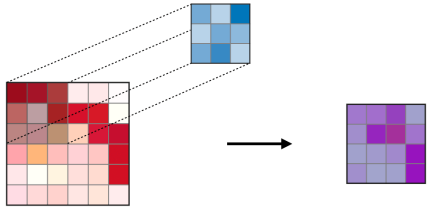
❒ **Architecture of a traditional CNN** – Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters that are described in the next sections.

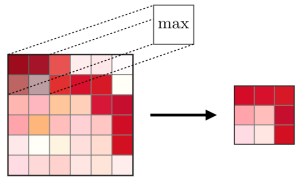
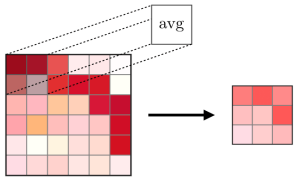
1.2 Types of layer

❒ **Convolutional layer (CONV)** – The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size F and stride S. The resulting output O is called *feature map* or *activation map*.

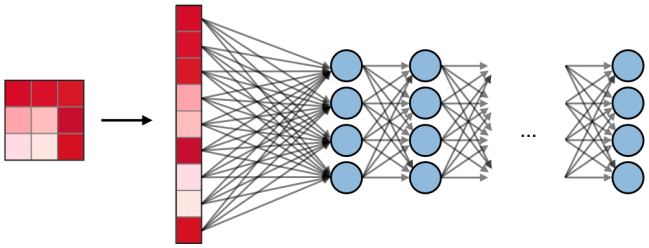


Remark: the convolution step can be generalized to the 1D and 3D cases as well.

❒ **Pooling (POOL)** – The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

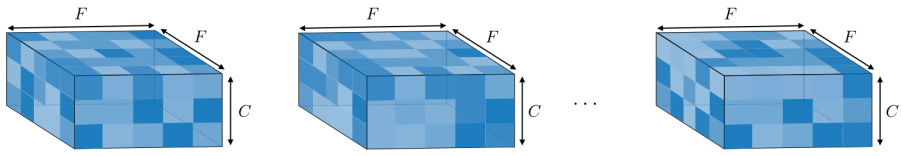
❒ **Fully Connected (FC)** – The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



1.3 Filter hyperparameters

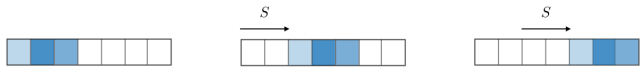
The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

❒ **Dimensions of a filter** – A filter of size $F \times F$ applied to an input containing C channels is a $F \times F \times C$ volume that performs convolutions on an input of size $I \times I \times C$ and produces an output feature map (also called activation map) of size $O \times O \times 1$.

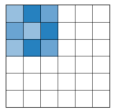
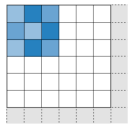
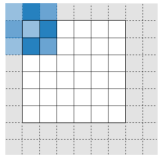


Remark: the application of K filters of size $F \times F$ results in an output feature map of size $O \times O \times K$.

❒ **Stride** – For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



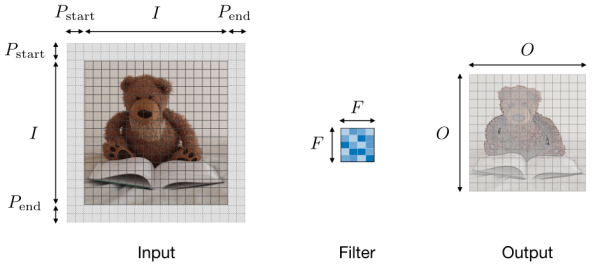
❒ **Zero-padding** – Zero-padding denotes the process of adding P zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

	Valid	Same	Full
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in [\![0, F-1]\!]$ $P_{\text{end}} = F - 1$
Illustration			
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\lceil \frac{I}{S} \rceil$ - Output size is mathematically convenient - Also called 'half' padding	- Maximum padding such that end convolutions are applied on the limits of the input - Filter 'sees' the input end-to-end

1.4 Tuning hyperparameters

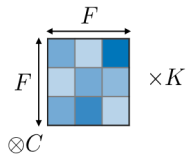
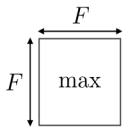
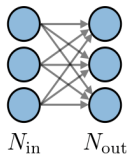
❒ **Parameter compatibility in convolution layer** – By noting I the length of the input volume size, F the length of the filter, P the amount of zero padding, S the stride, then the output size O of the feature map along that dimension is given by:

$$\boxed{O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1}$$



Remark: often times, $P_{start} = P_{end} \triangleq P$, in which case we can replace $P_{start} + P_{end}$ by $2P$ in the formula above.

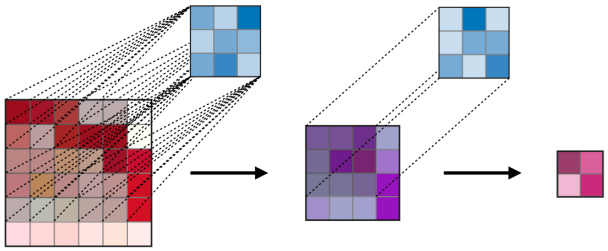
❒ **Understanding the complexity of the model** – In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	- One bias parameter per filter - In most cases, $S < F$ - A common choice for K is $2C$	- Pooling operation done channel-wise - In most cases, $S = F$	- Input is flattened - One bias parameter per neuron - The number of FC neurons is free of structural constraints

❒ **Receptive field** – The receptive field at layer k is the area denoted $R_k \times R_k$ of the input that each pixel of the k-th activation map can 'see'. By calling F_j the filter size of layer j and S_i the stride value of layer i and with the convention $S_0 = 1$, the receptive field at layer k can be computed with the formula:

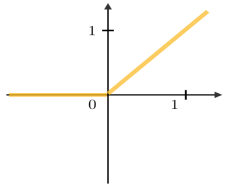
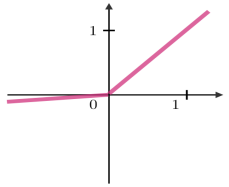
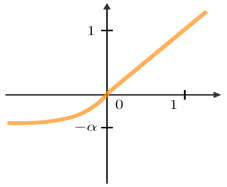
$$R_k = 1 + \sum_{j=1}^{k} (F_j - 1) \prod_{i=0}^{j-1} S_i$$

In the example below, we have $F_1 = F_2 = 3$ and $S_1 = S_2 = 1$, which gives $R_2 = 1+2 \cdot 1+2 \cdot 1 = 5$.



1.5 Commonly used activation functions

❒ **Rectified Linear Unit** – The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

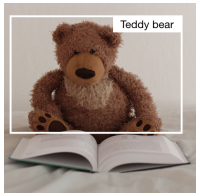
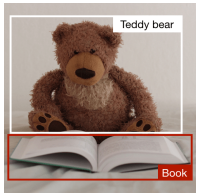
ReLU	Leaky ReLU	ELU
$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

❒ **Softmax** – The softmax step can be seen as a generalized logistic function that takes as input a vector of scores $x \in \mathbb{R}^n$ and outputs a vector of output probability $p \in \mathbb{R}^n$ through a softmax function at the end of the architecture. It is defined as follows:

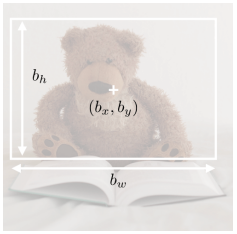
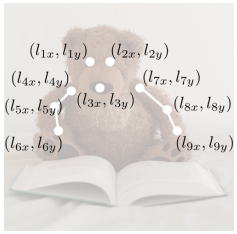
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

1.6 Object detection

❒ **Types of models** – There are 3 main types of object recognition algorithms, for which the nature of what is predicted is different. They are described in the table below:

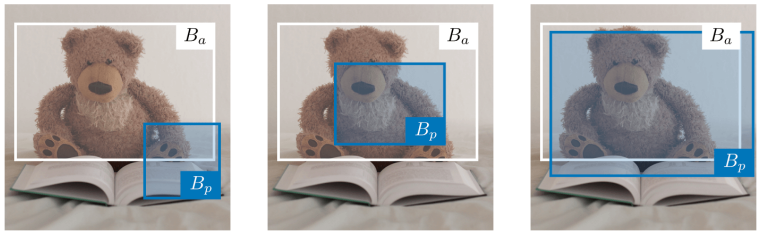
Image classification	Classification w. localization	Detection
		
- Classifies a picture - Predicts probability of object	- Detects object in a picture - Predicts probability of object and where it is located	- Detects up to several objects in a picture - Predicts probabilities of objects and where they are located
Traditional CNN	Simplified YOLO, R-CNN	YOLO, R-CNN

❒ **Detection** – In the context of object detection, different methods are used depending on whether we just want to locate the object or detect a more complex shape in the image. The two main ones are summed up in the table below:

Bounding box detection	Landmark detection
Detects the part of the image where the object is located	- Detects a shape or characteristics of an object (e.g. eyes) - More granular
	
Box of center (b_x,b_y), height b_h and width b_w	Reference points (l_{1x},l_{1y}), ...,(l_{nx},l_{ny})

❒ Intersection over Union – Intersection over Union, also known as IoU, is a function that quantifies how correctly positioned a predicted bounding box B_p is over the actual bounding box B_a. It is defined as:

$$\boxed{\text{IoU}(B_p,B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}}$$



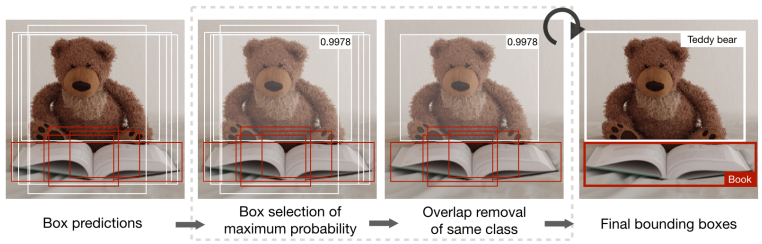
$\text{IoU}(B_p, B_a) = 0.1$ $\text{IoU}(B_p, B_a) = 0.5$ $\text{IoU}(B_p, B_a) = 0.9$

Remark: we always have $IoU \in [0,1]$. By convention, a predicted bounding box B_p is considered as being reasonably good if $IoU(B_p,B_a) \geqslant 0.5$.

❒ Anchor boxes – Anchor boxing is a technique used to predict overlapping bounding boxes. In practice, the network is allowed to predict more than one box simultaneously, where each box prediction is constrained to have a given set of geometrical properties. For instance, the first prediction can potentially be a rectangular box of a given form, while the second will be another rectangular box of a different geometrical form.

❒ Non-max suppression – The non-max suppression technique aims at removing duplicate overlapping bounding boxes of a same object by selecting the most representative ones. After having removed all boxes having a probability prediction lower than 0.6, the following steps are repeated while there are boxes remaining:

- Step 1: Pick the box with the largest prediction probability.
- Step 2: Discard any box having an IoU $\geqslant 0.5$ with the previous box.



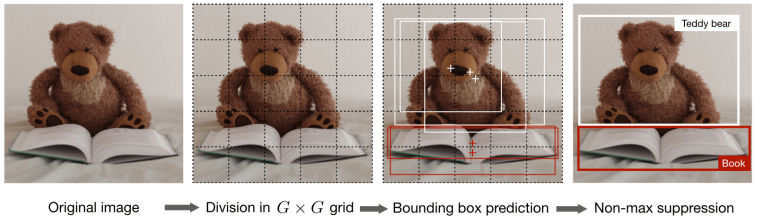
❒ YOLO – You Only Look Once (YOLO) is an object detection algorithm that performs the following steps:

- Step 1: Divide the input image into a $G \times G$ grid.
- Step 2: For each grid cell, run a CNN that predicts y of the following form:

$$\boxed{y = \Big[\underbrace{p_c,b_x,b_y,b_h,b_w,c_1,c_2,...,c_p}_{\text{repeated } k \text{ times}},...\Big]^T \in \mathbb{R}^{G\times G\times k\times(5+p)}}$$

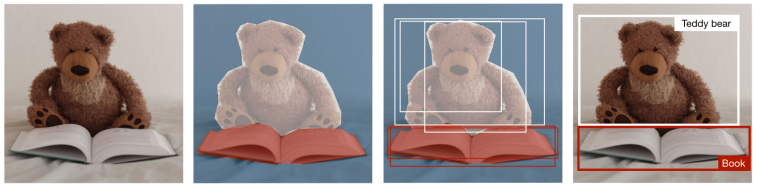
where p_c is the probability of detecting an object, b_x,b_y,b_h,b_w are the properties of the detected bouding box, c_1,...,c_p is a one-hot representation of which of the p classes were detected, and k is the number of anchor boxes.

- Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



Remark: when $p_c = 0$, then the network does not detect any object. In that case, the corresponding predictions $b_x, ..., c_p$ have to be ignored.

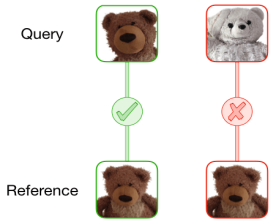
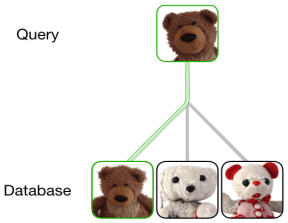
❒ R-CNN – Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.



Remark: although the original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.

1.6.1 Face verification and recognition

❒ **Types of models** – Two main types of model are summed up in table below:

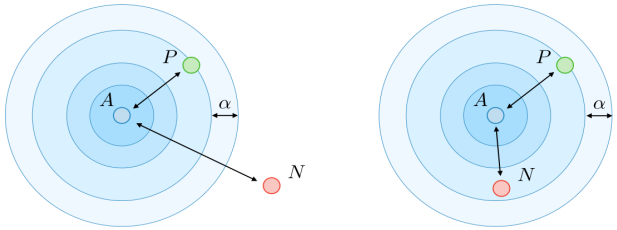
Face verification	Face recognition
- Is this the correct person? - One-to-one lookup	- Is this one of the K persons in the database? - One-to-many lookup
	

❒ **One Shot Learning** – One Shot Learning is a face verification algorithm that uses a limited training set to learn a similarity function that quantifies how different two given images are. The similarity function applied to two images is often noted $d(\text{image 1}, \text{image 2})$.

❒ **Siamese Network** – Siamese Networks aim at learning how to encode images to then quantify how different two images are. For a given input image $x^{(i)}$, the encoded output is often noted as $f(x^{(i)})$.

❒ **Triplet loss** – The triplet loss ℓ is a loss function computed on the embedding representation of a triplet of images A (anchor), P (positive) and N (negative). The anchor and the positive example belong to a same class, while the negative example to another one. By calling $\alpha \in \mathbb{R}^+$ the margin parameter, this loss is defined as follows:

$$\boxed{\ell(A,P,N) = \max\left(d(A,P) - d(A,N) + \alpha, 0\right)}$$



$\ell(A,P,N) = 0$ $\quad$ $\ell(A,P,N) > 0$

1.6.2 Neural style transfer

❒ **Motivation** – The goal of neural style transfer is to generate an image G based on a given content C and a given style S.

Content C $\quad$ Style S $\quad$ Generated image G

❒ **Activation** – In a given layer l, the activation is noted $a^{[l]}$ and is of dimensions $n_H \times n_w \times n_c$

❒ **Content cost function** – The content cost function $J_{\text{content}}(C,G)$ is used to determine how the generated image G differs from the original content image C. It is defined as follows:

$$\boxed{J_{\text{content}}(C,G) = \frac{1}{2}||a^{[l](C)} - a^{[l](G)}||^2}$$

❒ **Style matrix** – The style matrix $G^{[l]}$ of a given layer l is a Gram matrix where each of its elements $G^{[l]}_{kk'}$ quantifies how correlated the channels k and k' are. It is defined with respect to activations $a^{[l]}$ as follows:

$$\boxed{G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l]}_{ijk} a^{[l]}_{ijk'}}$$

Remark: the style matrix for the style image and the generated image are noted $G^{[l](S)}$ and $G^{[l](G)}$ respectively.

❒ **Style cost function** – The style cost function $J_{\text{style}}(S,G)$ is used to determine how the generated image G differs from the style S. It is defined as follows:

$$\boxed{J^{[l]}_{\text{style}}(S,G) = \frac{1}{(2n_H n_w n_c)^2}||G^{[l](S)} - G^{[l](G)}||^2_F = \frac{1}{(2n_H n_w n_c)^2} \sum_{k,k'=1}^{n_c} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'}\right)^2}$$

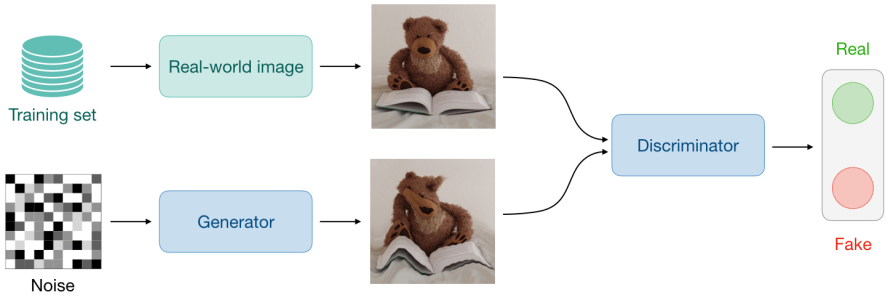
❒ **Overall cost function** – The overall cost function is defined as being a combination of the content and style cost functions, weighted by parameters α,β, as follows:

$$\boxed{J(G) = \alpha J_{\text{content}}(C,G) + \beta J_{\text{style}}(S,G)}$$

Remark: a higher value of α will make the model care more about the content while a higher value of β will make it care more about the style.

1.6.3 Architectures using computational tricks

❒ **Generative Adversarial Network** – Generative adversarial networks, also known as GANs, are composed of a generative and a discriminative model, where the generative model aims at generating the most truthful output that will be fed into the discriminative which aims at differentiating the generated and true image.



Remark: use cases using variants of GANs include text to image, music generation and synthesis.

❒ ResNet – The Residual Network architecture (also called ResNet) uses residual blocks with a high number of layers meant to decrease the training error. The residual block has the following characterizing equation:

$$\boxed{a^{[l+2]} = g(a^{[l]} + z^{[l+2]})}$$

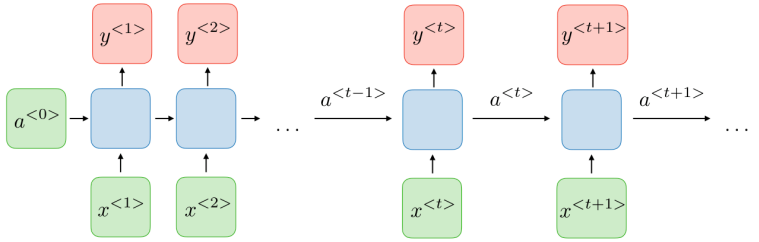
❒ Inception Network – This architecture uses inception modules and aims at giving a try at different convolutions in order to increase its performance. In particular, it uses the 1×1 convolution trick to lower the burden of computation.

⋆ ⋆ ⋆

2 Recurrent Neural Networks

2.1 Overview

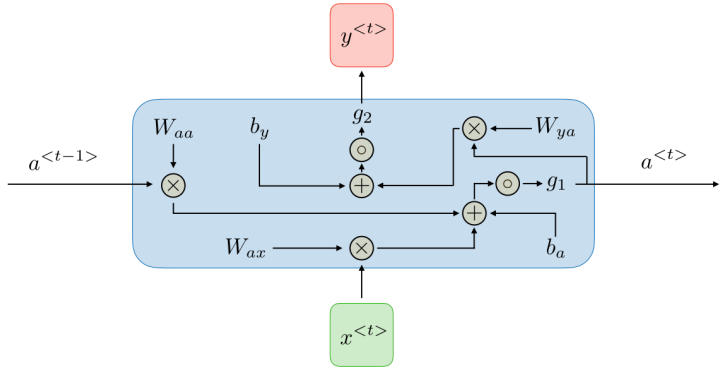
❒ Architecture of a traditional RNN – Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

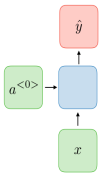
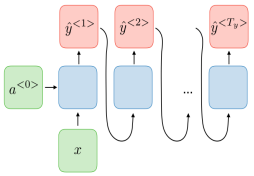
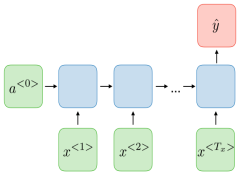
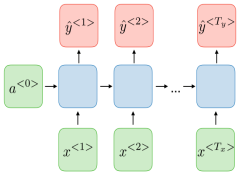
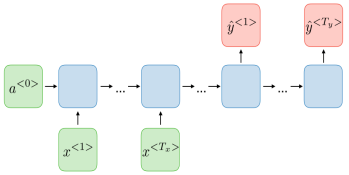
where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions



The pros and cons of a typical RNN architecture are summed up in the table below:

Advantages	Drawbacks
- Possibility of processing input of any length - Model size not increasing with size of input - Computation takes into account historical information - Weights are shared across time	- Computation being slow - Difficulty of accessing information from a long time ago - Cannot consider any future input for the current state

❒ Applications of RNNs – RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

❒ **Loss function** – In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time steps is defined based on the loss at every time step as follows:

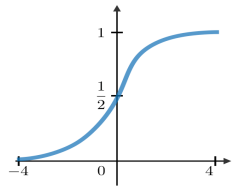
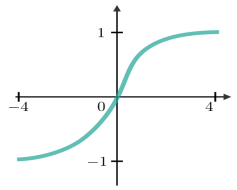
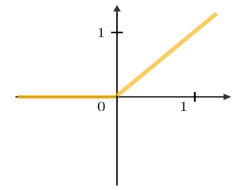
$$\boxed{\mathcal{L}(\widehat{y},y) = \sum_{t=1}^{T_y} \mathcal{L}(\widehat{y}^{<t>},y^{<t>})}$$

❒ **Backpropagation through time** – Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix W is expressed as follows:

$$\boxed{\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \left.\frac{\partial \mathcal{L}^{(T)}}{\partial W}\right|_{(t)}}$$

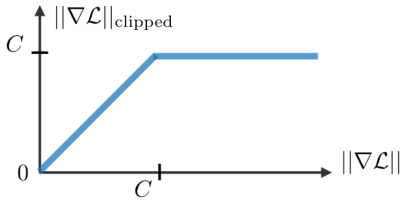
2.2 Handling long term dependencies

❒ **Commonly used activation functions** – The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$
		

❒ **Vanishing/exploding gradient** – The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

❒ **Gradient clipping** – It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



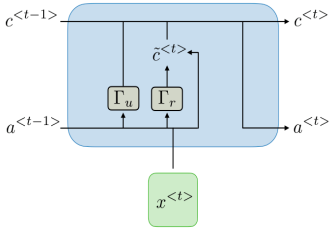
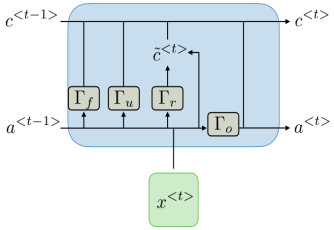
❒ **Types of gates** – In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted Γ and are equal to:

$$\boxed{\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)}$$

where W, U, b are coefficients specific to the gate and σ is the sigmoid function. The main ones are summed up in the table below:

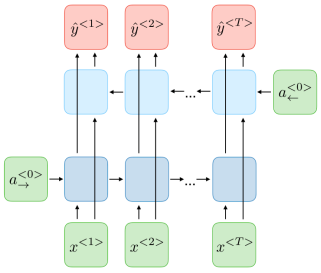
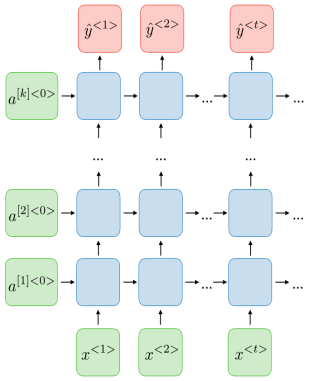
Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

❒ **GRU/LSTM** – Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

	Gated Recurrent Unit (GRU)	**Long Short-Term Memory** (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1-\Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Remark: the sign $\star$ denotes the element-wise multiplication between two vectors.

❒ **Variants of RNNs** – The table below sums up the other commonly used RNN architectures:

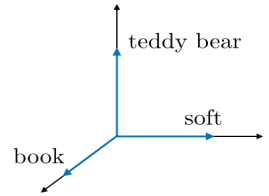
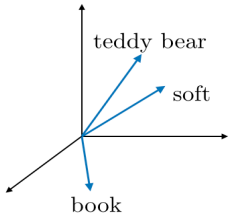
Bidirectional (BRNN)	**Deep** (DRNN)
	

2.3 Learning word representation

In this section, we note V the vocabulary and $|V|$ its size.

2.3.1 Motivation and notations

❒ **Representation techniques** – The two main ways of representing words are summed up in the table below:

1-hot representation	**Word embedding**
	
- Noted o_w - Naive approach, no similarity information	- Noted e_w - Takes into account words similarity

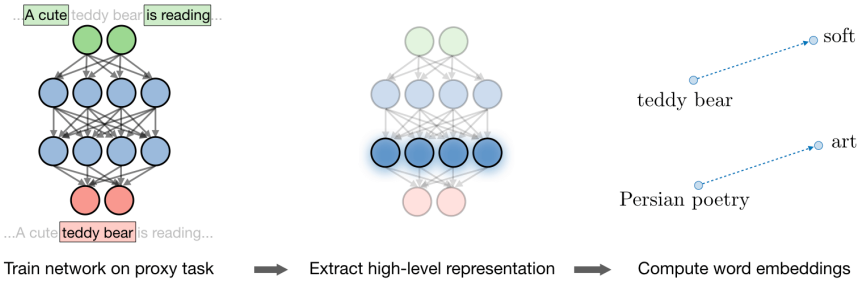
❒ **Embedding matrix** – For a given word w, the embedding matrix E is a matrix that maps its 1-hot representation o_w to its embedding e_w as follows:

$$\boxed{e_w = Eo_w}$$

Remark: learning the embedding matrix can be done using target/context likelihood models.

2.3.2 Word embeddings

❒ **Word2vec** – Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words. Popular models include skip-gram, negative sampling and CBOW.



❒ **Skip-gram** – The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word t happening with a context word c. By noting θ_t a parameter associated with t, the probability $P(t|c)$ is given by:

$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.

❒ **Negative sampling** – It is a set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously, with the models being trained on sets of k negative examples and 1 positive example. Given a context word c and a target word t, the prediction is expressed by:

$$P(y = 1|c,t) = \sigma(\theta_t^T e_c)$$

Remark: this method is less computationally expensive than the skip-gram model.

❒ **GloVe** – The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurence matrix X where each $X_{i,j}$ denotes the number of times that a target i occurred with a context j. Its cost function J is as follows:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log(X_{ij}))^2$$

here f is a weighting function such that $X_{i,j} = 0 \Longrightarrow f(X_{i,j}) = 0$.

Given the symmetry that e and θ play in this model, the final word embedding $e_w^{(\text{final})}$ is given by:

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

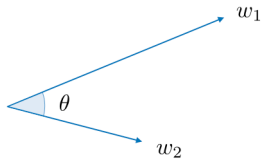
Remark: the individual components of the learned word embeddings are not necessarily interpretable.

2.4 Comparing words

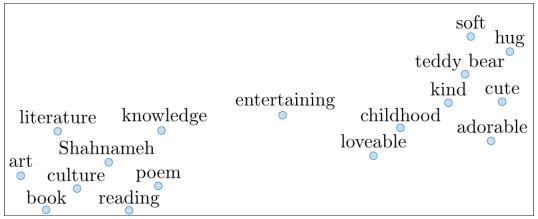
❒ **Cosine similarity** – The cosine similarity between words w_1 and w_2 is expressed as follows:

$$\text{similarity} = \frac{w_1 \cdot w_2}{||w_1||\ ||w_2||} = \cos(\theta)$$

Remark: θ is the angle between words w_1 and w_2.



❒ ***t*-SNE** – t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique aimed at reducing high-dimensional embeddings into a lower dimensional space. In practice, it is commonly used to visualize word vectors in the 2D space.



2.5 Language model

❒ **Overview** – A language model aims at estimating the probability of a sentence $P(y)$.

❒ **n-gram model** – This model is a naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

❒ **Perplexity** – Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words T. The perplexity is such that the lower, the better and is defined as follows:

$$\text{PP} = \prod_{t=1}^{T} \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

Remark: PP is commonly used in t-SNE.

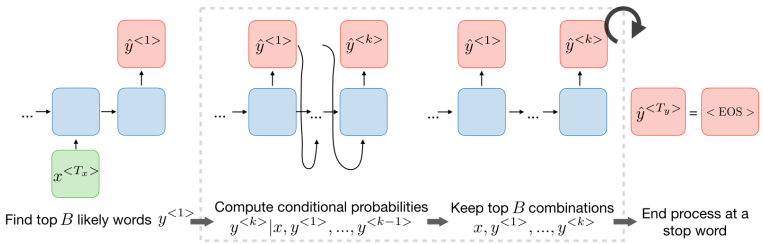
2.6 Machine translation

❒ **Overview** – A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model. The goal is to find a sentence y such that:

$$y = \underset{y^{<1>},\ldots,y^{<T_y>}}{\arg\max}\ P(y^{<1>},\ldots,y^{<T_y>}|x)$$

❒ **Beam search** – It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence y given an input x.

- Step 1: Find top B likely words $y^{<1>}$
- Step 2: Compute conditional probabilities $y^{<k>}|x,y^{<1>},\ldots,y^{<k-1>}$
- Step 3: Keep top B combinations $x,y^{<1>},\ldots,y^{<k>}$



Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.

❒ **Beam width** – The beam width B is a parameter for beam search. Large values of B yield to better result but with slower performance and increased memory. Small values of B lead to worse results but is less computationally intensive. A standard value for B is around 10.

❒ **Length normalization** – In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective } = \frac{1}{T_y^\alpha}\sum_{t=1}^{T_y}\log\left[p(y^{<t>}|x,y^{<1>},...,y^{<t-1>})\right]$$

Remark: the parameter α can be seen as a softener, and its value is usually between 0.5 and 1.

❒ **Error analysis** – When obtaining a predicted translation $\widehat{y}$ that is bad, one can wonder why we did not get a good translation y^* by performing the following error analysis:

Case	$P(y^*\|x) > P(\widehat{y}\|x)$	$P(y^*\|x) \leqslant P(\widehat{y}\|x)$
Root cause	Beam search faulty	RNN faulty
Remedies	Increase beam width	- Try different architecture - Regularize - Get more data

❒ **Bleu score** – The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on n-gram precision. It is defined as follows:

$$\text{bleu score} = \exp\left(\frac{1}{n}\sum_{k=1}^{n}p_k\right)$$

where p_n is the bleu score on n-gram only defined as follows:

$$p_n = \frac{\sum_{\text{n-gram}\in\widehat{y}}\text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram}\in\widehat{y}}\text{count}(\text{n-gram})}$$

Remark: a brevity penalty may be applied to short predicted translations to prevent an artificially inflated bleu score.

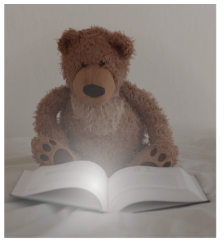
2.7 Attention

❒ **Attention model** – This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting $\alpha^{<t,t'>}$ the amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ and $c^{<t>}$ the context at time t, we have:

$$c^{<t>} = \sum_{t'}\alpha^{<t,t'>}a^{<t'>} \quad \text{with} \quad \sum_{t'}\alpha^{<t,t'>} = 1$$

Remark: the attention scores are commonly used in image captioning and machine translation.

A cute teddy bear is reading Persian literature



A cute teddy bear is reading Persian literature

❒ **Attention weight** – The amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ is given by $\alpha^{<t,t'>}$ computed as follows:

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x}\exp(e^{<t,t''>})}$$

Remark: computation complexity is quadratic with respect to T_x.

⋆ ⋆ ⋆

3 Deep Learning Tips and Tricks

3.1 Data processing

❒ **Data augmentation** – Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

❒ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

3.2 Training a neural network

3.2.1 Definitions

❒ **Epoch** – In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

❒ **Mini-batch gradient descent** – During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

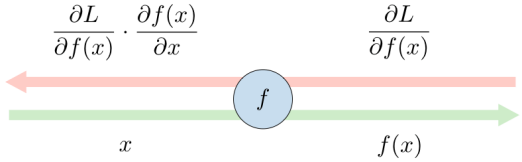
❒ **Loss function** – In order to quantify how a given model performs, the loss function L is usually used to evaluate to what extent the actual outputs y are correctly predicted by the model outputs z.

❒ **Cross-entropy loss** – In the context of binary classification in neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

3.2.2 Finding optimal weights

❒ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.

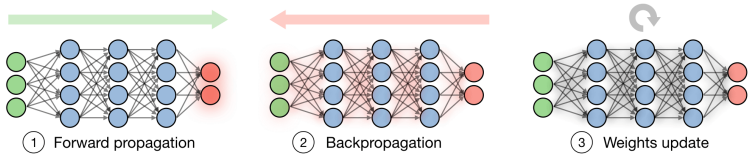


Using this method, each weight is updated with the rule:

$$\boxed{w \longleftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}}$$

❒ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.

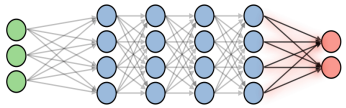
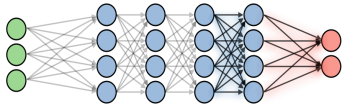
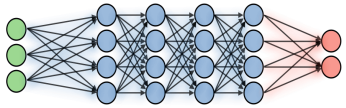


3.3 Parameter tuning

3.3.1 Weights initialization

❒ **Xavier initialization** – Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

❒ **Transfer learning** – Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

3.3.2 Optimizing convergence

❒ **Learning rate** – The learning rate, often noted α or sometimes η, indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

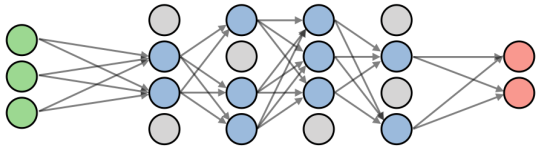
❒ **Adaptive learning rates** – Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful. They are summed up in the table below:

Method	Explanation	Update of w	Update of b
Momentum	- Dampens oscillations - Improvement to SGD - 2 parameters to tune	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Speeds up learning algorithm by controlling oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Most popular method - 4 parameters to tune	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \longleftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD.

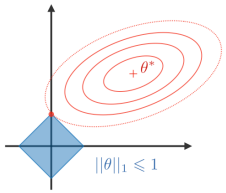
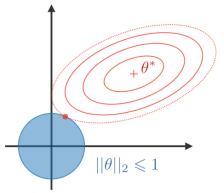
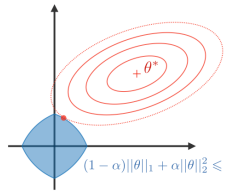
3.4 Regularization

❒ **Dropout** – Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.

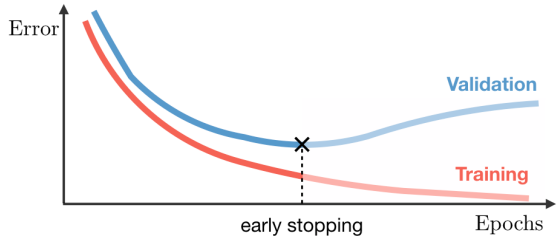


Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter $1-p$.

❒ **Weight regularization** – In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$... + \lambda \|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda \|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda \left[(1-\alpha) \|\|\theta\|\|_1 + \alpha \|\|\theta\|\|_2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

❒ **Early stopping** – This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



3.5 Good practices

❒ **Overfitting small batch** – When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

❒ **Gradient checking** – Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

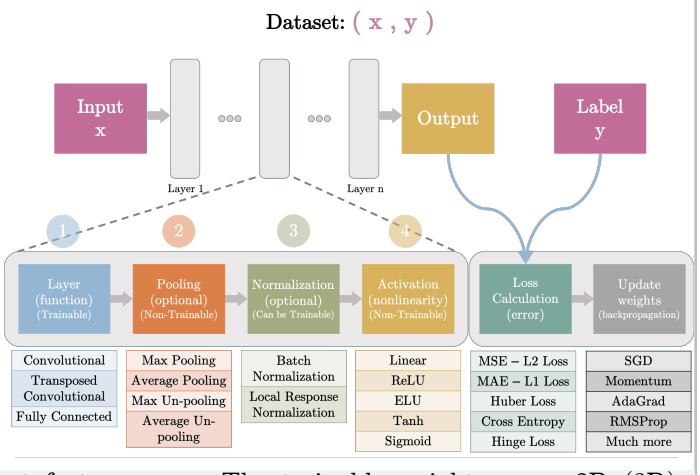
	Numerical gradient	**Analytical gradient**
Formula	$\frac{df}{dx}(x) \approx \frac{f(x+h)-f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Comments	- Expensive; loss has to be computed two times per dimension - Used to verify correctness of analytical implementation -Trade-off in choosing h not too small (numerical instability) nor too large (poor gradient approx.)	- 'Exact' result - Direct computation - Used in the final implementation

⋆ ⋆ ⋆

Cheat Sheet – Convolutional Neural Network

Convolutional Neural Network:

The data gets into the CNN through the input layer and passes through various hidden layers before getting to the output layer. The output of the network is compared to the actual labels in terms of loss or error. The partial derivatives of this loss w.r.t the trainable weights are calculated, and the weights are updated through one of the various methods using backpropagation.



CNN Template:

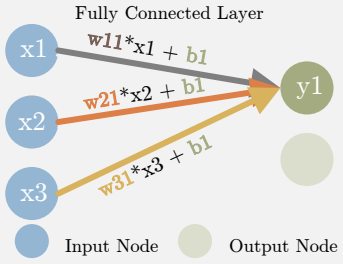
Most of the commonly used hidden layers (not all) follow a pattern

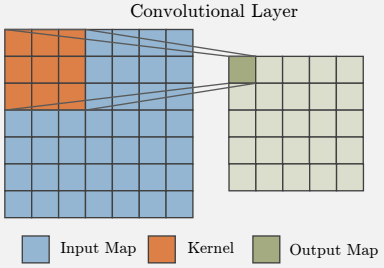
1. **Layer function:** Basic transforming function such as convolutional or fully connected layer.

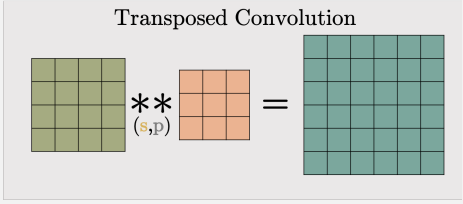
a. **Fully Connected:** Linear functions between the input and the output.

a. **Convolutional Layers:** These layers are applied to 2D (3D) input feature maps. The trainable weights are a 2D (3D) kernel/filter that moves across the input feature map, generating dot products with the overlapping region of the input feature map.

b. **Transposed Convolutional (DeConvolutional) Layer:** Usually used to increase the size of the output feature map (Upsampling) The idea behind the transposed convolutional layer is to undo (not exactly) the convolutional layer







2. **Pooling:** Non-trainable layer to change the size of the feature map

a. **Max/Average Pooling:** Decrease the spatial size of the input layer based on selecting the maximum/average value in receptive field defined by the kernel

b. **UnPooling:** A non-trainable layer used to increase the spatial size of the input layer based on placing the input pixel at a certain index in the receptive field of the output defined by the kernel.

3. **Normalization:** Usually used just before the activation functions to limit the unbounded activation from increasing the output layer values too high

a. **Local Response Normalization LRN:** A **non-trainable layer** that square-normalizes the pixel values in a feature map within a local neighborhood.

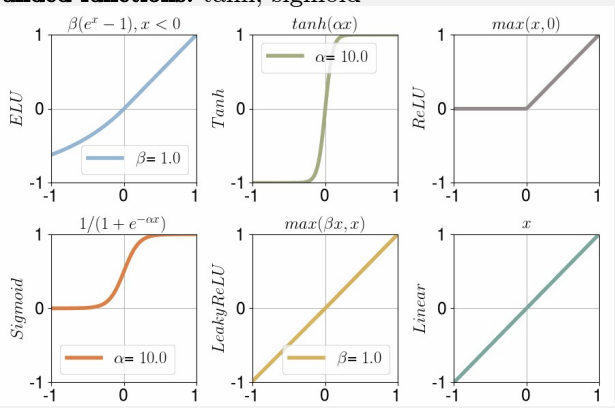
b. **Batch Normalization:** A trainable approach to normalizing the data by learning scale and shift variable during training.

3. **Activation:** Introduce non-linearity so CNN can efficiently map non-linear complex mapping.

a. **Non-parametric/Static functions:** Linear, ReLU

b. **Parametric functions:** ELU, tanh, sigmoid, Leaky ReLU

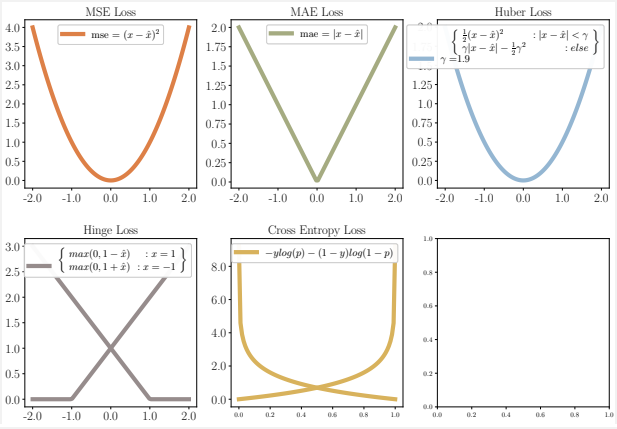
c. **Bounded functions:** tanh, sigmoid



5. **Loss function:** Quantifies how far off the CNN prediction is from the actual labels.

a. Regression Loss Functions: MAE, MSE, Huber loss

b. Classification Loss Functions: Cross entropy, Hinge loss



Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

Cheat Sheet – Famous CNNs

AlexNet – 2012

Why: AlexNet was born out of the need to improve the results of the ImageNet challenge.

What: The network consists of 5 Convolutional (CONV) layers and 3 Fully Connected (FC) layers. The activation used is the Rectified Linear Unit (ReLU).

How: Data augmentation is carried out to reduce over-fitting, Uses Local response localization.

AlexNet Network - Structural Details

Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							62,378,344

VGGNet – 2014

Why: VGGNet was born out of the need to reduce the # of parameters in the CONV layers and improve on training time

What: There are multiple variants of VGGNet (VGG16, VGG19, etc.)

How: The important point to note here is that all the conv kernels are of size 3x3 and maxpool kernels are of size 2x2 with a stride of two.

VGG16 - Structural Details

#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
						Total							138,423,208

ResNet – 2015

Why: Neural Networks are notorious for not being able to find a simpler mapping when it exists. ResNet solves that.

What: There are multiple versions of ResNetXX architectures where 'XX' denotes the number of layers. The most used ones are ResNet50 and ResNet101. Since the vanishing gradient problem was taken care of (more about it in the How part), CNN started to get deeper and deeper

How: ResNet architecture makes use of shortcut connections do solve the vanishing gradient problem. The basic building block of ResNet is a Residual block that is repeated throughout the network.

ResNet18 - Structural Details

#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
						Total								11,511,784

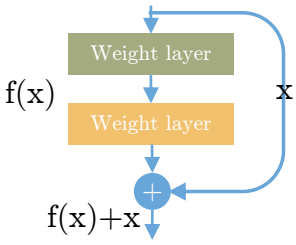


Figure 1 ResNet Block

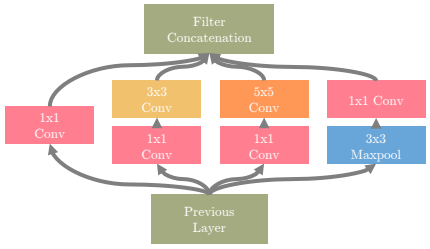


Figure 2 Inception Block

Inception – 2014

Why: Lager kernels are preferred for more global features, on the other hand, smaller kernels provide good results in detecting area-specific features. For effective recognition of such a variable-sized feature, we need kernels of different sizes. That is what Inception does.

What: The Inception network architecture consists of several inception modules of the following structure. Each inception module consists of four operations in parallel, 1x1 conv layer, 3x3 conv layer, 5x5 conv layer, max pooling

How: Inception increases the network space from which the best network is to be chosen via training. Each inception module can capture salient features at different levels.

GoogLeNet - Structural Details

	Input Image			output			Layer	Input Layer	Stride	Pad	Kernel	in	out	Param
	227	227	3	112	112	64	conv1	input	2	1	7	3	64	9472
	112	112	64	56	56	64	maxpool1	conv1	2	0.5	3	64	64	0
	56	56	64	56	56	64	conv1x1	maxpool1	1	0	1	64	64	4160
	56	56	64	56	56	192	conv2-1		1	1	3	64	192	110784
	56	56	192	28	28	192	maxpool2		2	0.5	3	192	192	0
inception (3a)	28	28	192	28	28	96	conv1x1a	maxpool2	1	0	1	192	96	18528
	28	28	96	28	28	16	conv1x1b	maxpool2	1	0	1	192	16	3088
	28	28	192	28	28	192	maxpool-a	maxpool2	1	0	1	192	192	0
	28	28	192	28	28	64	conv1x1c	maxpool2	1	0	1	192	64	12352
	28	28	96	28	28	128	conv3-3	conv1x1a	1	1	3	96	128	110720
	28	28	16	28	28	32	conv5x5	conv1x1b	1	2	5	16	32	12832
	28	28	192	28	28	32	conv1x1d	maxpool-a	1	0	1	192	32	6176
				28	28	256	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
inception (3b)	28	28	256	28	28	128	conv1x1a	depth-concat	1	0	1	1	256	128	32896
	28	28	128	28	28	32	conv1x1b	depth-concat	1	0	1	1	256	32	8224
	28	28	192	28	28	256	maxpool-a	depth-concat	1	1	3	3	256	256	0
	28	28	192	28	28	128	conv1x1c	depth-concat	1	0	1	1	256	128	32896
	28	28	96	28	28	192	conv3-3	conv1x1a	1	1	3	3	128	192	221376
	28	28	16	28	28	96	conv5x5	conv1x1b	1	2	5	5	32	96	76896
	28	28	192	28	28	64	conv1x1d	maxpool-a	1	0	1	1	256	64	16448
				28	28	480	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
	28	28	480	14	14	480	maxpool3	depth-concat	2	0.5	3	3	480	480	0
inception (4a)	14	14	480	14	14	96	conv1x1a	maxpool3	1	0	1	1	480	96	46176
	14	14	480	14	14	16	conv1x1b	maxpool3	1	0	1	1	480	16	7696
	14	14	480	14	14	480	maxpool-a	maxpool3	1	1	3	3	480	480	0
	14	14	480	14	14	192	conv1x1c	maxpool3	1	0	1	1	480	192	92352
	14	14	96	14	14	208	conv3-3	conv1x1a	1	1	3	3	96	208	179920
	14	14	16	14	14	48	conv5x5	conv1x1b	1	2	5	5	16	48	19248
	14	14	192	14	14	64	conv1x1d	maxpool-a	1	0	1	1	480	64	30784
				14	14	512	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
inception (4b)	14	14	512	14	14	112	conv1x1a	depth-concat	1	0	1	1	512	112	57456
	14	14	512	14	14	24	conv1x1b	depth-concat	1	0	1	1	64	24	1560
	14	14	512	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
	14	14	512	14	14	160	conv1x1c	depth-concat	1	0	1	1	64	160	10400
	14	14	96	14	14	224	conv3-3	conv1x1a	1	1	3	3	112	224	226016
	14	14	16	14	14	64	conv5x5	conv1x1b	1	2	5	5	24	64	38464
	14	14	160	14	14	64	conv1x1d	maxpool-a	1	0	1	1	64	64	4160
				14	14	512	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
inception (4c)	14	14	512	14	14	128	conv1x1a	depth-concat	1	0	1	1	512	128	65664
	14	14	512	14	14	24	conv1x1b	depth-concat	1	0	1	1	64	24	1560
	14	14	512	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
	14	14	512	14	14	128	conv1x1c	depth-concat	1	0	1	1	64	128	8320
	14	14	96	14	14	256	conv3-3	conv1x1a	1	1	3	3	128	256	295168
	14	14	16	14	14	64	conv5x5	conv1x1b	1	2	5	5	24	64	38464
	14	14	128	14	14	64	conv1x1d	maxpool-a	1	0	1	1	64	64	4160
				14	14	512	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
inception (4d)	14	14	512	14	14	144	conv1x1a	depth-concat	1	0	1	1	512	144	73872
	14	14	512	14	14	32	conv1x1b	depth-concat	1	0	1	1	64	32	2080
	14	14	512	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
	14	14	512	14	14	112	conv1x1c	depth-concat	1	0	1	1	64	112	7280
	14	14	96	14	14	288	conv3-3	conv1x1a	1	1	3	3	144	288	373536
	14	14	16	14	14	64	conv5x5	conv1x1b	1	2	5	5	32	64	51264
	14	14	112	14	14	64	conv1x1d	maxpool-a	1	0	1	1	64	64	4160
				14	14	528	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
inception (4e)	14	14	528	14	14	160	conv1x1a	depth-concat	1	0	1	1	528	160	84640
	14	14	528	14	14	32	conv1x1b	depth-concat	1	0	1	1	64	32	2080
	14	14	528	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
	14	14	528	14	14	256	conv1x1c	depth-concat	1	0	1	1	64	256	16640
	14	14	96	14	14	320	conv3-3	conv1x1a	1	1	3	3	160	320	461120
	14	14	16	14	14	128	conv5x5	conv1x1b	1	2	5	5	32	128	102528
	14	14	256	14	14	128	conv1x1d	maxpool-a	1	0	1	1	64	128	8320
				14	14	832	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
	14	14	832	7	7	832	maxpool4	depth-concat	2	0.5	3	3	832	832	0
inception (5a)	7	7	832	7	7	160	conv1x1a	maxpool4	1	0	1	1	832	160	133280
	7	7	832	7	7	32	conv1x1b	maxpool4	1	0	1	1	832	32	26656
	7	7	832	7	7	832	maxpool-a	maxpool4	1	1	3	3	832	832	0
	7	7	832	7	7	256	conv1x1c	maxpool4	1	0	1	1	832	256	213248
	7	7	96	7	7	320	conv3-3	conv1x1a	1	1	3	3	160	320	461120
	7	7	16	7	7	128	conv5x5	conv1x1b	1	2	5	5	32	128	102528
	7	7	256	7	7	128	conv1x1d	maxpool-a	1	0	1	1	832	128	106624
				7	7	832	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
inception (5b)	7	7	832	7	7	192	conv1x1a	depth-concat	1	0	1	1	832	192	159936
	7	7	832	7	7	48	conv1x1b	depth-concat	1	0	1	1	832	48	39984
	7	7	832	7	7	832	maxpool-a	depth-concat	1	1	3	3	832	832	0
	7	7	832	7	7	384	conv1x1c	depth-concat	1	0	1	1	832	384	319872
	7	7	96	7	7	384	conv3-3	conv1x1a	1	1	3	3	192	384	663936
	7	7	16	7	7	128	conv5x5	conv1x1b	1	2	5	5	48	128	153728
	7	7	384	7	7	128	conv1x1d	maxpool-a	1	0	1	1	128	128	16512
				7	7	1024	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d						
	7	7	1024	1	1	1024	avgpool	depth-concat	1	0	7	7	1024	1024	0
	1	1	1024	1	1	1000	fc	depth-concat	1	0	1	1	1024	1000	1025000
						Total									6,414,360

Comparison

Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here

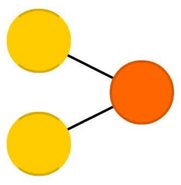
A mostly complete chart of

Neural Networks

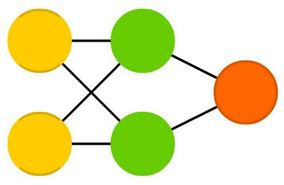
 asimovinstitute.org

- Input Cell
- Backfed Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Gated Memory Cell
- Kernel
- Convolution or Pool

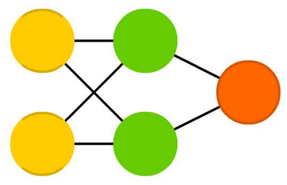
Perceptron (P)



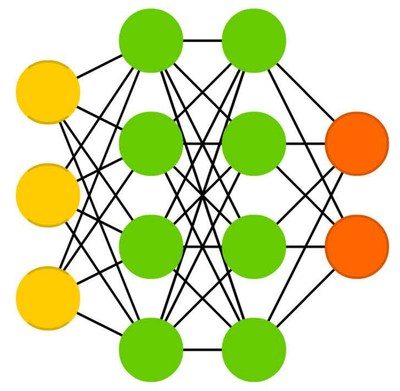
Feed Forward (FF)



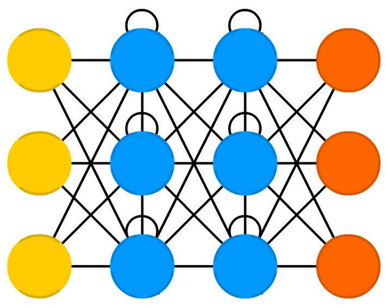
Radial Basis Network (RBF)



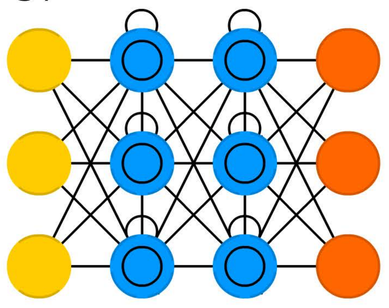
Deep Feed Forward (DFF)



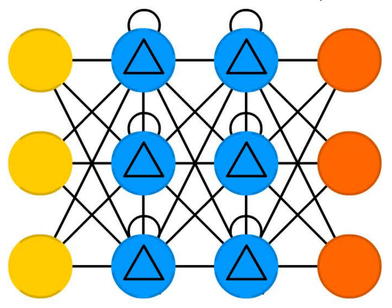
Recurrent Neural Network (RNN)



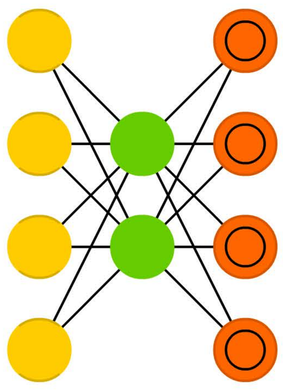
Long / Short Term Memory (LSTM)



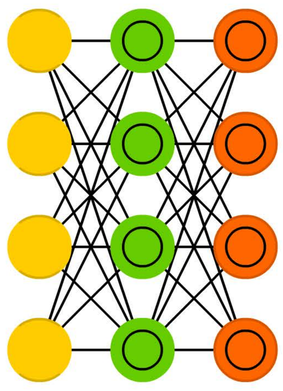
Gated Recurrent Unit (GRU)



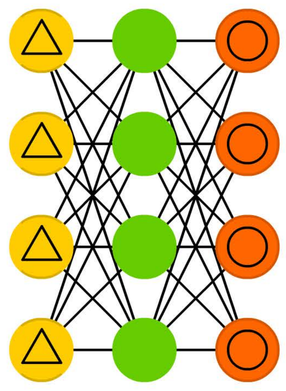
Auto Encoder (AE)



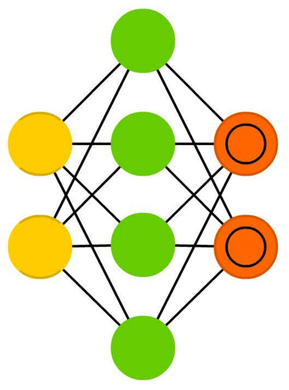
Variational AE (VAE)



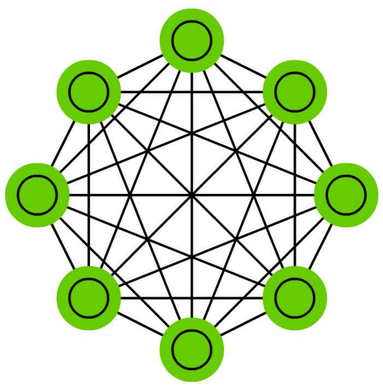
Denoising AE (DAE)



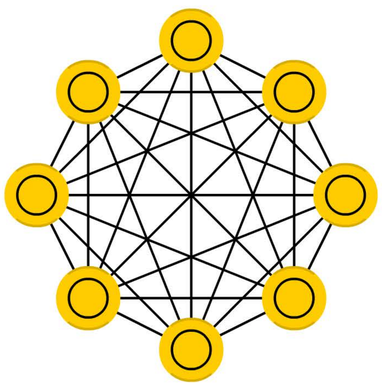
Sparse AE (SAE)



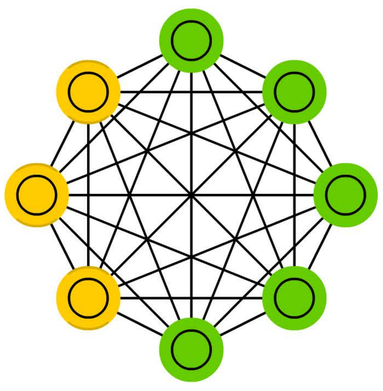
Markov Chain (MC)



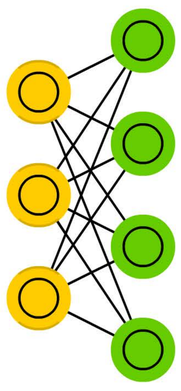
Hopfield Network (HN)



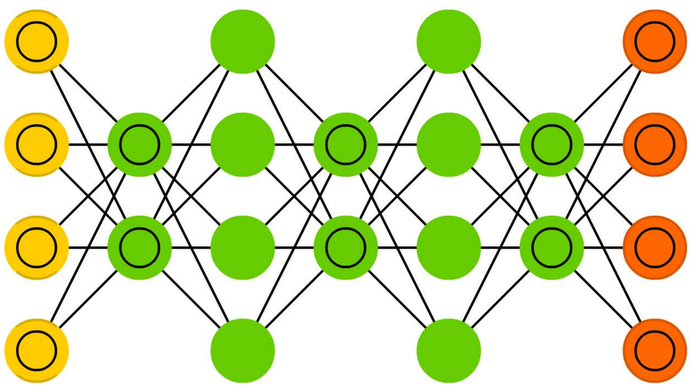
Boltzmann Machine (BM)



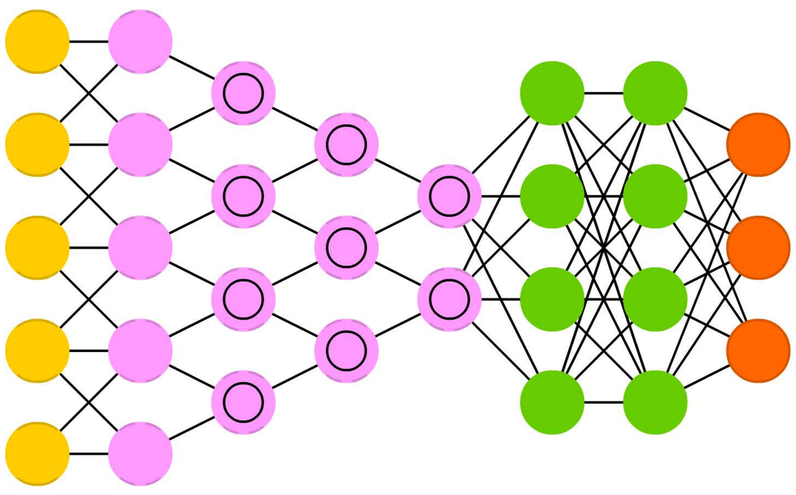
Restricted BM (RBM)



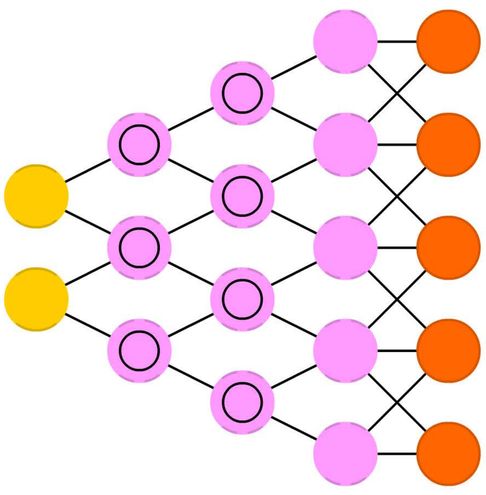
Deep Belief Network (DBN)



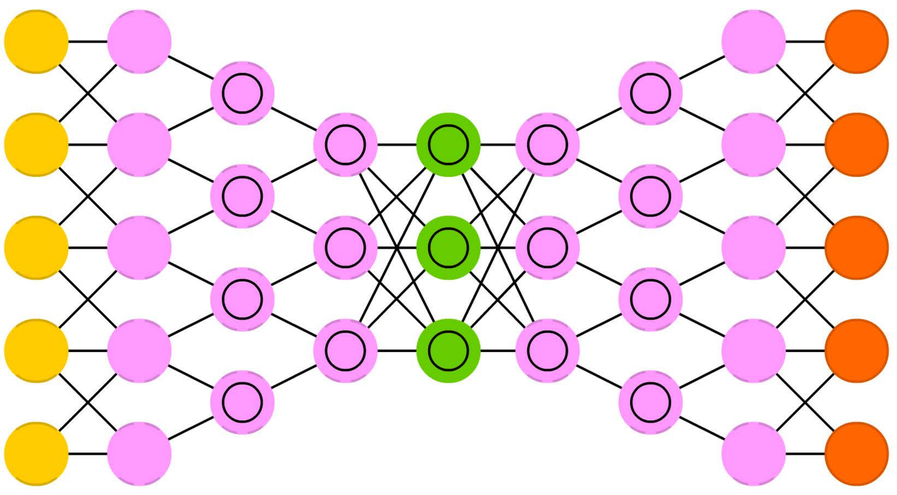
Deep Convolutional Network (DCN)



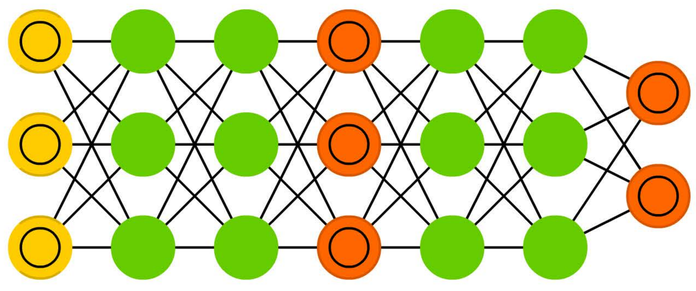
Deconvolutional Network (DN)



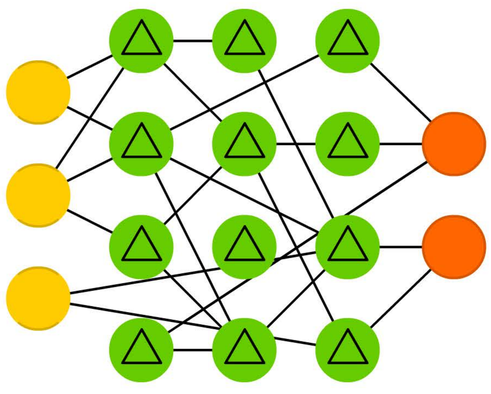
Deep Convolutional Inverse Graphics Network (DCIGN)



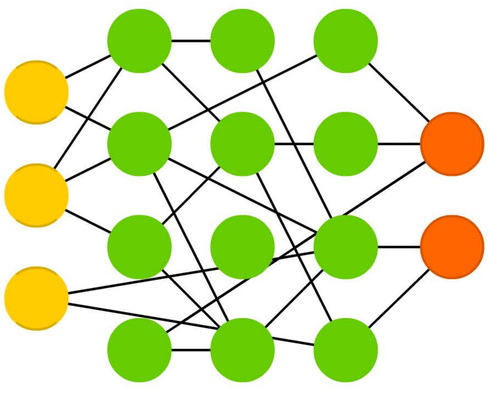
Generative Adversarial Network (GAN)



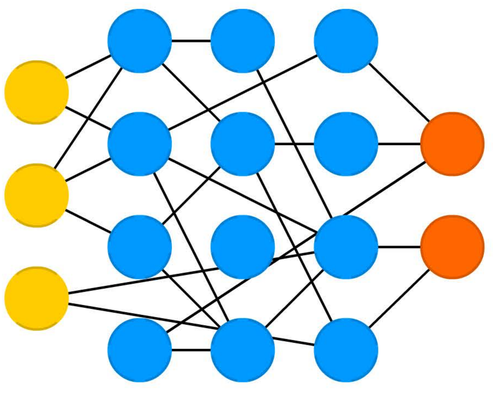
Liquid State Machine (LSM)



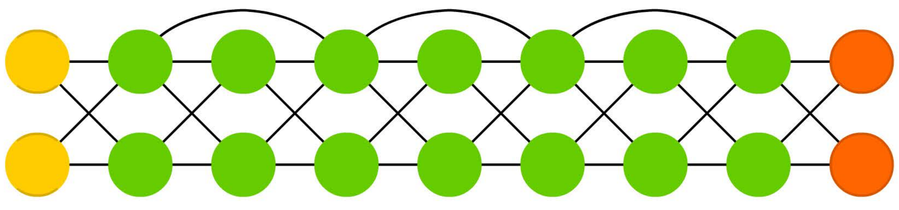
Extreme Learning Machine (ELM)



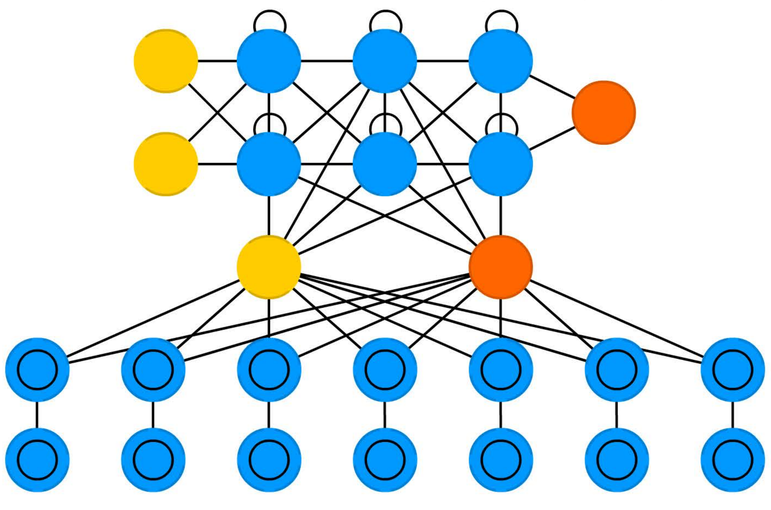
Echo State Network (ESN)



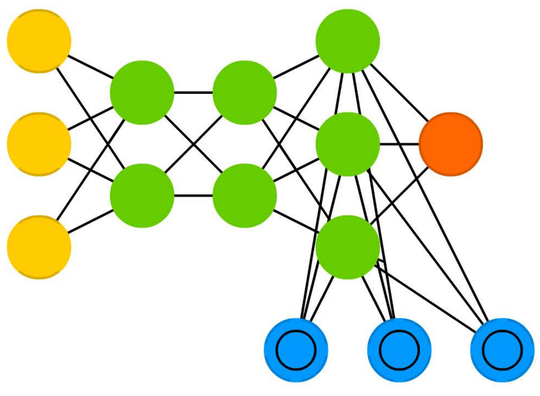
Deep Residual Network (DRN)



Differentiable Neural Computer (DNC)



Neural Turing Machine (NTM)



Capsule Network (CN)

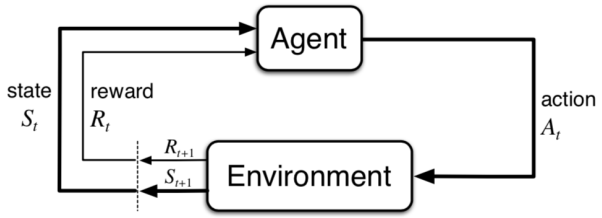
Kohonen Network (KN)

Attention Network (AN)

5. Reinforcement learning

Reinforcement Learning Cheat Sheet

Agent-Environment Interface



The Agent at each step t receives a representation of the environment's *state*, $S_t \in S$ and it selects an action $A_t \in A(s)$. Then, as a consequence of its action the agent receives a *reward*, $R_{t+1} \in R \in \mathbb{R}$.

Policy

A *policy* is a mapping from a state to an action

$$\pi_t(s|a) \tag{1}$$

That is the probability of select an action $A_t = a$ if $S_t = s$.

Reward

The total *reward* is expressed as:

$$G_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{2}$$

Where γ is the *discount factor* and H is the *horizon*, that can be infinite.

Markov Decision Process

A **Markov Decision Process**, MPD, is a 5-tuple (S, A, P, R, γ) where:

$$\begin{aligned}
&\text{finite set of states:}\\
&s \in S\\
&\text{finite set of actions:}\\
&a \in A\\
&\text{state transition probabilities:}\\
&p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\}\\
&\text{expected reward for state-action-nexstate:}\\
&r(s',s,a) = \mathbb{E}[R_{t+1}|S_{t+1} = s', S_t = s, A_t = a]
\end{aligned} \tag{3}$$

Value Function

Value function describes *how good* is to be in a specific state s under a certain policy π. For MDP:

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{4}$$

Informally, is the expected return (expected cumulative discounted reward) when starting from s and following π

Optimal

$$V_*(s) = \max_\pi V_\pi(s) \tag{5}$$

Action-Value (Q) Function

We can also denoted the expected reward for state, action pairs.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] \tag{6}$$

Optimal

The optimal value-action function:

$$q_*(s,a) = \max_\pi q^\pi(s,a) \tag{7}$$

Clearly, using this new notation we can redefine V^*, equation 5, using $q^*(s,a)$, equation 7:

$$V_*(s) = \max_{a \in A(s)} q_{\pi *}(s,a) \tag{8}$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

Bellman Equation

An important recursive property emerges for both Value (4) and Q (6) functions if we expand them.

Value Function

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\left[G_t|S_t = s\right]\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s\right]\\
&= \underbrace{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)}_{\text{Sum of all probabilities } \forall \text{ possible } r}\\
&\quad \left[r + \gamma \underbrace{\mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s'\right]}_{\text{Expected reward from } s_{t+1}}\right]\\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\left[r + \gamma V_\pi(s')\right]
\end{aligned} \tag{9}$$

Similarly, we can do the same for the Q function:

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s, A_t = a\right]\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s'\right]\right]\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_\pi(s')\right]
\end{aligned} \tag{10}$$

Dynamic Programming

Taking advantages of the subproblem structure of the V and Q function we can find the optimal policy by just *planning*

Policy Iteration

We can now find the optimal policy

1. Initialisation
$V(s) \in \mathbb{R}$, (e.g $V(s) = 0$) and $\pi(s) \in A$ for all $s \in S$,
$\Delta \leftarrow 0$
2. Policy Evaluation
while $\Delta \geq \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
3. Policy Improvement
policy-stable $\leftarrow$ *true*
foreach $s \in S$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
  policy-stable $\leftarrow$ *old-action* $= \pi(s)$
end
if *policy-stable* return V $\approx V_*$ and $\pi \approx \pi_*$, else go to 2

Algorithm 1: Policy Iteration

Value Iteration

We can avoid to wait until $V(s)$ has converged and instead do policy improvement and truncated policy evaluation step in one operation

Initialise $V(s) \in \mathbb{R}, \text{e.g} V(s) = 0$
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
ouput: Deterministic policy $\pi \approx \pi_*$ such that $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$

Algorithm 2: Value Iteration

Monte Carlo Methods

Monte Carlo (MC) is a *Model Free* method, It does not require complete knowledge of the environment. It is based on **averaging sample returns** for each state-action pair. The following algorithm gives the basic implementation

Initialise for all $s \in S, a \in A(s)$:
  $Q(s, a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s, a) \leftarrow$ empty list
while *forever* **do**
  Choose $S_0 \in S$ and $A_0 \in A(S_0)$, all pairs have probability > 0
  Generate an episode starting at S_0, A_0 following π
  foreach *pair* s, a *appearing in the episode* **do**
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s, a))$
    $Q(s, a) \leftarrow average(Returns(s, a))$
  end
  foreach s *in the episode* **do**
    $\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$
  end
end

Algorithm 3: Monte Carlo first-visit

For non-stationary problems, the Monte Carlo estimate for, e.g, V is:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{11}$$

Where α is the learning rate, how much we want to forget about past experiences.

Sarsa

Sarsa (State-action-reward-state-action) is a on-policy TD control. The update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

n-step Sarsa

Define the n-step Q-Return

$$q^{(n)} = R_{t+1} + \gamma Rt + 2 + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step Sarsa update $Q(S, a)$ towards the n-step Q-return

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^{(n)} - Q(s_t, a_t)\right]$$

Forward View Sarsa(λ)

$$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

Forward-view Sarsa(λ):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^\lambda - Q(s_t, a_t)\right]$$

Initialise $Q(s, a)$ arbitrarily and
  $Q(terminal - state,) = 0$
foreach *episode* $\in$ *episodes* **do**
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  while s *is not terminal* **do**
    Take action a, observer r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
end

Algorithm 4: Sarsa(λ)

Temporal Difference - Q Learning

Temporal Difference (TD) methods learn directly from raw experience without a model of the environment's dynamics. TD substitutes the expected discounted reward G_t from the episode with an estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{12}$$

The following algorithm gives a generic implementation.

Initialise $Q(s, a)$ arbitrarily and
  $Q(terminal - state,) = 0$
foreach *episode* $\in$ *episodes* **do**
  while s *is not terminal* **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observer r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$
  end
end

Algorithm 5: Q Learning

Deep Q Learning

Created by *DeepMind*, Deep Q Learning, DQL, substitutes the Q function with a deep neural network called *Q-network*. It also keep track of some observation in a *memory* in order to use them to train the network.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\underbrace{(r + \gamma \max_a Q(s', a'; \theta_{i-1})}_{\text{target}} - \underbrace{Q(s, a; \theta_i))}_{\text{prediction}}^2 \right] \tag{13}$$

Where θ are the weights of the network and $U(D)$ is the experience replay history.

Initialise replay memory D with capacity N
Initialise $Q(s, a)$ arbitrarily
foreach *episode* $\in$ *episodes* **do**
  while s *is not terminal* **do**
    With probability ϵ select a random action $a \in A(s)$
    otherwise select $a = \max_a Q(s, a; \theta)$
    Take action a, observer r, s'
    Store transition (s, a, r, s') in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s'_j) from D
    Set $y_j \leftarrow \begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$
    Perform gradient descent step on $(y_j - Q(s_j, a_j; \Theta))^2$
    $s \leftarrow s'$
  end
end

Algorithm 6: Deep Q Learning

6. Programing languages

6.1 Python language

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = “123abc”
list = [value1, value2, …]
dictionary = { key1:value1, key2:value2, …}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict …)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition> :
    <code>
else if <condition> :
    <code>
...
else:
    <code>

if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>

for <variable> in <list>:
    <code>

for <variable> in
range(start,stop,step):
    <code>

for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>):
    <code>
    return <data>
```

Modules

```
import module
module.function()

from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()

f = open(<path>,'r')
for line in f:
    <code>
f.close()

f = open(<path>,'w')
f.write(<str>)
f.close()
```

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Python For Data Science *Cheat Sheet*

Importing Data

Learn Python for data science Interactively at www.DataCamp.com

Importing Data in Python

Most of the time, you'll use either **NumPy** or **pandas** to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r')
>>> text = file.read()
>>> print(file.closed)
>>> file.close()
>>> print(text)
```

Open the file for reading
Read a file's contents
Check whether file is closed
Close file

Using the context manager `with`

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline())
        print(file.readline())
        print(file.readline())
```

Read a single line

Table Data: Flat Files

Importing Flat Files with numpy

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',',
                      skiprows=2,
                      usecols=[0,2],
                      dtype=str)
```

String used to separate values
Skip the first 2 lines
Read the 1st and 3rd column
The type of the resulting array

Files with mixed data types

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True,
                         dtype=None)
```

Look for column header

```
>>> data_array = np.recfromcsv(filename)
```

The default `dtype` of the `np.recfromcsv()` function is `None`.

Importing Flat Files with pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5,
                       header=None,
                       sep='\t',
                       comment='#',
                       na_values[""])
```

Number of rows of file to read
Row number to use as col names
Delimiter to use
Character to split comments
String to recognize as NA/NaN

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                                  'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

SAS Files

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

Stata Files

```
>>> data = pd.read_stata('urbanpop.dta')
```

Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager `with`

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype
>>> data_array.shape
>>> len(data_array)
```

Data type of array elements
Array dimensions
Length of array

pandas DataFrames

```
>>> df.head()
>>> df.tail()
>>> df.index
>>> df.columns
>>> df.info()
>>> data_array = data.values
```

Return first DataFrame rows
Return last DataFrame rows
Describe index
Describe DataFrame columns
Info on DataFrame
Convert a DataFrame to an a NumPy array

Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

Exploring Dictionaries

Accessing Elements with Functions

```
>>> print(mat.keys())
>>> for key in data.keys():
        print(key)
meta
quality
strain
>>> pickled_data.values()
>>> print(mat.items())
```

Print dictionary keys
Print dictionary keys

Return dictionary values
Returns items in list format of (key, value) tuple pairs

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys()
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
>>> print(data['meta']['Description'].value)
```

Explore the HDF5 structure

Retrieve the value for a key

Navigating Your FileSystem

Magic Commands

```
!ls
%cd ..
%pwd
```

List directory contents of files and directories
Change current working directory
Return the current working directory path

`os` Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd()
>>> os.listdir(wd)
>>> os.chdir(path)
>>> os.rename("test1.txt",
              "test2.txt")
>>> os.remove("test1.txt")
>>> os.mkdir("newdir")
```

Store the name of current directory in a string
Output contents of the directory in a list
Change current working directory
Rename a file

Delete an existing file
Create a new directory

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

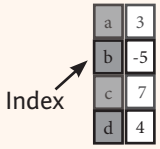
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

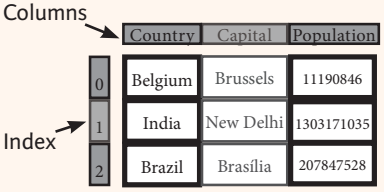
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5

>>> df[1:]
   Country    Capital  Population
1   India  New Delhi  1303171035
2  Brazil   Brasília   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat([0],[0])
'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at([0], ['Country'])
'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital    Brasília
Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series `s` where value is not >1
`s` where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b      NaN
  c      5.0
  d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c    5.0
  d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Python For Data Science *Cheat Sheet*

Pandas

Learn Python for Data Science **Interactively** at www.DataCamp.com

Reshaping Data

Pivot

```
>>> df3= df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

Spread rows into columns

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,
                         values='Value',
                         index='Date',
                         columns='Type'])
```

Spread rows into columns

Stack / Unstack

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

Pivot a level of column labels
Pivot a level of index labels

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

Melt

```
>>> pd.melt(df2,
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

Gather columns into rows

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

Iteration

```
>>> df.iteritems()
>>> df.iterrows()
```

(Column-index, Series) pairs
(Row-index, Series) pairs

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()]
>>> df3.loc[:,(df3>1).all()]
>>> df3.loc[:,df3.isnull().any()]
>>> df3.loc[:,df3.notnull().all()]
```

Select cols with any vals >1
Select cols with vals > 1
Select cols with NaN
Select cols without NaN

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]
>>> df3.filter(items="a","b"])
>>> df.select(lambda x: not x%5)
```

Find same elements
Filter on values
Select specific elements

Where

```
>>> s.where(s > 0)
```

Subset the data

Query

```
>>> df6.query('second > first')
```

Query DataFrame

Setting/Resetting Index

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

Set the index
Reset the index
Rename DataFrame

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528
3	Brazil	Brasília	207847528

Backward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

0	3
1	3
2	3
3	3
4	3

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

Duplicate Data

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

Return unique values
Check duplicates
Drop duplicates
Check index duplicates

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),
                              'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "f")
```

Drop NaN values
Fill NaN values with a predetermined value
Replace values with others

Combining Data

data1

	X1	X2
a		11.432
b		1.303
c		99.906

data2

	X1	X3
a		20.784
b		NaN
d		20.784

Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

	X1	X2	X3
a		11.432	20.784
b		1.303	NaN
c		99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

	X1	X2	X3
a		11.432	20.784
b		1.303	NaN
d		NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

	X1	X2	X3
a		11.432	20.784
b		1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

	X1	X2	X3
a		11.432	20.784
b		1.303	NaN
c		99.906	NaN
d		NaN	20.784

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

Dates

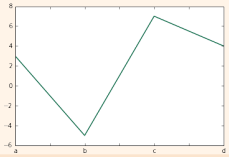
```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

Visualization

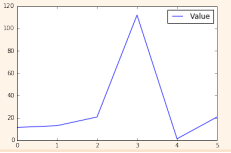
Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```



```
>>> df2.plot()
>>> plt.show()
```



Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

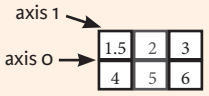
Use the following import convention:

```
>>> import numpy as np
```

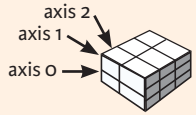
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                    Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)    Create an array of ones
>>> d = np.arange(10,25,5)             Create an array of evenly
                                       spaced values (step value)
>>> np.linspace(0,2,9)                 Create an array of evenly
                                       spaced values (number of samples)
>>> e = np.full((2,2),7)               Create a constant array
>>> f = np.eye(2)                      Create a 2X2 identity matrix
>>> np.random.random((2,2))            Create an array with random values
>>> np.empty((3,2))                    Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape             Array dimensions
>>> len(a)              Length of array
>>> b.ndim              Number of array dimensions
>>> e.size              Number of array elements
>>> b.dtype             Data type of array elements
>>> b.dtype.name        Name of data type
>>> b.astype(int)       Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                          Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                            Array-wise sum
>>> a.min()                            Array-wise minimum value
>>> b.max(axis=0)                      Maximum value of an array row
>>> b.cumsum(axis=1)                   Cumulative sum of the elements
>>> a.mean()                           Mean
>>> b.median()                         Median
>>> a.corrcoef()                       Correlation coefficient
>>> np.std(b)                          Standard deviation
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                               Select the element at the 2nd index
  3
>>> b[1,2]                             Select the element at row 1 column 2
  6.0                                  (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                             Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                           Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                              Select all items at row 0
  array([[1.5, 2., 3.]])               (equivalent to b[0:1, :])
>>> c[1,...]                           Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                          Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                             Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]      Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]      Select a subset of the matrix's rows
  array([[ 4. ,  5. ,  6. ,  4. ],     and columns
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)                Permute array dimensions
>>> i.T                                Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                          Flatten the array
>>> g.reshape(3,-2)                    Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                    Return a new array with shape (2,6)
>>> np.append(h,g)                     Append items to an array
>>> np.insert(a, 1, 5)                 Insert items in an array
>>> np.delete(a,[1])                   Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)       Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                   Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                         Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                   Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))             Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                         Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                     Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]   index
>>> np.vsplit(c,2)                     Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

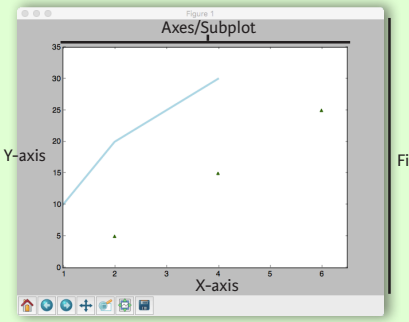
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

`>>> f, ax = plt.subplots(figsize=(5,6))`	Create a figure and one subplot

Seaborn styles

`>>> sns.set()`	(Re)set the seaborn default
`>>> sns.set_style("whitegrid")`	Set the matplotlib parameters
`>>> sns.set_style("ticks", {"xtick.major.size":8, "ytick.major.size":8})`	Set the matplotlib parameters
`>>> sns.axes_style("whitegrid")`	Return a dict of params or use with `with` to temporarily set the style

Context Functions

`>>> sns.set_context("talk")`	Set context to `"talk"`
`>>> sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":2.5})`	Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

`>>> sns.set_palette("husl",3)`	Define the color palette
`>>> sns.color_palette("husl")`	Use with `with` to temporarily set palette
`>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]`	
`>>> sns.set_palette(flatui)`	Set your own color palette

3 Plotting With Seaborn

Axis Grids

`>>> g = sns.FacetGrid(titanic, col="survived", row="sex")`	Subplot grid for plotting conditional relationships
`>>> g = g.map(plt.hist,"age")`	
`>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)`	Draw a categorical plot onto a Facetgrid
`>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)`	Plot data and regression model fits across a FacetGrid
`>>> h = sns.PairGrid(iris)`	Subplot grid for plotting pairwise relationships
`>>> h = h.map(plt.scatter)`	Plot pairwise bivariate distributions
`>>> sns.pairplot(iris)`	Grid for bivariate plot with marginal univariate plots
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')`	Plot bivariate distribution

Categorical Plots

Scatterplot

`>>> sns.stripplot(x="species", y="petal_length", data=iris)`	Scatterplot with one categorical variable
`>>> sns.swarmplot(x="species", y="petal_length", data=iris)`	Categorical scatterplot with non-overlapping points

Bar Chart

`>>> sns.barplot(x="sex", y="survived", hue="class", data=titanic)`	Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

`>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")`	Show count of observations

Point Plot

`>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male":"g", "female":"m"}, markers=["^","o"], linestyles=["-","--"])`	Show point estimates and confidence intervals as rectangular bars

Boxplot

`>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)`	Boxplot
`>>> sns.boxplot(data=iris,orient="h")`	Boxplot with wide-form data

Violinplot

`>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)`	Violin plot

Regression Plots

`>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)`	Plot data and a linear regression model fit

Distribution Plots

`>>> plot = sns.distplot(data.y, kde=False, color="b")`	Plot univariate distribution

Matrix Plots

`>>> sns.heatmap(uniform_data,vmin=0,vmax=1)`	Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

`>>> g.despine(left=True)`	Remove left spine
`>>> g.set_ylabels("Survived")`	Set the labels of the y-axis
`>>> g.set_xticklabels(rotation=45)`	Set the tick labels for x
`>>> g.set_axis_labels("Survived", "Sex")`	Set the axis labels
`>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])`	Set the limit and ticks of the x-and y-axis

Plot

`>>> plt.title("A Title")`	Add plot title
`>>> plt.ylabel("Survived")`	Adjust the label of the y-axis
`>>> plt.xlabel("Sex")`	Adjust the label of the x-axis
`>>> plt.ylim(0,100)`	Adjust the limits of the y-axis
`>>> plt.xlim(0,10)`	Adjust the limits of the x-axis
`>>> plt.setp(ax,yticks=[0,5])`	Adjust a plot property
`>>> plt.tight_layout()`	Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

`>>> plt.show()`	Show the plot
`>>> plt.savefig("foo.png")`	Save the plot as a figure
`>>> plt.savefig("foo.png", transparent=True)`	Save transparent figure

Close & Clear

Also see Matplotlib

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear an entire figure
`>>> plt.close()`	Close a window

Python For Data Science *Cheat Sheet*

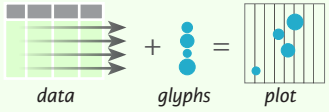
Bokeh

Learn Bokeh Interactively at www.DataCamp.com, taught by Bryan Van de Ven, core contributor

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:

Python lists, NumPy arrays, Pandas DataFrames and other sequences of values

2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                          Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",      Step 2
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)   Step 3
>>> output_file("lines.html")                    Step 4
>>> show(p)                                      Step 5
```

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

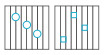
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
                x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

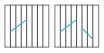
3 Renderers & Visual Customizations

Glyphs



Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```



Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

Colormapping

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
             color=dict(field='origin',
                        transform=color_mapper),
             legend='Origin')
```

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" ,[p1, r1]),("Two",[r2])],
                    location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

Columns

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Linked Plots

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100,
                tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200,
                tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

4 Output & Export

Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

HTML

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")
```

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> save(p1)
```

```
>>> show(layout)
>>> save(layout)
```

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]          Create a dense meshgrid
>>> np.ogrid[0:2,0:2]          Create an open meshgrid
>>> np.r_[[3,[0]*5,-1:1:10j]   Stack arrays vertically (row-wise)
>>> np.c_[b,c]                 Create stacked column-wise arrays
```

Shape Manipulation

```
>>> np.transpose(b)            Permute array dimensions
>>> b.flatten()                Flatten the array
>>> np.hstack((b,c))           Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))           Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)             Split the array horizontally at the 2nd index
>>> np.vpslit(d,2)             Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])        Create a polynomial object
```

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)       Vectorize functions
```

Type Handling

```
>>> np.real(c)                 Return the real part of the array elements
>>> np.imag(c)                 Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)  Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)        Cast object to a data type
```

Other Useful Functions

```
>>> np.angle(b,deg=True)       Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)  Create an array of evenly spaced values
                                    (number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g)               Unwrap
>>> np.logspace(0,10,3)        Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])     Return values from a list of arrays depending on
                               conditions
>>> misc.factorial(a)          Factorial
>>> misc.comb(10,3,exact=True) Combine N things taken at k time
>>> misc.central_diff_weights(3)  Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0)   Find the n-th derivative of a function at a point
```

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I                        Inverse
>>> linalg.inv(A)              Inverse
>>> A.T                        Tranpose matrix
>>> A.H                        Conjugate transposition
>>> np.trace(A)                Trace
```

Norm

```
>>> linalg.norm(A)             Frobenius norm
>>> linalg.norm(A,1)           L1 norm (max column sum)
>>> linalg.norm(A,np.inf)      L inf norm (max row sum)
```

Rank

```
>>> np.linalg.matrix_rank(C)   Matrix rank
```

Determinant

```
>>> linalg.det(A)              Determinant
```

Solving linear problems

```
>>> linalg.solve(A,b)          Solver for dense matrices
>>> E = np.mat(a).T            Solver for dense matrices
>>> linalg.lstsq(D,E)          Least-squares solution to linear matrix
                               equation
```

Generalized inverse

```
>>> linalg.pinv(C)             Compute the pseudo-inverse of a matrix
                               (least-squares solver)
>>> linalg.pinv2(C)            Compute the pseudo-inverse of a matrix
                               (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)         Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)   Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)   Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)   Dictionary Of Keys matrix
>>> E.todense()                Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A)   Identify sparse matrix
```

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I)       Inverse
```

Norm

```
>>> sparse.linalg.norm(I)      Norm
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) Solver for sparse matrices
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)      Sparse matrix exponential
```

Matrix Functions

Addition

```
>>> np.add(A,D)                Addition
```

Subtraction

```
>>> np.subtract(A,D)           Subtraction
```

Division

```
>>> np.divide(A,D)             Division
```

Multiplication

```
>>> np.multiply(D,A)           Multiplication
>>> np.dot(A,D)                Dot product
>>> np.vdot(A,D)               Vector dot product
>>> np.inner(A,D)              Inner product
>>> np.outer(A,D)              Outer product
>>> np.tensordot(A,D)          Tensor dot product
>>> np.kron(A,D)               Kronecker product
```

Exponential Functions

```
>>> linalg.expm(A)             Matrix exponential
>>> linalg.expm2(A)            Matrix exponential (Taylor Series)
>>> linalg.expm3(D)            Matrix exponential (eigenvalue
                               decomposition)
```

Logarithm Function

```
>>> linalg.logm(A)             Matrix logarithm
```

Trigonometric Tunctions

```
>>> linalg.sinm(D)             Matrix sine
>>> linalg.cosm(D)             Matrix cosine
>>> linalg.tanm(A)             Matrix tangent
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)            Hypberbolic matrix sine
>>> linalg.coshm(D)            Hyperbolic matrix cosine
>>> linalg.tanhm(A)            Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.sigm(A)                 Matrix sign function
```

Matrix Square Root

```
>>> linalg.sqrtm(A)            Matrix square root
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)  Evaluate matrix function
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)      Solve ordinary or generalized
                               eigenvalue problem for square matrix
>>> l1, l2 = la                Unpack eigenvalues
>>> v[:,0]                     First eigenvector
>>> v[:,1]                     Second eigenvector
>>> linalg.eigvals(A)          Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)     Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  Construct sigma matrix in SVD
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C)       LU Decomposition
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)  Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)         SVD
```

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

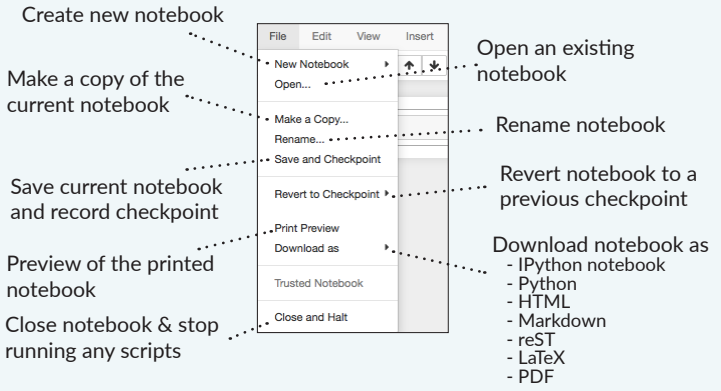
```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Python For Data Science *Cheat Sheet*

Jupyter Notebook

Learn More Python for Data Science Interactively at www.DataCamp.com

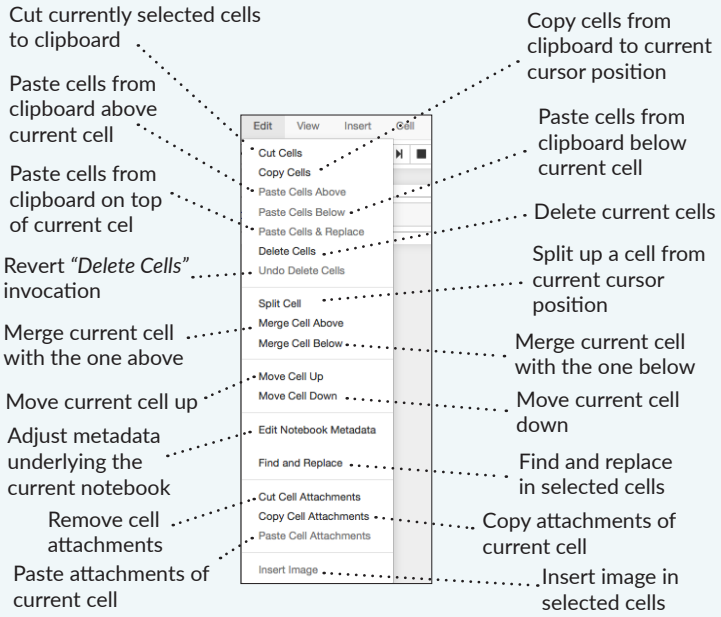
Saving/Loading Notebooks



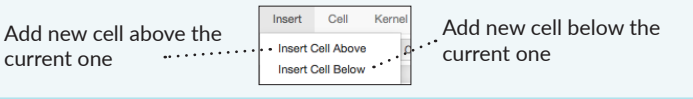
Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells



Insert Cells

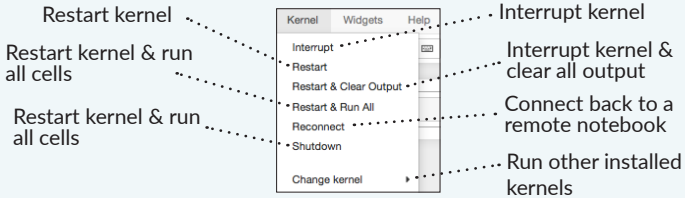


Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython IRkernel IJulia

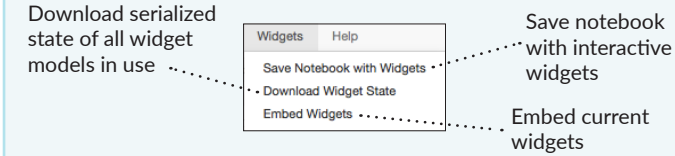
Installing Jupyter Notebook will automatically install the IPython kernel.



Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

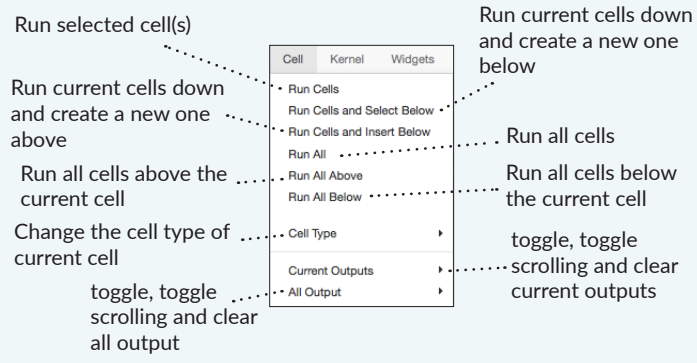


Command Mode:

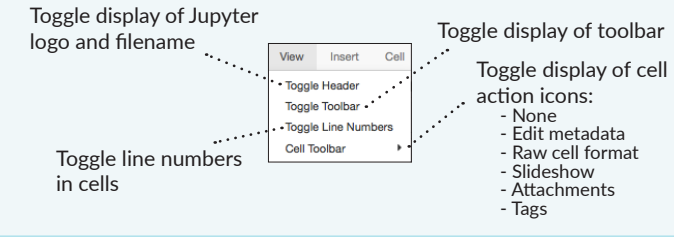
Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

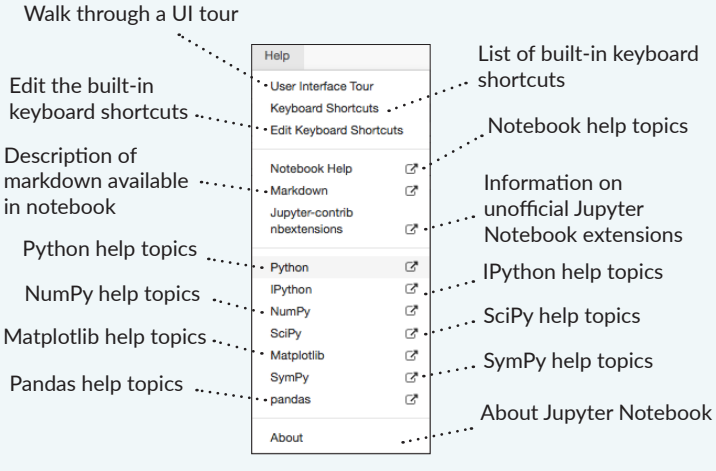
Executing Cells



View Cells



Asking For Help



Jupyter Notebook Markdown Cheatsheet

From SqlBak.com with 💙

`#␣Header 1` `Header 1` `========`	# Header 1
`##␣Header 2` `Header 2` `--------`	## Header 2
`###␣Header 3`	### Header 3
`####␣Header 4`	#### Header 4
`#####␣Header 5`	##### *Header 5*
`*italics*` `_italics_`	*italics*
`\*literal asterisks\*`	*literal asterisks*
`**bold**` `__bold__`	**bold**
`~~strikethrough~~`	~~strikethrough~~
`1.␣First item` `2.␣Second item` `␣␣1.␣Subitem`	1. First item 2. Second item A. Subitem
`*␣Item 1` `␣Indent` `-␣Item 2` `␣+␣Item 3`	• Item 1 Indent • Item 2 ▪ Item 3
`- [x] Done` `- [ ] To do`	• ☑ Done • ☐ To do
`A` `Line␣␣` `Break`	A Line Break
`---` `* * *`	--- * * *

`<a id="anchor"></a>` `[Go to anchor](#anchor)` `#␣Top Header` `[Go to header](#Top-Header)`	Go to anchor
`https://sqlbak.com` `[Link](https://sqlbak.com "optional title")` `Click [here][id]` `[id]:https://sqlbak.com`	Link
`> blockquote text`	> blockquote text
`` ```python print('hello'); ``` `inline_code();` ````	`print('hello');`
`|Left  |Center|Right|` `|:-----|:----:|----:|` `|1  |A  |C  |` `|2  |B  |D  |`	Left / Center / Right 1 / A / C 2 / B / D
`![alt text](logo.png "Title")` `![][id]` `[id]:logo.png "Title"`	
`$$\sqrt{k}$$` `Inline: $\sqrt{k}$`	$\sqrt{k}$
`[![Img Alt Text](http://img.youtube.com/vi/aZCXOw707nc/0.jpg)](https://youtu.be/aZCXOw707nc "Video Title")`	

Cheatography

Natural Language Processing with Python & nltk Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15485/

Handling Text

`text='Some words'`	assign string
`list(text)`	Split text into character tokens
`set(text)`	Unique tokens
`len(text)`	Number of characters

Accessing corpora and lexical resources

`from nltk.corpus import brown`	import CorpusReader object
`brown.words(text_id)`	Returns pretokenised document as list of words
`brown.fileids()`	Lists docs in Brown corpus
`brown.categories()`	Lists categories in Brown corpus

Tokenization

`text.split(" ")`	Split by space
`nltk.word_tokenizer(text)`	nltk in-built word tokenizer
`nltk.sent_tokenize(doc)`	nltk in-built sentence tokenizer

Lemmatization & Stemming

`input="List listed lists listing listings"`	Different suffixes
`words=input.lower().split(' ')`	Normalize (lowercase) words
`porter=nltk.PorterStemmer`	Initialise Stemmer
`[porter.stem(t) for t in words]`	Create list of stems
`WNL=nltk.WordNetLemmatizer()`	Initialise WordNet lemmatizer
`[WNL.lemmatize(t) for t in words]`	Use the lemmatizer

Part of Speech (POS) Tagging

`nltk.help.upenn_tagset('MD')`	Lookup definition for a POS tag
`nltk.pos_tag(words)`	nltk in-built POS tagger
	<use an alternative tagger to illustrate ambiguity>

Sentence Parsing

`g=nltk.data.load('grammar.cfg')`	Load a grammar from a file
`g=nltk.CFG.fromstring("""..."")`	Manually define grammar
`parser=nltk.ChartParser(g)`	Create a parser out of the grammar
`trees=parser.parse_all(text)`	
`for tree in trees: ... print tree`	
`from nltk.corpus import treebank`	
`treebank.parsed_sents('wsj_0001.mrg')`	Treebank parsed sentences

Text Classification

`from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer`	
`vect=CountVectorizer().fit(X_train)`	Fit bag of words model to data
`vect.get_feature_names()`	Get features
`vect.transform(X_train)`	Convert to doc-term matrix

Entity Recognition (Chunking/Chinking)

`g="NP: {<DT>?<JJ>*<NN>}"`	Regex chunk grammar
`cp=nltk.RegexpParser(g)`	Parse grammar
`ch=cp.parse(pos_sent)`	Parse tagged sent. using grammar
`print(ch)`	Show chunks
`ch.draw()`	Show chunks in IOB tree
`cp.evaluate(test_sents)`	Evaluate against test doc
`sents=nltk.corpus.treebank.tagged_sents()`	
`print(nltk.ne_chunk(sent))`	Print chunk tree

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 28th May, 2018.
Last updated 29th May, 2018.

RegEx with Pandas & Named Groups

```
df=pd.DataFrame(time_sents, columns=['text'])
df['text'].str.split().str.len()
df['text'].str.contains('word')
df['text'].str.count(r'\d')
df['text'].str.findall(r'\d')
df['text'].str.replace(r'\w+day\b', '???')
df['text'].str.replace(r'(\w)', lambda x: x.groups()
[0][:3])
df['text'].str.extract(r'(\d?\d):(\d\d)')
df['text'].str.extractall(r'((\d?\d):(\d\d) ?
([ap]m))')
df['text'].str.extractall(r'(?P<digits>\d)')
```

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 28th May, 2018.
Last updated 29th May, 2018.

Cheatography

spaCy Cheat Sheet

by Nuozhi via cheatography.com/122797/cs/22963/

Init

```
from spacy.lang.en import English
nlp = English()
```

Basic

```
doc = nlp("SOME TEXTS")
span = doc[i:j]
token = doc[i]
```

Pre-trained Model

```
nlp = spacy.load('en_core_web_sm')
doc = nlp(MY_TEXT)
```

Name entity

```
doc.ents
```

.text
.label_

spacy.tokens

Doc	`Doc(nlp.vocab, words=words, spaces = spaces)`
Span	`Span(doc, i, j, label="PERSON")`

index: i, j
words: a collection of words
spaces: a collecture of booleans

Matcher

```
matcher = spacy.matcher.Matcher(nlp.vocab)
matches = matcher(doc)
```

[(id, start, end)]

Add pattern to matcher

```
pattern = [ { key: value } ]
matcher.add("PATTERN_NAME", None, pattern)
```

Two types of key:
1. regex pattern
2. label (i.e. POS, entity)

Phrase matching

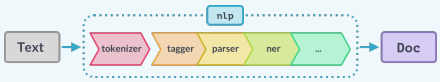
```
matcher = spacy.matcher.PhraseMatcher(nlp.vocab)
pattern = nlp("Golden Retriever")
matcher.add("DOG", None, pattern)
for match_id, start, end in matcher(doc):
    span = doc[start:end]
```

Similarity

word vector	token.vector
Doc	`doc1.similarity(doc2)`
Span	`span1.similarity(span2)`
Token	`token1.similarity(token2)`
Doc by Token	`doc.similarity(token)`

return a similarity score 0~1
NOT for small model
cosine similarity by default

Pipeline




```
nlp.pipe_names
nlp.pipeline
```

Add pipeline component

```
def fn(doc):
    # function body
    return doc
nlp.add_pipe(fn, last, first, before, after)
```

Set custom attributes

add metadata	`doc._.ATTR = "ATTRIBUTE NAME"`
register globally	`Doc.set_extension("ATTR", default=None)`

set to doc, tokens, spans
access property via ._

Extension attribute types

attribute	`Token.set_extension("ATTR", default=Bool)`
property	`Span.set_extension("PROP", getter=fn)`
method	`Doc.set_extension("METHOD", method=fn)`

Boost up

```
nlp.pipe(DATA)
```

Passing in context

```
data = [ ("SOME TEXTS", {"KEY": "VAL"}), (...), ]
# Method 1
for doc, ctx in nlp.pipe(data, as_tuple=True):
    print( doc.ATTR, ctx[KEY] )
# Method 2
Doc.set_extension("KEY", default=None)
for doc, ctx in nlp.pipe(data, as_tuples=True):
    doc._.KEY = ctx["KEY"]
```

Using tokenizer only

```
# Method 1
doc = nlp.make_doc("SOME TEXTS")
# Method 2
with nlp.disable_pipes("tagger", "parser"):
    doc = nlp(text)
```


By **Nuozhi**
cheatography.com/nuozhi/

Not published yet.
Last updated 24th May, 2020.
Page 1 of 1.

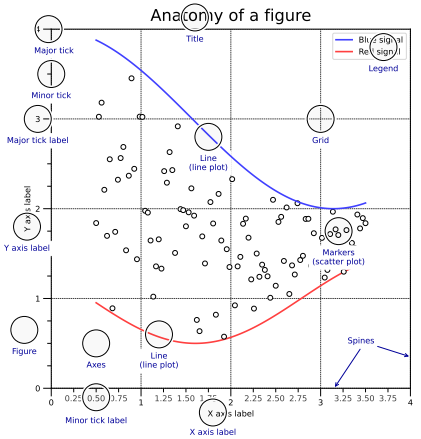
matplotlib Cheat sheet

Version 3.5.0

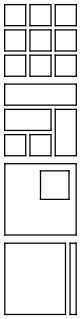
Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout



```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X],Y,[fmt],…)` API
X, **Y**, fmt, color, marker, linestyle

`scatter(X,Y,…)` API
X, **Y**, [s]izes, [c]olors, marker, cmap

`bar[h](x,height,…)` API
x, **height**, width, bottom, align, color

`imshow(Z,…)` API
Z, cmap, interpolation, extent, origin

`contour[f]([X],[Y],Z,…)` API
X, Y, **Z**, levels, colors, extent, origin

`pcolormesh([X],[Y],Z,…)` API
X, Y, **Z**, vmin, vmax, cmap

`quiver([X],[Y],U,V,…)` API
X, Y, **U**, **V**, C, units, angles

`pie(X,…)` API
Z, explode, labels, colors, radius

`text(x,y,text,…)` API
x, **y**, **text**, va, ha, size, weight, transform

`fill[_between][x](…)` API
X, Y1, Y2, color, where

Advanced plots

`step(X,Y,[fmt],…)` API
X, **Y**, fmt, color, marker, where

`boxplot(X,…)` API
X, notch, sym, bootstrap, widths

`errorbar(X,Y,xerr,yerr,…)` API
X, **Y**, xerr, yerr, fmt

`hist(X, bins, …)` API
X, bins, range, density, weights

`violinplot(D,…)` API
D, positions, widths, vert

`barbs([X],[Y], U, V, …)` API
X, Y, **U**, **V**, C, length, pivot, sizes

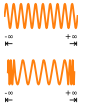
`eventplot(positions,…)` API
positions, orientation, lineoffsets

`hexbin(X,Y,C,…)` API
X, **Y**, C, gridsize, bins

Scales

API

```
ax.set_[xy]scale(scale,…)
```



Projections

API

```
subplot(…,projection=p)
```

p='polar'

p='3d'

p=Orthographic() API
from cartopy.crs import Cartographic

Lines

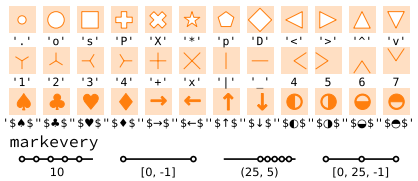
API

```
linestyle or ls
```


```
capstyle or dash_capstyle
```

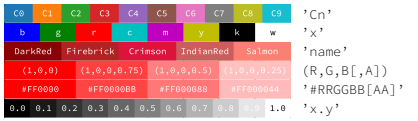
Markers

API



Colors

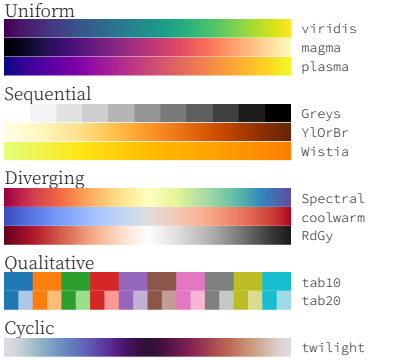
API



Colormaps

API

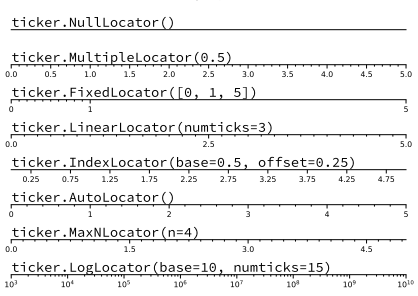
```
plt.get_cmap(name)
```



Tick locators

API

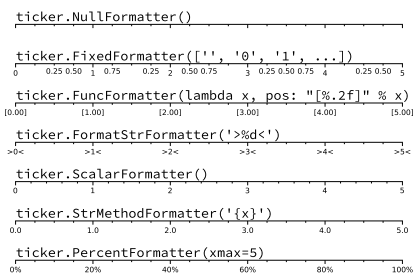
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



Tick formatters

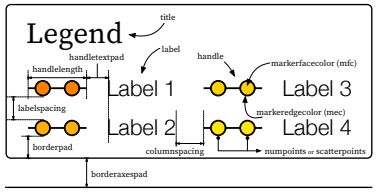
API

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

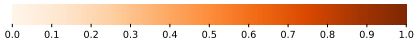


Ornaments

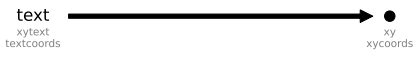
`ax.legend(…)` API
handles, labels, loc, title, frameon



`ax.colorbar(…)` API
mappable, ax, cax, orientation



`ax.annotate(…)` API
text, **xy**, **xytext**, xycoords, textcoords, arrowprops



Event handling

API

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

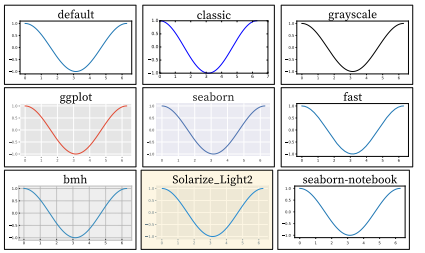
API

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

API

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.patch.set_alpha(0)
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(list)
ax.set_[xy]ticklabels(list)
ax.set_[sup]title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
fig.patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

API

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

Ten simple rules

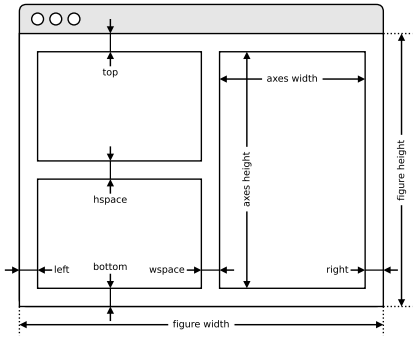
READ

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

Axes adjustments

API

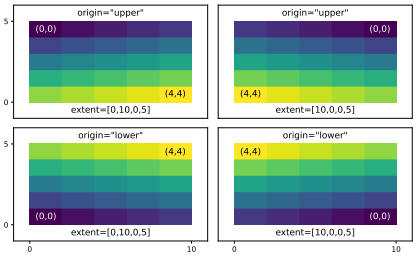
plt.**subplots_adjust**(...)



Extent & origin

API

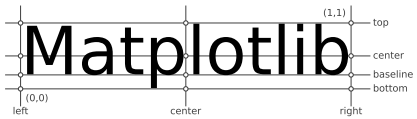
ax.**imshow**(extent=..., origin=...)



Text alignments

API

ax.**text**(..., ha=... , va=..., ...)



Text parameters

API

ax.**text**(..., family=..., size=..., weight=...)
ax.**text**(..., fontproperties=...)

The quick brown fox — xx-large (1.73)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

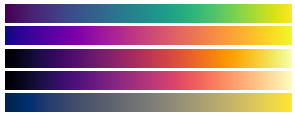
The quick brown fox jumps over the lazy dog — black (900)
The quick brown fox jumps over the lazy dog — bold (700)
The quick brown fox jumps over the lazy dog — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
The quick brown fox jumps over the lazy dog — cursive

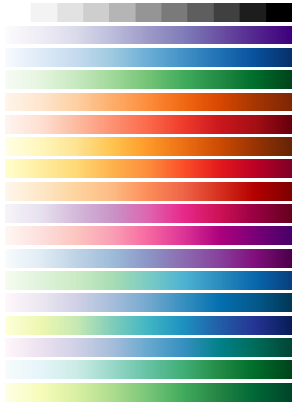
The quick brown fox jumps over the lazy dog — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

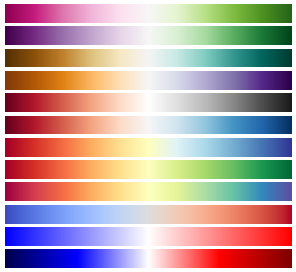
Uniform colormaps



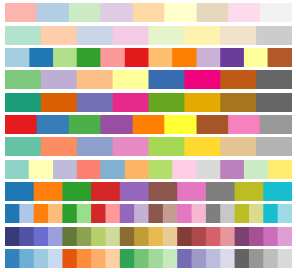
Sequential colormaps



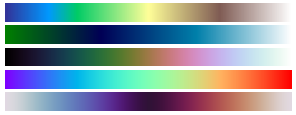
Diverging colormaps



Qualitative colormaps



Miscellaneous colormaps

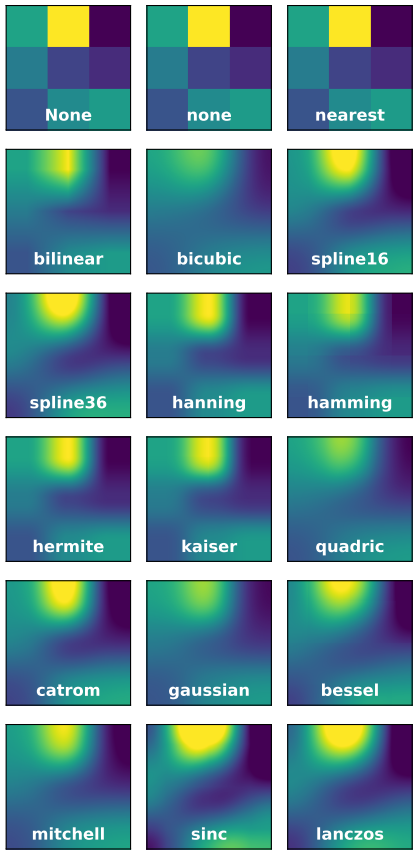


Color names

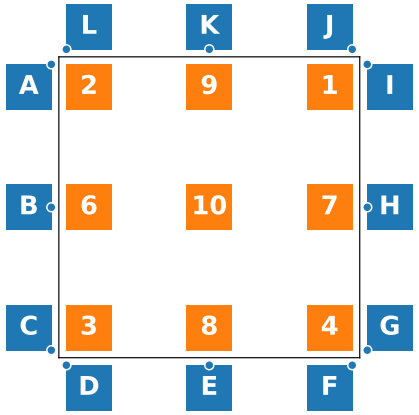
API

Image interpolation

API



Legend placement



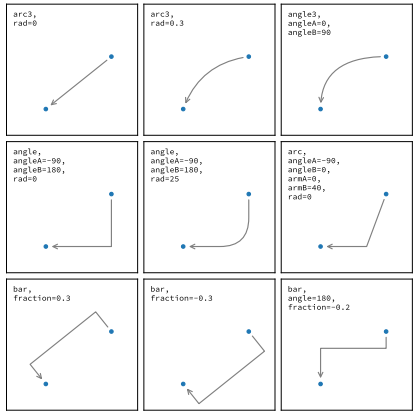
ax.**legend**(loc="string", bbox_to_anchor=(x,y))

2: upper left — 9: upper center — 1: upper right
6: center left — 10: center — 7: center right
3: lower left — 8: lower center — 4: lower right

A: upper right / (-0.1,0.9) — B: center right / (-0.1,0.5)
C: lower right / (-0.1,0.1) — D: upper left / (0.1,-0.1)
E: upper center / (0.5,-0.1) — F: upper right / (0.9,-0.1)
G: lower left / (1.1,0.1) — H: center left / (1.1,0.5)
I: upper left / (1.1,0.9) — J: lower right / (0.9,1.1)
K: lower center / (0.5,1.1) — L: lower left / (0.1,1.1)

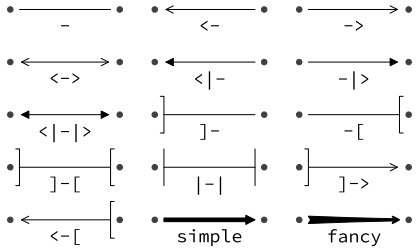
Annotation connection styles

API



Annotation arrow styles

API



How do I ...

... resize a figure?
→ fig.set_size_inches(w, h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure/an axes?
→ fig.clear() → ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_[xy]ticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.set_[xy]ticks(rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                          slow
plot(X, Y, marker="o", ls="")          fast

for i in range(n): plot(X[i])          slow
plot(sum([x+[None] for x in X],[]))    fast

cla(), imshow(…), canvas.draw()        slow
im.set_data(…), canvas.draw()          fast
```

Beyond Matplotlib

Seaborn: Statistical Data Visualization
Cartopy: Geospatial Data Processing
yt: Volumetric data Visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A Grammar of Graphics for Python

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

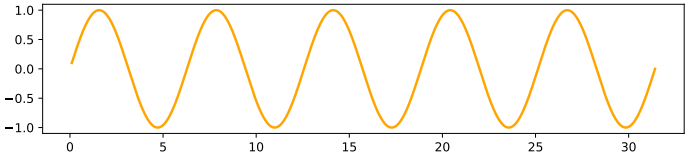
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

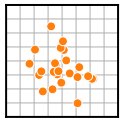
4 Observe



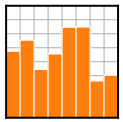
Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



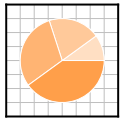
```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```



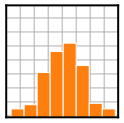
```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

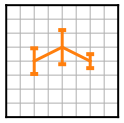
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```



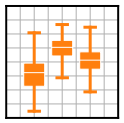
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



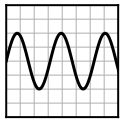
```
Z = np.random.normal(0, 1, (100,3))

ax.boxplot(Z)
```



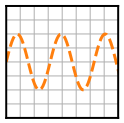
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

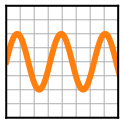
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



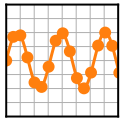
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



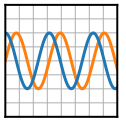
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



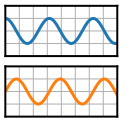
Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named *Axes*):

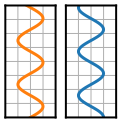
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

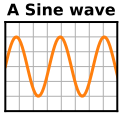


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

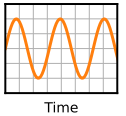


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

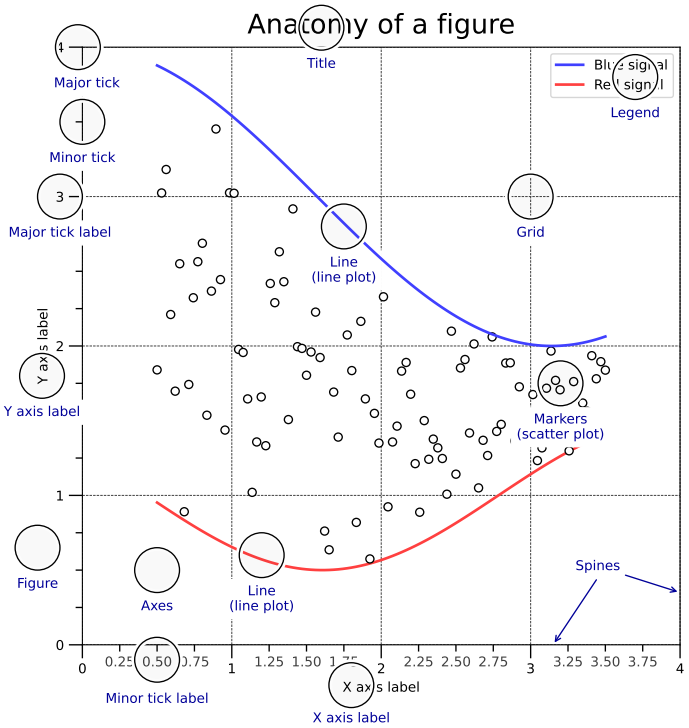
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

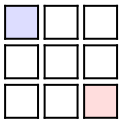
Matplotlib for intermediate users

A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.

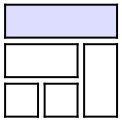


Figure, axes & spines

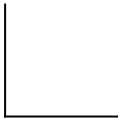
```
fig, axs = plt.subplots((3,3))
axs[0,0].set_facecolor("#ddddff")
axs[2,2].set_facecolor("#ffffdd")
```



```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```

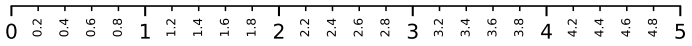


```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```



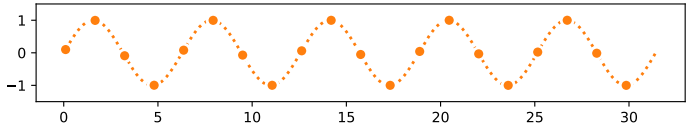
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```



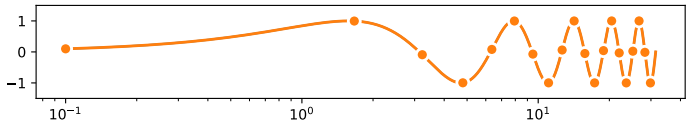
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=25, mec="1.0")
```



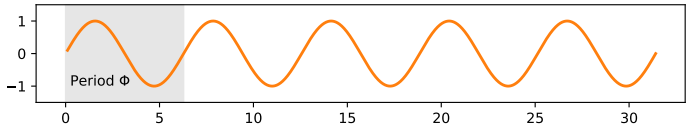
Scales & projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=25, mec="1.0")
```



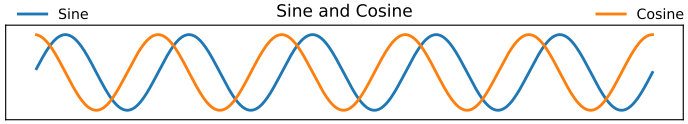
Text & ornaments

```
ax.fill_betweenx([-1,1],[0],[2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



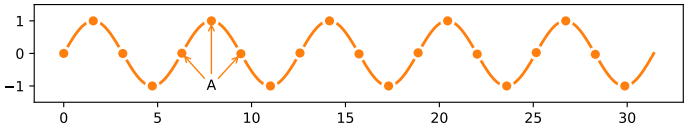
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1),ncol=2,
          mode="expand",  loc="lower left")
```



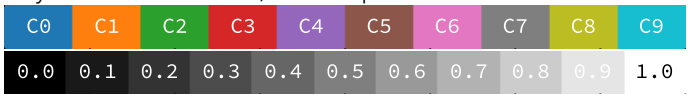
Annotation

```
ax.annotate("A", (X[250],Y[250]),(X[250],-1),
  ha="center", va="center",arrowprops =
  {"arrowstyle" : "->", "color": "C1"})
```



Colors

Any color can be used, but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-columns A4 paper with 2cm margins on each side and a column separation of 1cm. The width of a figure is (21 - 2*2 - 1)/2 = 8cm. One inch being 2.54cm, figure size should be 3.15×3.15 in.

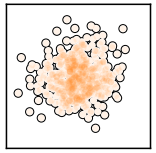
```
fig = plt.figure(figsize=(3.15,3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Matplotlib tips & tricks

Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density. Multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1, 1, 500)
Y = np.random.normal(-1, 1, 500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1",  lw=0, alpha=0.1)
```



Rasterization

If your figure has many graphical elements, such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

Offline rendering

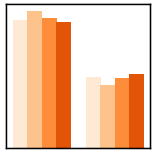
Use the Agg backend to render a figure directly in an array.

```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure()))
... # draw som stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

Range of continuous colors

You can use colormap to pick from a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Oranges")
colors = cmap([0.2, 0.4, 0.6, 0.8])

ax.hist(X, 2, histtype='bar', color=colors)
```



Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
  fx.Stroke(linewidth=3, foreground='1.0'),
  fx.Normal()])
```

Multiline plot

You can plot several lines at once using None as separator.

```
X,Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
  X.extend([x, x, None]), Y.extend([0, sin(x), None])
ax.plot(X, Y, "black")
```

Dotted lines

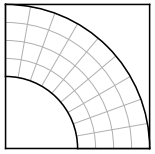
To have rounded dotted lines, use a custom `linestyle` and modify `dash_capstyle`.

```
ax.plot([0,1], [0,0], "C1",
        linestyle = (0, (0.01, 1)), dash_capstyle="round")
ax.plot([0,1], [1,1], "C1",
        linestyle = (0, (0.01, 2)), dash_capstyle="round")
```

Combining axes

You can use overlaid axes with different projections.

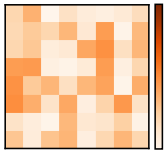
```
ax1 = fig.add_axes([0,0,1,1],
                   label="cartesian")
ax2 = fig.add_axes([0,0,1,1],
                   label="polar",
                   projection="polar")
```



Colorbar adjustment

You can adjust a colorbar's size when adding it.

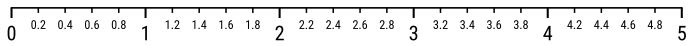
```
im = ax.imshow(Z)

cb = plt.colorbar(im,
        fraction=0.046, pad=0.04)
cb.set_ticks([])
```



Taking advantage of typography

You can use a condensed font such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname("Roboto Condensed")
```



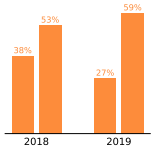
Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

Hatching

You can achieve a nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/" )
```



Read the documentation

Matplotlib comes with an extensive documentation explaining the details of each command and is generally accompanied by examples. Together with the huge online gallery, this documentation is a gold-mine.

6.2. R language

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.

help.search('weighted mean')
Search the help files for a word or phrase.

help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.

class(iris)
Find the class an object belongs to.

Using Packages

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	`2 4 6`	Join elements into a vector
`2:6`	`2 3 4 5 6`	An integer sequence
`seq(2, 3, by=0.5)`	`2.0 2.5 3.0`	A complex sequence
`rep(1:2, times=3)`	`1 2 1 2 1 2`	Repeat a vector
`rep(1:2, each=3)`	`1 1 1 2 2 2`	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.

rev(x)
Return x reversed.

table(x)
See counts of values.

unique(x)
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** package.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x`

`df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

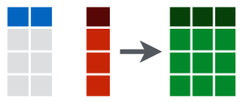
`df[2, 2]`

`nrow(df)`
Number of rows.

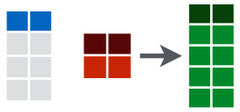
`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Dates

See the **lubridate** package.

Data Import

with readr, tibble, and tidyr

Cheat Sheet

RStudio

R's **tidyverse** is built around **tidy data** stored in **tibbles**, an enhanced version of a data frame.

readr

The front side of this sheet shows how to read text files into R with **readr**.

tidyr

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

Other types of data

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Write functions

Save **x**, an R object, to **path**, a file path, with:

write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**
Tibble/df to comma delimited file.

write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**
Tibble/df to file with any delimiter.

write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**
Tibble/df to a CSV for excel

write_file(x, path, append = FALSE**)**
String to file.

write_lines(x, path, na = "NA", append = FALSE**)**
String vector to file, one element per line.

write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**
Object to RDS file.

write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**
Tibble/df to tab delimited files.

Read functions

Read tabular data to tibbles

These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive()**)**

```
a,b,c
1,2,3
4,5,NA
```

A	B	C
1	2	3
4	5	NA

read_csv()
Reads comma delimited files.
read_csv("file.csv")

```
a;b;c
1;2;3
4;5;NA
```

A	B	C
1	2	3
4	5	NA

read_csv2()
Reads Semi-colon delimited files.
read_csv2("file2.csv")

```
a|b|c
1|2|3
4|5|NA
```

A	B	C
1	2	3
4	5	NA

read_delim(delim, quote = "\"", escape_backslash = FALSE, escape_double = TRUE**)** Reads files with any delimiter.
read_delim("file.txt", delim = "|")

```
a b c
1 2 3
4 5 NA
```

A	B	C
1	2	3
4	5	NA

read_fwf(col_positions**)**
Reads fixed width files.
read_fwf("file.fwf", col_positions = c(1, 3, 5))

read_tsv()
Reads tab delimited files. Also **read_table().**
read_tsv("file.tsv")

Useful arguments

```
a,b,c
1,2,3
4,5,NA
```

Example file
write_csv (path = "file.csv",
x = read_csv("a,b,c\n1,2,3\n4,5,NA"))

A	B	C
1	2	3
4	5	NA

No header
read_csv("file.csv",
***col_names = FALSE**)*

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv("file.csv",
***col_names = c("x", "y", "z")**)*

1	2	3
4	5	NA

Skip lines
read_csv("file.csv",
***skip = 1**)*

A	B	C
1	2	3

Read in a subset
read_csv("file.csv",
***n_max = 1**)*

A	B	C
1	2	3
NA	NA	NA

Missing Values
read_csv("file.csv",
***na = c("4", "5", ".")**)*

Read non-tabular data

read_file(file, locale = default_locale()**)**
Read a file into a single string.

read_file_raw(file**)**
Read a file into a raw vector.

read_lines(file, skip = 0, n_max = -1L, locale = default_locale(), na = character(), progress = interactive()**)**
Read each line into its own string.

read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**
Read each line into a raw vector.

read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**
Apache style log files.

Parsing data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing
- **col_guess()** - the default
- **col_character()**
- **col_double()**
- **col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**)** and **col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number()**
- **col_numeric()**
- **col_skip()**

```
x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()
))
```

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess(**x, na = c("", "NA"), locale = default_locale()**)**
- **parse_character(**x, na = c("", "NA"), locale = default_locale()**)**
- **parse_datetime(**x, format = "", na = c("", "NA"), locale = default_locale()**)** Also **parse_date()** and **parse_time()**
- **parse_double(**x, na = c("", "NA"), locale = default_locale()**)**
- **parse_factor(**x, levels, ordered = FALSE, na = c("", "NA"), locale = default_locale()**)**
- **parse_integer(**x, na = c("", "NA"), locale = default_locale()**)**
- **parse_logical(**x, na = c("", "NA"), locale = default_locale()**)**
- **parse_number(**x, na = c("", "NA"), locale = default_locale()**)**

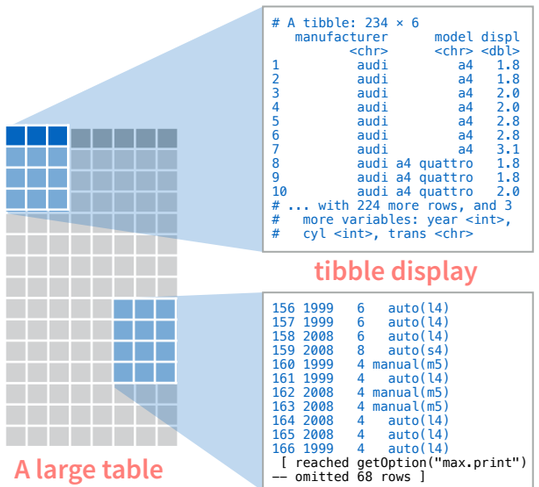
x$A <- parse_number(x$A)

Learn more at **browseVignettes(package = c("readr", "tibble", "tidyr"))** • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen.
- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting




```
# A tibble: 234 × 6
   manufacturer      model displ
          <chr>      <chr> <dbl>
1          audi         a4   1.8
2          audi         a4   1.8
3          audi         a4   2.0
4          audi         a4   2.0
5          audi         a4   2.8
6          audi         a4   2.8
7          audi         a4   3.1
8          audi a4 quattro   1.8
9          audi a4 quattro   1.8
10         audi a4 quattro   2.0
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

```
156 1999   6   auto(l4)
157 1999   6   auto(l4)
158 2008   6   auto(l4)
159 2008   8   auto(s4)
160 1999   4 manual(m5)
161 1999   4   auto(l4)
162 2008   4 manual(m5)
163 2008   4 manual(m5)
164 2008   4   auto(l4)
165 2008   4   auto(l4)
166 1999   4   auto(l4)
 [ reached getOption("max.print")
-- omitted 68 rows ]
```

- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View entire data set with **View(**x, title**)** or **glimpse(**x, width = NULL, …**)**
- Revert to data frame with **as.data.frame()** (required for some older packages)

Construct a tibble in two ways

tibble(…**)**
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…**)**
Construct by rows.
tribble(
~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <dbl>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …**)** Convert data frame to tibble.

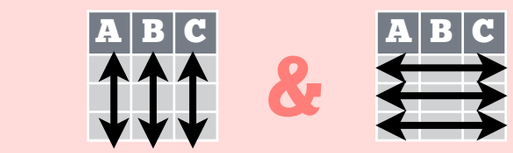
enframe(x, name = "name", value = "value"**)**
Converts named vector to a tibble with a names column and a values column.

is_tibble(x**)** Test whether x is a tibble.

Tidy Data with tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

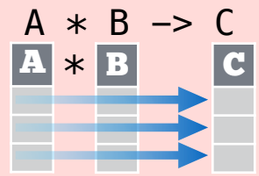


Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:

Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout. Each uses the idea of a key column: value column pair.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

Gather moves column names into a key column, gathering the column values into a single value column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

Spread moves the unique values of a key column into the column names, spreading the values of a value column across the new columns that result.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

key value

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...**)**
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up")**)**
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...**)**
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x,list(x2 = 2), x2)

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...**)**
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split and Combine Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate_rows(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

Data Wrangling
with dplyr and tidyr
Cheat Sheet

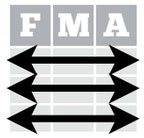
RStudio

Tidy Data - A foundation for wrangling in R

In a tidy data set:

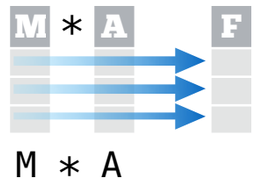
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

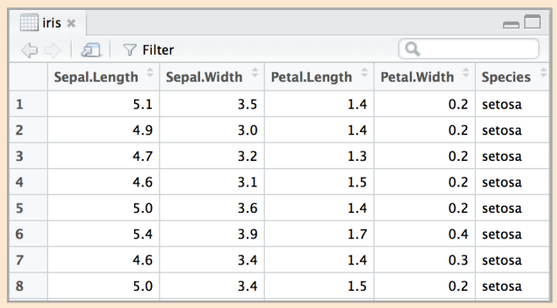
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

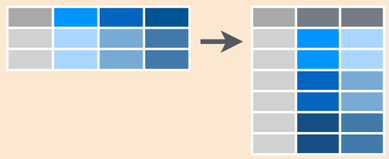
`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

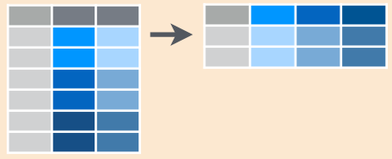
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Reshaping Data - Change the layout of a data set



tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))

Select columns whose name contains a character string.

select(iris, ends_with("Length"))

Select columns whose name ends with a character string.

select(iris, everything())

Select every column.

select(iris, matches(".t."))

Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))

Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))

Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))

Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)

Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first**
First value of a vector.

dplyr::**last**
Last value of a vector.

dplyr::**nth**
Nth value of a vector.

dplyr::**n**
of values in a vector.

dplyr::**n_distinct**
of distinct values in a vector.

IQR
IQR of a vector.

min
Minimum value in a vector.

max
Maximum value in a vector.

mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

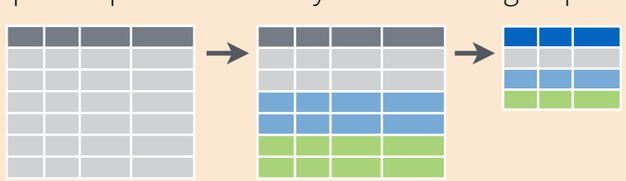
sd
Standard deviation of a vector.

Group Data

dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)
Compute separate summary row for each group.



Make New Variables

dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.

dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead**
Copy with values shifted by 1.

dplyr::**lag**
Copy with values lagged by 1.

dplyr::**dense_rank**
Ranks with no gaps.

dplyr::**min_rank**
Ranks. Ties get min rank.

dplyr::**percent_rank**
Ranks rescaled to [0, 1].

dplyr::**row_number**
Ranks. Ties got to first value.

dplyr::**ntile**
Bin vector into n buckets.

dplyr::**between**
Are values between a and b?

dplyr::**cume_dist**
Cumulative distribution.

dplyr::**cumall**
Cumulative `all`

dplyr::**cumany**
Cumulative `any`

dplyr::**cummean**
Cumulative `mean`

cumsum
Cumulative `sum`

cummax
Cumulative `max`

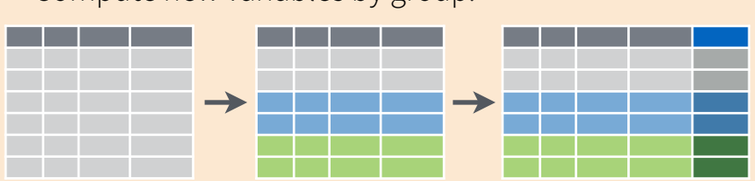
cummin
Cumulative `min`

cumprod
Cumulative `prod`

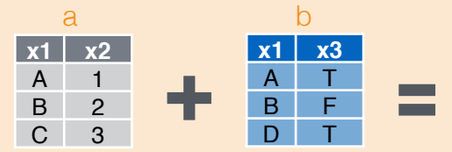
pmax
Element-wise `max`

pmin
Element-wise `min`

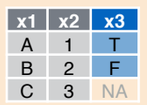
iris %>% group_by(Species) %>% mutate(…)
Compute new variables by group.



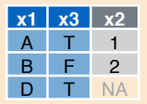
Combine Data Sets



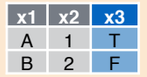
Mutating Joins



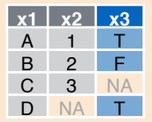
dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.



dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

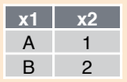


dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

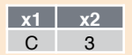


dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

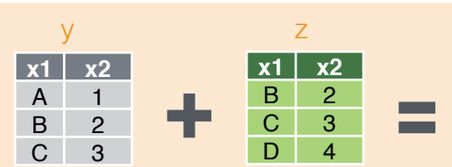
Filtering Joins



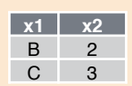
dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.



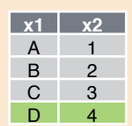
dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.



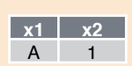
Set Operations



dplyr::**intersect(y, z)**
Rows that appear in both y and z.

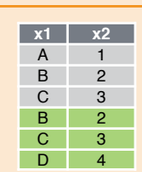


dplyr::**union(y, z)**
Rows that appear in either or both y and z.

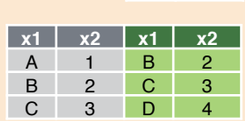


dplyr::**setdiff(y, z)**
Rows that appear in y but not z.

Binding



dplyr::**bind_rows(y, z)**
Append z to y as new rows.



dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

devtools::**install_github("rstudio/EDAWR")** for data sets

Learn more with **browseVignettes(package = c("dplyr", "tidyr"))** • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

R For Data Science *Cheat Sheet*

data.table

Learn R for data science Interactively at www.DataCamp.com

data.table

data.table is an R package that provides a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.

Load the package:

```
> library(data.table)
```

Creating A `data.table`

```
> set.seed(45L)
> DT <- data.table(V1=c(1L,2L),
                   V2=LETTERS[1:3],
                   V3=round(rnorm(4),4),
                   V4=1:12)
```

Create a `data.table` and call it `DT`

Subsetting Rows Using `i`

```
> DT[3:5,]
> DT[3:5]
> DT[V2=="A"]
> DT[V2 %in% c("A","C")]
```

Select 3rd to 5th row
Select 3rd to 5th row
Select all rows that have value `A` in column `V2`
Select all rows that have value `A` or `C` in column `V2`

Manipulating on Columns in `j`

```
> DT[,V2]
 [1] "A" "B" "C" "A" "B" "C" ...
> DT[,.(V2,V3)]
> DT[,sum(V1)]
 [1] 18
> DT[,.(sum(V1),sd(V3))]
   V1        V2
1: 18 0.4546055
> DT[,.(Aggregate=sum(V1),
        Sd.V3=sd(V3))]
   Aggregate      Sd.V3
1:        18 0.4546055
> DT[,.(V1,Sd.V3=sd(V3))]

> DT[,.(print(V2),
        plot(V3),
        NULL)]
```

Return `V2` as a vector

Return `V2` and `V3` as a `data.table`
Return the sum of all elements of `V1` in a vector
Return the sum of all elements of `V1` and the std. dev. of `V3` in a `data.table`

The same as the above, with new names

Select column `V2` and compute std. dev. of `V3`, which returns a single value and gets recycled
Print column `V2` and plot `V3`

Doing `j` by Group

```
> DT[,.(V4.Sum=sum(V4)),by=V1]
   V1 V4.Sum
1:  1     36
2:  2     42
> DT[,.(V4.Sum=sum(V4)),
        by=.(V1,V2)]
> DT[,.(V4.Sum=sum(V4)),
        by=sign(V1-1)]
   sign V4.Sum
1:    0     36
2:    1     42
> DT[,.(V4.Sum=sum(V4)),
        by=.(V1.01=sign(V1-1))]
> DT[1:5,.(V4.Sum=sum(V4)),
         by=V1]
> DT[,.N,by=V1]
```

Calculate sum of `V4` for every group in `V1`

Calculate sum of `V4` for every group in `V1` and `V2`
Calculate sum of `V4` for every group in `sign(V1-1)`

The same as the above, with new name for the variable you're grouping by
Calculate sum of `V4` for every group in `V1` after subsetting on the first 5 rows
Count number of rows for every group in `V1`

General form: `DT[i, j, by]`

"Take `DT`, subset rows using `i`, then calculate `j` grouped by `by`"

Adding/Updating Columns By Reference in `j` Using `:=`

```
> DT[,V1:=round(exp(V1),2)]
> DT
      V1 V2      V3 V4
1: 2.72  A -0.1107  1
2: 7.39  B -0.1427  2
3: 2.72  C -1.8893  3
4: 7.39  A -0.3571  4
...
> DT[,c("V1","V2"):=list(round(exp(V1),2),
                         LETTERS[4:6])]
> DT[,':='(V1=round(exp(V1),2),
           V2=LETTERS[4:6])][]
      V1 V2      V3 V4
1:   15.18  D -0.1107  1
2: 1619.71  E -0.1427  2
3:   15.18  F -1.8893  3
4: 1619.71  D -0.3571  4
> DT[,V1:=NULL]
> DT[,c("V1","V2"):=NULL]
> Cols.chosen=c("A","B")
> DT[,Cols.Chosen:=NULL]

> DT[,(Cols.Chosen):=NULL]
```

`V1` is updated by what is after `:=`
Return the result by calling `DT`

Columns `V1` and `V2` are updated by what is after `:=`
Alternative to the above one. With `[]`, you print the result to the screen

Remove `V1`
Remove columns `V1` and `V2`

Delete the column with column name `Cols.chosen`
Delete the columns specified in the variable `Cols.chosen`

Indexing And Keys

```
> setkey(DT,V2)
> DT["A"]
   V1 V2      V3 V4
1:  1  A -0.2392  1
2:  2  A -1.6148  4
3:  1  A  1.0498  7
4:  2  A  0.3262 10
> DT[c("A","C")]
> DT["A",mult="first"]

> DT["A",mult="last"]

> DT[c("A","D")]
   V1 V2      V3 V4
1:  1  A -0.2392  1
2:  2  A -1.6148  4
3:  1  A  1.0498  7
4:  2  A  0.3262 10
5: NA  D      NA NA
> DT[c("A","D"),nomatch=0]
   V1 V2      V3 V4
1:  1  A -0.2392  1
2:  2  A -1.6148  4
3:  1  A  1.0498  7
4:  2  A  0.3262 10
> DT[c("A","C"),sum(V4)]

> DT[c("A","C"),
     sum(V4),
     by=.EACHI]
   V2 V1
1:  A 22
2:  C 30
> setkey(DT,V1,V2)
> DT[.(2,"C")]
   V1 V2      V3 V4
1:  2  C  0.3262  6
2:  2  C -1.6148 12
> DT[.(2,c("A","C"))]
   V1 V2      V3 V4
1:  2  A -1.6148  4
2:  2  A  0.3262 10
3:  2  C  0.3262  6
4:  2  C -1.6148 12
```

A key is set on `V2`; output is returned invisibly
Return all rows where the key column (set to `V2`) has the value `A`

Return all rows where the key column (`V2`) has value `A` or `C`
Return first row of all rows that match value `A` in key column `V2`
Return last row of all rows that match value `A` in key column `V2`
Return all rows where key column `V2` has value `A` or `D`

Return all rows where key column `V2` has value `A` or `D`

Return total sum of `V4`, for rows of key column `V2` that have values `A` or `C`
Return sum of column `V4` for rows of `V2` that have value `A`, and anohter sum for rows of `V2` that have value `C`

Sort by `V1` and then by `V2` within each group of `V1` (invisible)
Select rows that have value 2 for the first key (`V1`) and the value `C` for the second key (`V2`)

Select rows that have value 2 for the first key (`V1`) and within those rows the value `A` or `C` for the second key (`V2`)

Advanced Data Table Operations

```
> DT[.N-1]
> DT[,.N]
> DT[,.(V2,V3)]
> DT[,list(V2,V3)]
> DT[,mean(V3),by=.(V1,V2)]
   V1 V2      V1
1:  1  A  0.4053
2:  1  B  0.4053
3:  1  C  0.4053
4:  2  A -0.6443
5:  2  B -0.6443
6:  2  C -0.6443
```

Return the penultimate row of the `DT`
Return the number of rows
Return `V2` and `V3` as a `data.table`
Return `V2` and `V3` as a `data.table`
Return the result of `j`, grouped by all possible combinations of groups specified in `by`

`.SD` & `.SDcols`

```
> DT[,print(.SD),by=V2]
> DT[,.SD[c(1,.N)],by=V2]
> DT[,lapply(.SD,sum),by=V2]

> DT[,lapply(.SD,sum),by=V2,
     .SDcols=c("V3","V4")]
   V2      V3 V4
1:  A -0.478 22
2:  B -0.478 26
3:  C -0.478 30
> DT[,lapply(.SD,sum),by=V2,
     .SDcols=paste0("V",3:4)]
```

Look at what `.SD` contains
Select the first and last row grouped by `V2`
Calculate sum of columns in `.SD` grouped by `V2`

Calculate sum of `V3` and `V4` in `.SD` grouped by `V2`

Calculate sum of `V3` and `V4` in `.SD` grouped by `V2`

Chaining

```
> DT <- DT[,.(V4.Sum=sum(V4)),
           by=V1]
   V1 V4.Sum
1:  1     36
2:  2     42
> DT[V4.Sum>40]
> DT[,.(V4.Sum=sum(V4)),
     by=V1][V4.Sum>40]
   V1 V4.Sum
1:  2     42
> DT[,.(V4.Sum=sum(V4)),
     by=V1][order(-V1)]
   V1 V4.Sum
1:  2     42
2:  1     36
```

Calculate sum of `V4`, grouped by `V1`

Select that group of which the sum is >40
Select that group of which the sum is >40 (chaining)

Calculate sum of `V4`, grouped by `V1`, ordered on `V1`

`set()`-Family

`set()`

Syntax: `for (i in from:to) set(DT, row, column, new value)`

```
> rows <- list(3:4,5:6)
> cols <- 1:2
> for(i in seq_along(rows))
    {set(DT,
         i=rows[[i]],
         j=cols[i],
         value=NA)}
```

Sequence along the values of `rows`, and for the values of `cols`, set the values of those elements equal to `NA` (invisible)

`setnames()`

Syntax: `setnames(DT,"old","new")[]`

```
> setnames(DT,"V2","Rating")
> setnames(DT,
           c("V2","V3"),
           c("V2.rating","V3.DC"))
```

Set name of `V2` to `Rating` (invisible)
Change 2 column names (invisible)

`setnames()`

Syntax: `setcolorder(DT,"neworder")`

```
> setcolorder(DT,
              c("V2","V1","V4","V3"))
```

Change column ordering to contents of the specified vector (invisible)

Data Transformation with data.table : : CHEAT SHEET

Basics

data.table is an extremely fast and memory efficient package for transforming data in R. It works by converting R's native data frame objects into data.tables with new and enhanced functionality. The basics of working with data.tables are:

dt[i, j, by]

Take data.table **dt**,
subset rows using **i**
and manipulate columns with **j**,
grouped according to **by**.

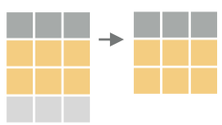
data.tables are also data frames – functions that work with data frames therefore also work with data.tables.

Create a data.table

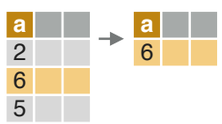
data.table(a = c(1, 2), b = c("a", "b")**)** – create a data.table from scratch. Analogous to data.frame().

setDT(df**)*** or **as.data.table(**df**)** – convert a data frame or a list to a data.table.

Subset rows using i



dt[**1:2**,] – subset rows based on row numbers.



dt[**a > 5**,] – subset rows based on values in one or more columns.

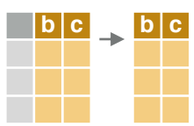
LOGICAL OPERATORS TO USE IN i

<	<=	is.na()	%in%	|	**%like%**
>	>=	!is.na()	!	&	**%between%**

Manipulate columns with j

EXTRACT

dt[, **c(2)**] – extract columns by number. Prefix column numbers with "-" to drop.



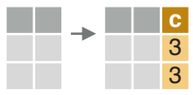
dt[, **.(b, c)**] – extract columns by name.

SUMMARIZE

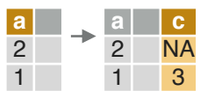
dt[, **.(x = sum(a))**] – create a data.table with new columns based on the summarized values of rows.

Summary functions like mean(), median(), min(), max(), etc. can be used to summarize rows.

COMPUTE COLUMNS*



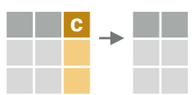
dt[, **c := 1 + 2**] – compute a column based on an expression.



dt[**a == 1**, **c := 1 + 2**] – compute a column based on an expression but only for a subset of rows.

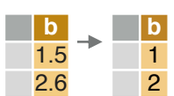
dt[, **`:=`(c = 1 , d = 2)**] – compute multiple columns based on separate expressions.

DELETE COLUMN



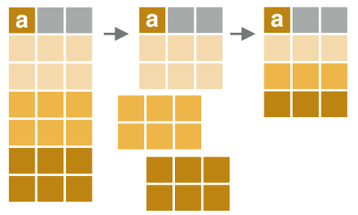
dt[, **c := NULL**] – delete a column.

CONVERT COLUMN TYPE



dt[, **b := as.integer(b)**] – convert the type of a column using as.integer(), as.numeric(), as.character(), as.Date(), etc..

Group according to by



dt[, j, **by = .(a)**] – group rows by values in specified columns.

dt[, j, **keyby = .(a)**] – group *and simultaneously sort* rows by values in specified columns.

COMMON GROUPED OPERATIONS

dt[, **.(c = sum(b))**, **by = a**] – summarize rows within groups.

dt[, **c := sum(b)**, **by = a**] – create a new column and compute rows within groups.

dt[, **.SD[1]**, **by = a**] – extract first row of groups.

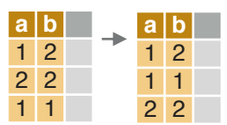
dt[, **.SD[.N]**, **by = a**] – extract last row of groups.

Chaining

dt[…**][**…**]** – perform a sequence of data.table operations by *chaining* multiple "[]".

Functions for data.tables

REORDER

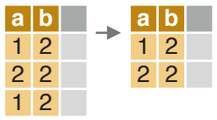


setorder(dt, a, **-**b**)** – reorder a data.table according to specified columns. Prefix column names with "**-**" for descending order.

* SET FUNCTIONS AND :=

data.table's functions prefixed with "set" and the operator ":=" work without "<-" to alter data without making copies in memory. E.g., the more efficient "setDT(df)" is analogous to "df <- as.data.table(df)".

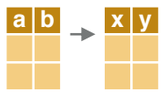
UNIQUE ROWS



unique(dt, by = c("a", "b")**)** – extract unique rows based on columns specified in "by". Leave out "by" to use all columns.

uniqueN(dt, by = c("a", "b")**)** – count the number of unique rows based on columns specified in "by".

RENAME COLUMNS



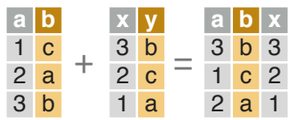
setnames(dt, c("a", "b"), c("x", "y")**)** – rename columns.

SET KEYS

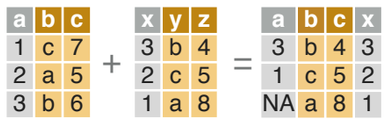
setkey(dt, a, b**)** – set keys to enable fast repeated lookup in specified columns using "dt[.(value),]" or for merging without specifying merging columns using "dt_a[dt_b]".

Combine data.tables

JOIN



dt_a[dt_b, **on = .(b = y)**] – join data.tables on rows with equal values.



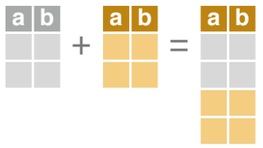
dt_a[dt_b, **on = .(b = y, c > z)**] – join data.tables on rows with equal *and unequal* values.

ROLLING JOIN

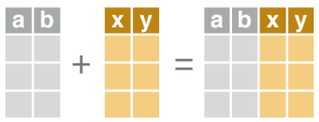
a	id	date		b	id	date		a	id	date	b
1	A	01-01-2010	+	1	A	01-01-2013	=	2	A	01-01-2013	1
2	A	01-01-2012		1	B	01-01-2013		2	B	01-01-2013	1
3	A	01-01-2014									
1	B	01-01-2010									
2	B	01-01-2012									

dt_a[dt_b, **on = .(id = id, date = date), roll = TRUE**] – join data.tables on matching rows in id columns but only keep the most recent preceding match with the left data.table according to date columns. "roll = -Inf" reverses direction.

BIND



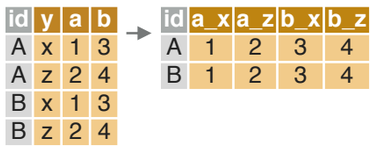
rbind(dt_a, dt_b**)** – combine rows of two data.tables.



cbind(dt_a, dt_b**)** – combine columns of two data.tables.

Reshape a data.table

RESHAPE TO WIDE FORMAT

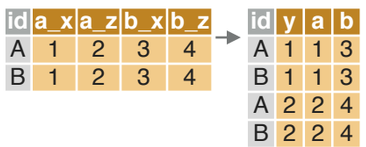


dcast(dt,
id ~ y,
value.var = c("a", "b")**)**

Reshape a data.table from long to wide format.

dt	A data.table.
id ~ y	Formula with a LHS: ID columns containing IDs for multiple entries. And a RHS: columns with values to spread in column headers.
value.var	Columns containing values to fill into cells.

RESHAPE TO LONG FORMAT



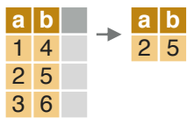
melt(dt,
id.vars = c("id"),
measure.vars = patterns("^a", "^b"),
variable.name = "y",
value.name = c("a", "b")**)**

Reshape a data.table from wide to long format.

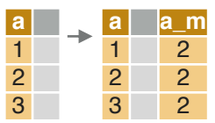
dt	A data.table.
id.vars	ID columns with IDs for multiple entries.
measure.vars	Columns containing values to fill into cells (often in pattern form).
variable.name, value.name	Names of new columns for variables and values derived from old headers.

Apply function to cols.

APPLY A FUNCTION TO MULTIPLE COLUMNS



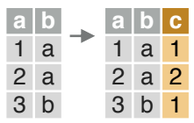
dt[, **lapply(.SD, mean)**, **.SDcols = c("a", "b")**] – apply a function – e.g. mean(), as.character(), which.max() – to columns specified in .SDcols with lapply() and the .SD symbol. Also works with groups.



cols <- c("a")
dt[, **paste0(cols, "_m") := lapply(.SD, mean)**, **.SDcols = cols**] – apply a function to specified columns and assign the result with suffixed variable names to the original data.

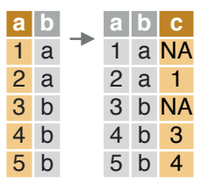
Sequential rows

ROW IDS



dt[, **c := 1:.N**, **by = b**] – within groups, compute a column with sequential row IDs.

LAG & LEAD



dt[, **c := shift(a, 1)**, **by = b**] – within groups, duplicate a column with rows *lagged* by specified amount.

dt[, **c := shift(a, 1, type = "lead")**, **by = b**] – within groups, duplicate a column with rows *leading* by specified amount.

read & write files

IMPORT

fread("file.csv"**)** – read data from a flat file such as .csv or .tsv into R.

fread("file.csv", select = c("a", "b")**)** – read specified columns from a flat file into R.

EXPORT

fwrite(dt, "file.csv"**)** – write data to a flat file from R.

Machine Learning Modelling in R : : CHEAT SHEET

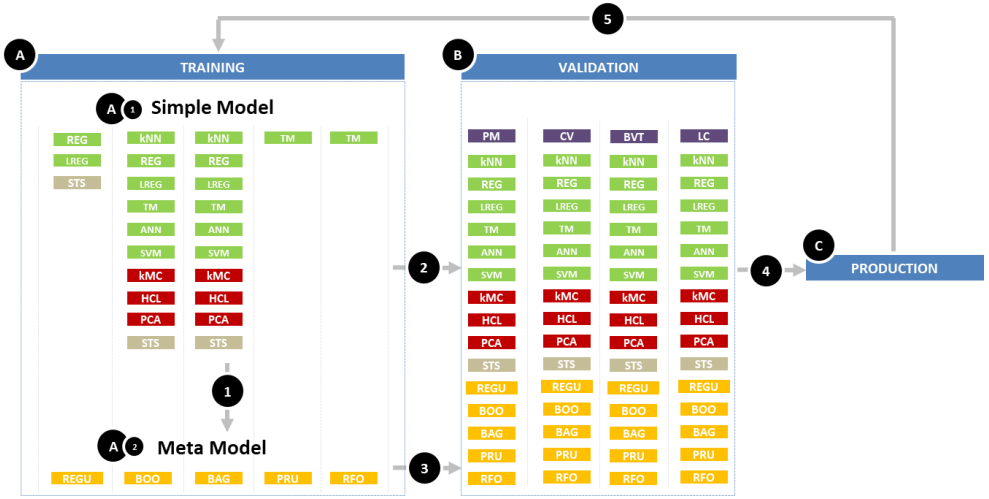
Supervised & Unsupervised Learning

	ALGORITHM	DESCRIPTION	R PACKAGE::FUNCTION	SAMPLE CODE
SUPERVISED LEARNING	NBC Naïve Bayes classifier	A classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature	e1071::naiveBayes	naiveBayes(class ~ ., data = x)
	kNN k–Nearest Neighbours	A non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression	class::knn	knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
	REG Linear Regression	Model the linear relationship between a scalar dependant variable *Y* and one or more explanatory variables (or independent variables) denoted *X*	stats::lm	lm(dist ~ speed, data=cars)
	LREG Logistic Regression	Used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.	stats::glm	glm(Y ~ ., family = binomial (link = 'logit'), data = X)
	TM Tree-Based Models	The idea is to consecutively divide (branch) the training dataset based on the input features until an assignment criterion with respect to the target variable into a "data bucket" (leaf) is reached	rpart::rpart	rpart(Kyphosis ~ Age + Number + Start, data = kyphosis)
	ANN Artificial Neural Network	Neural networks are built from units called perceptrons. Perceptrons have one or more inputs, an activation function and an output. An ANN model is built up by combining perceptrons in structured layers.	neuralnet::neuralnet	neuralnet(f,data=train_,hidden=c(5,3),linear.output=T)
	SVM Support Vector Machine	A data classification method that separates data using hyperplanes	e1071::svm	svm(formula, data = NULL, ..., subset, na.action = na.omit, scale = TRUE)
UNSUPERVISED LEARNING	PCA Principal Component Analysis	A procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.	stats::prcomp stats::princomp FactoMineR::PCA ade4::dudi.pca amap::acp	**stats** : prcomp(formula, data = NULL, subset, na.action, ...) **stats** : princomp(formula, data = NULL, subset, na.action, ...) **FactoMineR** : PCA(decathlon, quanti.sup = 11:12, quali.sup=13) **ade4 :** dudi.pca(deug$tab, center = deug$cent, scale = FALSE, scan = FALSE) **amap :** acp(lubisch)
	KMC k-Mean Clustering	Aims at partitioning *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean	stats::kmeans	kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"), trace=FALSE)
	HCL Hierarchical Clustering	An approach which builds a hierarchy from the bottom-up, and doesn't require the number of clusters to be specified beforehand.	stats::hclust	hclust(d, method = "complete", members = NULL)

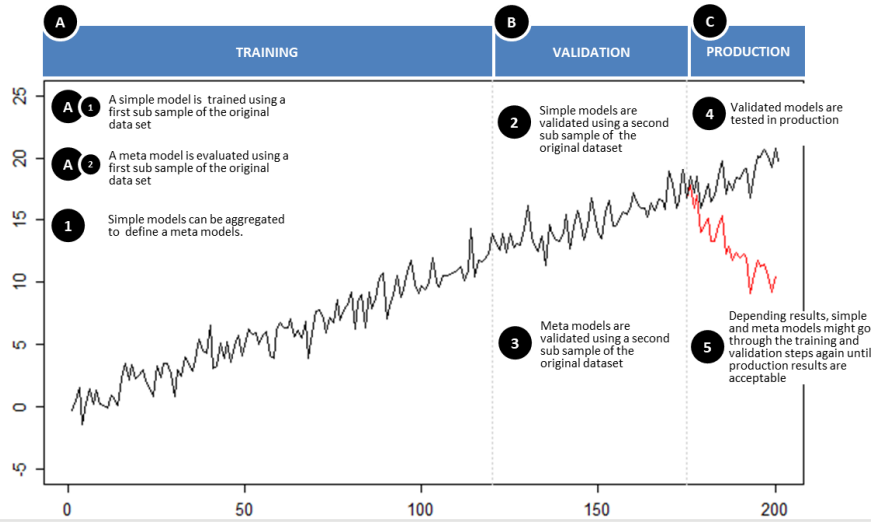
Meta-Algorithm, Time Series & Model Validation

	ALGORITHM	DESCRIPTION	R PACKAGE::FUNCTION	SAMPLE CODE
META ALGORITHM	REGU Regularisation L1 (Lasso) L2 (Ridge)	Regularization adds a penalty on the different parameters of a model to reduce the freedom of the model. Hence, the model will be less likely to fit the noise of the training data and will improve the generalization abilities of the model	glmnet::glmnet	**L1 :** glmnet(myMatrixA, myMatrixB , family = "gaussian", alpha = 1) **L2 :** glmnet(myMatrixA, myMatrixB , family = "gaussian", alpha = 0)
	BOO Boosting	A process of iteratively refining, e.g. by reweighting, of estimated regression and classification functions (though it has primarily been applied to the latter), in order to improve predictive ability.	**Parametric model** - mboost::glmboost	glmboost(Yen ~ ., data = curr1[trnidxs,])
	BAG Bagging	Bagging is a way to increase the power of a predictive statistical model by taking multiple random samples (with replacement) of the training data set, and using each of them to construct a separate model and separate predictions for the original test set	**All models**: foreach **Tree models**: ipred::bagging	**foreach :** d <- data.frame(x=1:10, y=rnorm(10)) s <- foreach(d=iter(d, by='row'), .combine=rbind) %dopar%d identical(s, d) **ipred :** bagging(formula, data, subset, na.action=na.rpart, \dots)
	PRU Pruning	Pruning is a technique that reduces the size of decision tree by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier and hence improves predictive accuracy by reducing overfitting	rpart::prune	prune(x, cp = 0.1)
	RFO Random Forrest	An ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression)	randomForest::randomForest	randomForest(X ~ . , data = Y , subset = mySub)
TIME SERIES	STS Lead-lag analysis, Auto-correlation, Spectral analysis, Time series clustering, Seasonality, Trend....	Random sampling of observations for training and testing a model can be an issue when faced with a times dimension. Random sampling may either destroy serial correlation properties in the data which we would like to exploit	stats xts forecast spectral TTR	**Auto-correlation**: acf(x, lag.max = NULL, type = c("correlation", "covariance", "partial")) **Spectral Analysis**: spec.pgram(myTs, spans = NULL) **Seasonal Decomposition of Time Series** - stl(x, s.window = 7, t.window = 50, t.jump = 1) .....
MODEL VALIDATION	PM Performance metrics	Depends on the problem: • **Regression**: squared errors, outliers, error rate... • **Classification**: Accuracy, precision, recall, F-score...	**Regression**-stats::outlierTest, stats:: qqPlot .... **Classification**-ROCR:: **Tree**: caret:: confusionMatrix	**Regresion**: fit <- lm(Y~X,data=myData) outlierTest(fit) qqPlot(fit, main="QQ Plot")
	BVT Biais-Variance Tradeoff	• Simple models with few parameters are easier to compute but may lead to poorer fits (**high bias**). • Complex models may provide more accurate fits but may over-fit the data (**high variance**)	Tailored to the analysis	Tailored to the analysis
	CV Cross validation	Cross validation compares the test performances of different model realisations with different sets or values of parameters	caret::createDataPartition caret::createFolds	createDataPartition(classes, p = 0.8, list = FALSE)
	LC Learning Curves	Learning curves plot a model's training and test errors, or the chosen performance metric, depending on the training set size	caret:: learing_curve_dat	learing_curve_dat(dat, outcome = NULL, proportion = (1:10)/10, test_prop = 0, verbose = TRUE, ...)

Standard Modelling Workflow



Time Series View



Machine Learning with R

mlr

Introduction

***mlr* offers a unified interface for the basic building blocks of machine learning: tasks, learners, hyperparameters, etc.**

Tasks contain a description of a task (classification, regression, clustering, etc.) and a data set.

Learners specify a machine learning algorithm (GLM, SVM, xgboost, etc.) and its parameters.

Hyperparameters are learner settings that can be specified directly or tuned. A **parameter set** lists the possible hyperparameters for a given learner.

Wrapped Models are learners that have been trained on a task and can be used to make predictions.

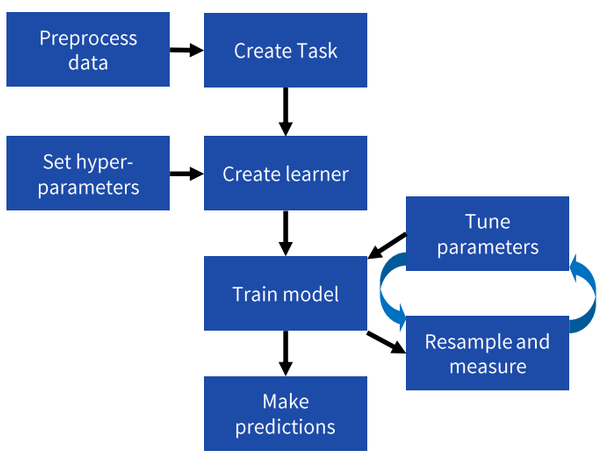
Predictions are the results of applying a model to either new data or the original training data.

Measures control how learner performance is evaluated, e.g. RMSE, LogLoss, AUC, etc.

Resampling estimates generalization performance by separating training data from test data. Common strategies include holdout and cross-validation.

Links: Tutorial | CRAN | Github

mlr workflow



Setup

Preprocessing data

`createDummyFeatures(obj=,target=,method=,cols=)`
Creates (0,1) flags for each non-numeric variable excluding `target`. Can be applied to entire dataset or only specific `cols`

`normalizeFeatures(obj=,target=,method=,cols=, range=,on.constant=)`
Normalizes numerical features according to specified `method`:
- `"center"` (subtract mean)
- `"scale"` (divide by std. deviation)
- `"standardize"` (center and scale)
- `"range"` (linear scale to given range, default `range=c(0,1)`)

`mergeSmallFactorLevels(task=,cols=,min.perc=)`
Combine infrequent factor levels into a single merged level

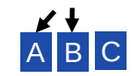
`summarizeColumns(obj=)` where `obj` is a data.frame or task.
Provides type, NA, and distributional data about each column

See also `capLargeValues` `dropFeatures` `removeConstantFeatures` `summarizeLevels`

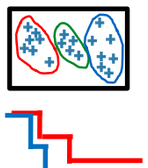
Creating a task

`makeClassifTask(data=,target=)`
Classification of a target variable, with optional positive class `positive`

`makeRegrTask(data=,target=)`
Regression on a target variable



`makeMultilabelTask(data=,target=)`
Classification where the target can belong to more than one class per observation



`makeClusterTask(data=)`
Unsupervised clustering on a data set

`makeSurvTask(data=,target= c("time","event"))`
Survival analysis with a survival time column and an event column

`makeCostSensTask(data=,costs=)`
Cost-sensitive classification where each observation-cost pair has a specified cost

Other arguments that can be passed to a `task`:
- `weights=` Weighting vector to apply to observations
- `blocking=` Factor vector where each level indicates a block of observations that will not be split up in resampling

Making a learner

`makeLearner(cl=,predict.type=,...,par.vals=)`
Choose an algorithm class to perform the task and determine what that algorithm will predict
- `cl=`name of algorithm, e.g. `"classif.xgboost"` `"regr.randomForest"` `"cluster.kmeans"`
- `predict.type="response"` returns a prediction type that matches the source data; `"prob"` returns a predicted probability for classification problems only; `"se"` returns the a standard error of the prediction for regression problems only. Only certain learners can return `"prob"` and `"se"`
- `par.vals=` takes a list of hyperparameters and passes them to the learner; parameters can also be passed directly (`...`)

You can make multiple learners at once with `makeLearners()`

mlr has integrated over 170 different learning algorithms
- Full list: `View(listLearners())` shows all learners
- Available learners for a task: `View(listLearners(task))`
- Filtered list: `View(listLearners("classif", properties=c("prob", "factors")))` shows all classification learners `"classif"` which can predict probabilities `"prob"` and handle factor inputs `"factors"`
- See also `getLearnerProperties()`

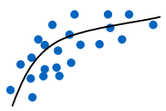
Training & Testing

Setting hyperparameters

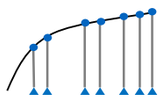
`setHyperPars(learner=,...)`
Set the hyperparameters (settings) for each learner, if you don't want to use the defaults. You can also specify hyperparameters in the `makeLearner()` call

`getParamSet(learner=)`
Show the possible universe of parameters for your learner; can take a learner directly, or a text string such as `"classif.qda"`

Train a model and predict



`train(learner=,task=)`
Train a model (`WrappedModel`) by applying a learner to a task. By default, the model will train on all observations. The underlying model can be extracted with `getLearnerModel()`



`predict(object=,task=,newdata=)`
Use a trained model to make predictions on a task or dataset. The resulting `pred` object can be viewed with `View(pred)` or accessed by `as.data.frame(pred)`

Measuring performance

`performance(pred=,measures=)`
Calculate performance of predictions according to one or more of several measures (use `listMeasures()` for full list):
- classif `acc auc bac ber brier[.scaled] f1 fdr fn fnr fp fpr gmean multiclass[.au1u .aunp .aunu .brier] npv ppv qsr ssr tn tnr tp tpr wkappa`
- regr `arsq expvar kendalltau mae mape medae medse mse msle rae rmse rmsle rrse rsq sae spearmanrho sse`
- cluster `db dunn G1 G2 silhouette`
- multilabel `multilabel[.f1 .subset01 .tpr .ppv .acc .hamloss]`
- costsens `mcp meancosts`
- surv `cindex`
- other `featperc timeboth timepredict timetrain`

For detailed performance data on classification tasks, use:
- `calculateConfusionMatrix(pred=)`
- `calculateROCMeasures(pred=)`

Resampling a learner

`makeResampleDesc(method=,...,stratify=)`
`method` must be one of the following:
- `"CV"` (cross-validation, for number of folds use `iters=`)
- `"LOO"` (leave-one-out cross-validation, for folds use `iters=`)
- `"RepCV"` (repeated cross-validation, for number of repetitions use `reps=`, for folds use `folds=`)
- `"Subsample"` (aka Monte-Carlo cross-validation, for iterations use `iters=`, for train % use `split=`)
- `"Bootstrap"` (out-of-bag bootstrap, uses `iters=`)
- `"Holdout"` (for train % use `split=`)

`stratify` keeps target proportions consistent across samples.

`makeResampleInstance(desc=,task=)` can reduce noise by ensuring the resampling is done identically every time.

`resample(learner=,task=,resampling=,measures=)`
Train and test model according to specified resampling strategy.

mlr includes several pre-specified resample descriptions: `cv2` (2-fold cross-validation), `cv3`, `cv5`, `cv10`, `hout` (holdout with split 2/3 for training, 1/3 for testing).
Convenience functions also exist to `resample()` with a specific strategy: `crossval()`, `repcv()`, `holdout()`, `subsample()`, `bootstrapOOB()`, `bootstrapB632()`, `bootstrapB632plus()`

Refining Performance

Tuning hyperparameters

Set search space using `makeParamSet(make<type>Param())`
- `makeNumericParam(id=,lower=,upper=,trafo=)`
- `makeIntegerParam(id=,lower=,upper=,trafo=)`
- `makeIntegerVectorParam(id=,len=,lower=,upper=, trafo=)`
- `makeDiscreteParam(id=,values=c(...))` (can also be used to test discrete values of numeric or integer parameters)

`trafo` transforms the parameter output using a specified function, e.g. `lower=-2,upper=2,trafo=function(x) 10^x` would test values between 0.01 and 100, scaled exponentially
Other acceptable parameter types include `Logical` `LogicalVector` `CharacterVector` `DiscreteVector`

Set a search algorithm with `makeTuneControl<type>()`
- `Grid(resolution=10L)` Grid of all possible points
- `Random(maxit=100)` Randomly sample search space
- `MBO(budget=)` Use Bayesian model-based optimization
- `Irace(n.instances=)` Iterated racing process
- Other types: `CMAES`, `Design`, `GenSA`

Tune using `tuneParams(learner=,task=,resampling=, measures=,par.set=,control=)`

Quickstart

Prepare data for training and testing

```
library(mlbench)
data(Soybean)
soy = createDummyFeatures(Soybean,target="Class")
tsk = makeClassifTask(data=soy,target="Class")
ho = makeResampleInstance("Holdout",tsk)
tsk.train = subsetTask(tsk,ho$train.inds[[1]])
tsk.test = subsetTask(tsk,ho$test.inds[[1]])
```

Convert the factor inputs in the Soybean dataset into (0,1) dummy features which can be used by the XGboost algorithm. Create a task to precict the "Class" column. Create a train set with 2/3 of data and a test set with the remaining 1/3 (default).

Create learner and evaluate performance

```
lrn = makeLearner("classif.xgboost",nrounds=10)
cv = makeResampleDesc("CV",iters=5)
res = resample(lrn,tsk.train,cv,acc)
```

Create an XGboost learner which will build 10 trees. Then test performance using 5-fold cross-validation. Accuracy should be between 0.90-0.92.

Tune hyperparameters and retrain model

```
ps = makeParamSet(makeNumericParam("eta",0,1),
  makeNumericParam("lambda",0,200),
  makeIntegerParam("max_depth",1,20))
tc = makeTuneControlMBO(budget=100)
tr = tuneParams(lrn,tsk.train,cv5,acc,ps,tc)
lrn = setHyperPars(lrn,par.vals=tr$x)
```

Tune hyperparameters `eta`, `lambda`, and `max_depth` by defining a search space and using Model Based Optimization (MBO) to control the search. Then perform 100 rounds of 5-fold cross-validation, improving accuracy to ~0.93. Update the XGboost learner with the tuned hyperparameters.

```
mdl = train(lrn,tsk.train)
prd = predict(mdl,tsk.test)
calculateConfusionMatrix(prd)
mdl = train(lrn,tsk)
```

Train the model on the train set and make predictions on the test set. Show performance as a confusion matrix. Finally, re-train model on the full set to use on new data. You are now ready to go out into the real world and make 93% accurate predictions!

Legend for functions (not all parameters shown):
`function(required_parameters=,optional_parameters=)`

Configuration

mlr's default settings can be changed using `configureMlr()`:

- `show.info` Whether to show verbose output by default when training, tuning, resampling, etc. (`TRUE`)
- `on.learner.error` How to handle a learner error. `"stop"` halts execution, `"warn"` returns NAs and displays a warning, `"quiet"` returns NAs with no warning (`"stop"`)
- `on.learner.warning` How to handle a learner warning. `"warn"` displays a warning, `"quiet"` supresses it (`"warn"`)
- `on.par.without.desc` How to handle a parameter with no description. `"stop"`, `"warn"`, `"quiet"` (`"stop"`)
- `on.par.out.of.bounds` How to handle a parameter with an out-of-bounds value. `"stop"`, `"warn"`, `"quiet"` (`"stop"`)
- `on.measure.not.applicable` How to handle a measure not applicable to a learner. `"stop"`,`"warn"`,`"quiet"` (`"stop"`)
- `show.learner.output` Whether to show learner output to the console during training (`TRUE`)
- `on.error.dump` Whether to create an error dump for crashed learners if `on.learner.error` is not set to `"stop"` (`TRUE`)

Use `getMlrOptions()` to see current settings

Parallelization

mlr works with the `parallelMap` package to take advantage of multicore and cluster computing for faster operations. mlr automatically detects which operations are able to run in parallel.

To begin parallel operation use:
`parallelStart(mode=,cpus=,level=)`

- `mode` determines how the parallelization is performed:
  - `"local"` no parallelization applied, simply uses `mapply`
  - `"multicore"` multicore execution on a single machine, uses `parallel::mclapply`. Not available in Windows.
  - `"socket"` multicore execution in socket mode
  - `"mpi"` Snow MPI cluster on one or multiple machines using `parallel::makeCluster` and `parallel::clusterMap`
  - `"BatchJobs"` Batch queuing HPC clusters using `BatchJobs::batchMap`
- `cpus` determines how many logical cores will be used
- `level` controls parallelization: `"mlr.benchmark"` `"mlr.resample"` `"mlr.selectFeatures"` `"mlr.tuneParams"` `"mlr.ensemble"`

To end parallelization, use `parallelStop()`

Imputation

`impute(obj=,target=,cols=,dummy.cols=,dummy.type=)` Applies specified logic to data frame or task containing NAs and returns an imputation description which can be used on new data

- `obj=`data frame or task on which to perform imputation
- `target=`specify target variable which will not be imputed
- `cols=`column names and logic for imputation*
- `dummy.cols=`column names to create a NA (T/F) column*
- `dummy.type=`set to `"numeric"` to use (0,1) instead of (T/F)

*Can also use `classes` and `dummy.classes` in place of `cols`

Imputation logic is passed to `cols` or `classes` via a list, e.g.: `cols=list(V1=imputeMean())` where `V1` is the column to which to apply the imputation, and `imputeMean()` is the imputation method. Available imputation methods include:

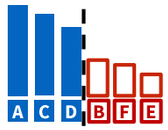
```
imputeConst(const=) imputeMedian() imputeMode()
imputeMin(multiplier=) imputeMax(multiplier=)
imputeNormal(mean=,sd=)
imputeHist(breaks=,use.mids=)
imputeLearner(learner=,features=)
```

`impute` returns a list containing the imputed dataset or task as well as an imputation description that can be used to reapply the same imputation to new data using `reimpute`

`reimpute(obj=,desc=)` Imputes missing values on a task or dataset (`obj`) using a description (`desc`) created by `impute`

Feature Extraction

Feature filtering



`filterFeatures(task=,method=, perc=,abs=,threshold=)` Uses a learner-agnostic feature evaluation method to rank feature importance, then includes only features in the top n percent (`perc=`), top n (`abs=`), or which meet a set performance threshold (`threshold=`).

Outputs a task with features that failed the test omitted. `method` defaults to `"randomForestSRC.rfsrc"`, but can be set to:
`"anova.test"` `"carscore"` `"cforest.importance"` `"chi.squared"` `"gain.ratio"` `"information.gain"` `"kruskal.test"` `"linear.correlation"` `"mrmr"` `"oneR"` `"permutation.importance"` `"randomForest.importance"` `"randomForestSRC.rfsrc"` `"randomForestSRC.var.select"` `"rank.correlation"` `"relief"` `"symmetrical.uncertainty"` `"univariate.model.score"` `"variance"`

Feature selection

`selectFeatures(learner=,task= resampling=,measures=,control=)` Uses a feature selection algorithm (`control`) to resample and build a model repeatedly using different feature sets each time in order to find the best set.

Available controls include:

- `makeFeatSelControlExhaustive(max.features=)` Try every combination of features up to optional `max.features`
- `makeFeatSelControlRandom(maxit=,prob=, max.features=)` Randomly sample features with probability `prob` (default 0.5) until `maxit` (default 100) iterations; return the best one found
- `makeFeatSelControlSequential(method=,maxit=, max.features=,alpha=,beta=)` Perform an iterative search using a `method` from the following: `"sfs"` forward search, `"sbs"` backward search, `"sffs"` floating forward search, `"sfbs"` floating backward search. `alpha` indicates minimum improvement required to add a feature; `beta` indicates minimum required to remove a feature
- `makeFeatSelControlGA(maxit=,max.features=,mu=, lambda=,crossover.rate=,mutation.rate=)` Genetic algorithm trains on random feature vectors, then uses crossover on the best performers to produce 'offspring', repeated over generations. `mu` is size of parent population, `lambda` is size of children population, `crossover.rate` is probability of choosing a bit from first parent, `mutation.rate` is probability of flipping a bit (on or off)

`selectFeatures` returns a `FeatSelResult` object which contains optimal features and an optimization path. To apply feature selection result (`fsr`) to your task (`tsk`), use:
`tsk = subsetTask(tsk,features=fsr$x)`

Benchmarking

`benchmark(learners=,tasks=,resamplings=,measures=)` Allows easy comparison of multiple learners on a single task, a single learner on multiple tasks, or multiple learners on multiple tasks. Returns a benchmark result object.

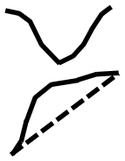
Benchmark results can be accessed with a variety of functions beginning with `getBMR<object>`: `AggrPerformance` `FeatSelResults` `FilteredFeatures` `LearnerIds` `LeanerShortNames` `Learners` `MeasureIds` `Measures` `Models` `Performances` `Predictions` `TaskDescs` `TaskIds` `TuneResults`

mlr contains several toy tasks which are useful for benchmarking:
`agri.task` `bc.task` `bh.task` `costiris.task` `iris.task` `lung.task` `mtcars.task` `pid.task` `sonar.task` `wpbc.task` `yeast.task`

Visualization

Performance

`generateThreshVsPerfData(obj=,measures=)` Measure performance at different probability cutoffs to determine optimal decision threshold for binary classification problems



- `plotThreshVsPerf(obj)` Plot visual representation of threshold curve(s) from `ThreshVsPerfData`
- `plotROCCurves(obj)` Plot receiver operating characteristic (ROC) curve from `ThreshVsPerfData`. Must set `measures=list(fpr,tpr)`

Residuals

- `plotResiduals(obj=)` Plots residuals for `Prediction` or `BenchmarkResult`

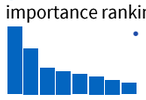
Learning curve

`generateLearningCurveData(learners=,task=, resampling=,percs=,measures=)` Measure performance of learner(s) trained on different percentages of task data

- `plotLearningCurve(obj=)` Plot curve showing learner performance vs. proportion of data used, uses `LearningCurveData`

Feature importance

`generateFilterValuesData(task=,method=)` Get feature importance rankings using specified filter method



- `plotFilterValues(obj=)` Plot bar chart of feature importance based on filter method using `FilterValuesData`

Hyperparameter tuning

`generateHyperParsEffectData(tune.result=)` Get the impact of different hyperparameter settings on model performance

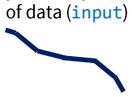
- `plotHyperParsEffect(hyperpars.effect.data=,x=,y=,z=)` Create a plot showing hyperparameter impact on performance using `HyperParsEffectData`

See also:

- `plotOptPath(op=)` Display details of optimization process. Takes `<obj>$opt.path`, where `<obj>` is an object of class `tuneResult` or `featSelResult`
- `plotTuneMultiCritResult(res=)` Show pareto front for results of tuning to multiple performance measures

Partial dependence

`generatePartialDependenceData(obj=,input=)` Get partial dependence of model (`obj`) prediction over each feature of data (`input`)

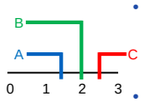


- `plotPartialDependence(obj=)` Plots partial dependence of model using `PartialDependenceData`

Benchmarking

- `plotBMRBoxplots(bmr=)` Distribution of performances
- `plotBMRSummary(bmr=)` Scatterplot of avg. performances
- `plotBMRRanksAsBarChart(bmr=)` Rank learners in bar plot

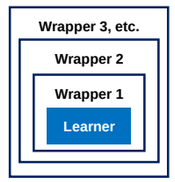
Other



- `generateCritDifferencesData(bmr=, measure=,p.value=,test=)` Perform critical-differences test using either the Bonferroni-Dunn (`"bd"`) or `"Nemenyi"` test
- `plotCritDifferences(obj=)`

- `generateCalibrationData(obj=)` Evaluate calibration of probability predictions vs. true incidence
- `plotCalibration(obj=)`

Wrappers



Wrappers fuse a learner with additional functionality. mlr treats a learner with wrappers as a single learner, and hyperparameters of wrappers can be tuned jointly with underlying model parameters. Models trained with wrappers will apply them to new data.

Preprocessing and imputation

```
makeDummyFeaturesWrapper(learner=)
makeImputeWrapper(learner=,classes=,cols=)
makePreprocWrapper(learner=,train=,predict=)
makePreprocWrapperCaret(learner=,...)
makeRemoveConstantFeaturesWrapper(learner=)
```

Class imbalance

```
makeOverBaggingWrapper(learner=)
makeSMOTEWrapper(learner=)
makeUndersampleWrapper(learner=)
makeWeightedClassesWrapper(learner=)
```

Cost-sensitive learning

```
makeCostSensClassifWrapper(learner=)
makeCostSensRegrWrapper(learner=)
makeCostSensWeightedPairsWrapper(learner=)
```

Multilabel classification

```
makeMultilabelBinaryRelevanceWrapper(learner=)
makeMultilabelClassifierChainsWrapper(learner=)
makeMultilabelDBRWrapper(learner=)
makeMultilabelNestedStackingWrapper(learner=)
makeMultilabelStackingWrapper(learner=)
```

Other

```
makeBaggingWrapper(learner=)
makeConstantClassWrapper(learner=)
makeDownsampleWrapper(learner=,dw.perc=)
makeFeatSelWrapper(learner=,resampling=,control=)
makeFilterWrapper(learner=,fw.perc=,fw.abs=,
fw.threshold=)
makeMultiClassWrapper(learner=)
makeTuneWrapper(learner=,resampling=,par.set=,
control=)
```

Nested Resampling

mlr supports **nested resampling** for complex operations such as tuning and feature selection through wrappers. In order to get a good estimate of generalization performance and avoid data leakage, both an outer (for tuning/feature selection) and an inner (for the base model) resampling process are advised.

- Outer resampling can be specified in `resample` or `benchmark`
- Inner resampling can be specified in `makeTuneWrapper`, `makeFeatSelWrapper`, etc.

Ensembles

`makeStackedLearner(base.learners=,super.learner=, method=)` Combines multiple learners to create an ensemble

- `base.learners=`learners to use for initial predictions
- `super.learner=`learner to use for final prediction
- `method=`how to combine base learner predictions:
  - `"average"` simple average of all base learners
  - `"stack.nocv"`, `"stack.cv"` train super learner on results of base learners, with or without cross-validation
  - `"hill.climb"` search for optimal weighted average
  - `"compress"` with a neural network for faster performance

R For Data Science *Cheat Sheet*

xts

Learn R for data science Interactively at www.DataCamp.com

xts

eXtensible Time Series (xts) is a powerful package that provides an extensible time series class, enabling uniform handling of many R time series classes by extending zoo.

Load the package as follows:

```
> library(xts)
```

xts Objects

xts objects have three main components:

- **coredata**: always a matrix for xts objects, while it could also be a vector for zoo objects
- **index**: vector of any `Date`, `POSIXct`, `chron`, `yearmon`, `yearqtr`, or `DateTime` classes
- **xtsAttributes**: arbitrary attributes

Creating xts Objects

```
> xts1 <- xts(x=1:10, order.by=Sys.Date()-1:10)
> data <- rnorm(5)
> dates <- seq(as.Date("2017-05-01"),length=5,by="days")
> xts2 <- xts(x=data, order.by=dates)
> xts3 <- xts(x=rnorm(10),
              order.by=as.POSIXct(Sys.Date()+1:10),
              born=as.POSIXct("1899-05-08"))
> xts4 <- xts(x=1:10, order.by=Sys.Date()+1:10)
```

Convert To And From xts

```
> data(AirPassengers)
> xts5 <- as.xts(AirPassengers)
```

Import From Files

```
> dat <- read.csv(tmp_file)
> xts(dat, order.by=as.Date(rownames(dat),"%m/%d/%Y"))
> dat_zoo <- read.zoo(tmp_file,
                      index.column=0,
                      sep=",",
                      format="%m/%d/%Y")
> dat_zoo <- read.zoo(tmp,sep=",",FUN=as.yearmon)
> dat_xts <- as.xts(dat_zoo)
```

Export xts Objects

```
> data_xts <- as.xts(matrix)
> tmp <- tempfile()
> write.zoo(data_xts,sep=",",file=tmp)
```

Replace & Update

```
> xts2[dates] <- 0
> xts5["1961"] <- NA
> xts2["2016-05-02"] <- NA
```

Replace values in `xts2` on `dates` with `0`
Replace dates from `1961` with `NA`
Replace the value at 1 specific index with `NA`

Applying Functions

```
> ep1 <- endpoints(xts4,on="weeks",k=2)
[1]  0  5 10
> ep2 <- endpoints(xts5,on="years")
[1]   0  12  24  36  48  60  72  84  96 108 120 132 144
> period.apply(xts5,INDEX=ep2,FUN=mean)
> xts5_yearly <- split(xts5,f="years")
> lapply(xts5_yearly,FUN=mean)
> do.call(rbind,
          lapply(split(xts5,"years"),
          function(w) last(w,n="1 month")))
> do.call(rbind,
          lapply(split(xts5,"years"),
          cumsum))
> rollapply(xts5, 3, sd)
```

Take index values by time
Calculate the yearly mean
Split `xts5` by year
Create a list of yearly means
Find the last observation in each year in `xts5`
Calculate cumulative annual passengers
Apply sd to rolling margins of `xts5`

Selecting, Subsetting & Indexing

Select

```
> mar55 <- xts5["1955-03"]
```

Get value for March 1955

Subset

```
> xts5_1954 <- xts5["1954"]
> xts5_janmarch <- xts5["1954/1954-03"]
> xts5_janmarch <- xts5["/1954-03"]
> xts4[ep1]
```

Get all data from 1954
Extract data from Jan to March '54
Get all data until March '54
Subset `xts4` using `ep2`

`first()` and `last()`

```
> first(xts4,'1 week')
> first(last(xts4,'1 week'),'3 days')
```

Extract first 1 week
Get first 3 days of the last week of data

Indexing

```
> xts2[index(xts3)]
> days <- c("2017-05-03","2017-05-23")
> xts3[days]
> xts2[as.POSIXct(days,tz="UTC")]
> index <- which(.indexwday(xts1)==0|.indexwday(xts1)==6)
> xts1[index]
```

Extract rows with the index of `xts3`
Extract rows using the vector `days`
Extract rows using `days` as POSIXct
Index of weekend days
Extract weekend days of `xts1`

Missing Values

```
> na.omit(xts5)
> xts_last <- na.locf(xts2)

> xts_last <- na.locf(xts2,
                      fromLast=TRUE)
> na.approx(xts2)
```

Omit NA values in `xts5`
Fill missing values in `xts2` using last observation
Fill missing values in `xts2` using next observation
Interpolate NAs using linear approximation

Arithmetic Operations

`coredata()` or `as.numeric()`

```
> xts3 + as.numeric(xts2)
> xts3 * as.numeric(xts4)
> coredata(xts4) - xts3
> coredata(xts4) / xts3
```

Addition
Multiplication
Subtraction
Division

Shifting Index Values

```
> xts5 - lag(xts5)
> diff(xts5,lag=12,differences=1)
```

Period-over-period differences
Lagged differences

Reindexing

```
> xts1 + merge(xts2,index(xts1),fill=0)
                  e1
2017-05-04 5.231538
2017-05-05 5.829257
2017-05-06 4.000000
2017-05-07 3.000000
2017-05-08 2.000000
2017-05-09 1.000000
> xts1 - merge(xts2,index(xts1),fill=na.locf)
                  e1
2017-05-04 5.231538
2017-05-05 5.829257
2017-05-06 4.829257
2017-05-07 3.829257
2017-05-08 2.829257
2017-05-09 1.829257
```

Addition

Subtraction

Merging

```
> merge(xts2,xts1,join='inner')
                 xts2 xts1
2017-05-05 -0.8382068   10

> merge(xts2,xts1,join='left',fill=0)
                 xts2 xts1
2017-05-01  1.7482704    0
2017-05-02 -0.2314678    0
2017-05-03  0.1685517    0
2017-05-04  1.1685649    0
2017-05-05 -0.8382068   10

> rbind(xts1, xts4)
```

Inner join of `xts2` and `xts1`

Left join of `xts2` and `xts1`, fill empty spots with `0`

Combine `xts1` and `xts4` by rows

Inspect Your Data

```
> core_data <- coredata(xts2)
> index(xts1)
```

Extract core data of objects
Extract index of objects

Class Attributes

```
> indexClass(xts2)
> indexClass(convertIndex(xts,'POSIXct'))
> indexTZ(xts5)
> indexFormat(xts5) <- "%Y-%m-%d"
```

Get index class
Replacing index class
Get index class
Change format of time display

Time Zones

```
> tzone(xts1) <- "Asia/Hong_Kong"
> tzone(xts1)
```

Change the time zone
Extract the current time zone

Periods, Periodicity & Timestamps

```
> periodicity(xts5)
> to.yearly(xts5)
> to.monthly(xts3)
> to.quarterly(xts5)
> to.period(xts5,period="quarters")
> to.period(xts5,period="years")
> nmonths(xts5)
> nquarters(xts5)
> nyears(xts5)
> make.index.unique(xts3,eps=1e-4)
> make.index.unique(xts3,drop=TRUE)
> align.time(xts3,n=3600)
```

Estimate frequency of observations
Convert `xts5` to yearly OHLC
Convert `xts3` to monthly OHLC
Convert `xts5` to quarterly OHLC
Convert to quarterly OHLC
Convert to yearly OHLC
Count the months in `xts5`
Count the quarters in `xts5`
Count the years in `xts5`
Make index unique
Remove duplicate times
Round index time to the next `n` seconds

Other Useful Functions

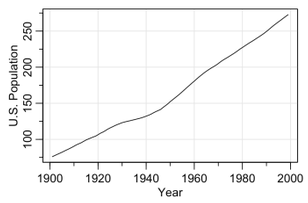
```
> .index(xts4)
> .indexwday(xts3)

> .indexhour(xts3)
> start(xts3)
> end(xts4)
> str(xts3)
> time(xts1)
> head(xts2)
> tail(xts2)
```

Extract raw numeric index of `xts1`
Value of week(day), starting on Sunday, in index of `xts3`
Value of hour in index of `xts3`
Extract first observation of `xts3`
Extract last observation of `xts4`
Display structure of `xts3`
Extract raw numeric index of `xts1`
First part of `xts2`
Last part of `xts2`

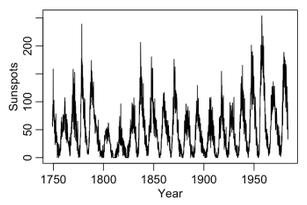
Time Series Cheat Sheet

Plot Time Series

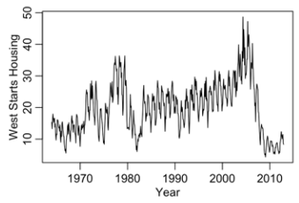
1. tsplot(x=time, y=data)



2. plot(ts(data, start=start_time, frequency=gap))



3. ts.plot(ts(data, start=start_time, frequency=gap))



Simulation

Autoregression of Order p

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + W_t$$

Moving Average of Order q

$$X_t = Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \ldots + \theta_q Z_{t-p}$$

ARMA (p, q)

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \ldots + \theta_q Z_{t-p}$$

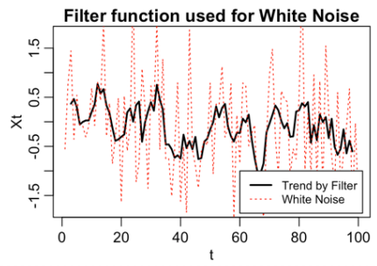
Simulation of ARMA (p, q)

arima.sim(model=list(ar=c($\phi_1, \ldots, \phi_p$), ma=c($\theta_1, \ldots, \theta_q$)), n=n)

Filters

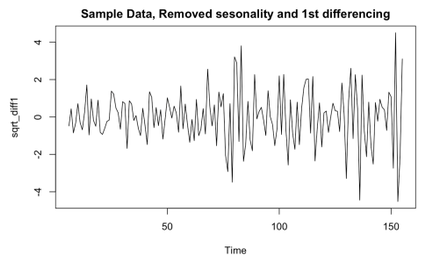
Linear Filter: filter()

filter(data, filter=filter_coefficients, sides=2, method="convolution", circular=F)



Differencing Filter: diff()

diff(data, lag=4, differences=1)

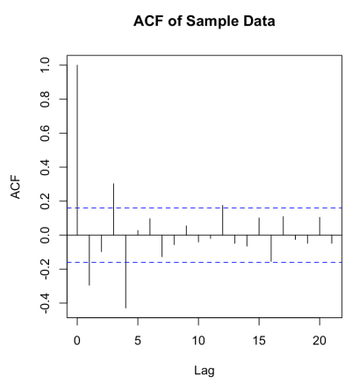


Auto-correlation

Use ACF and PACF to detect model

(Complete) Auto-correlation function: acf()

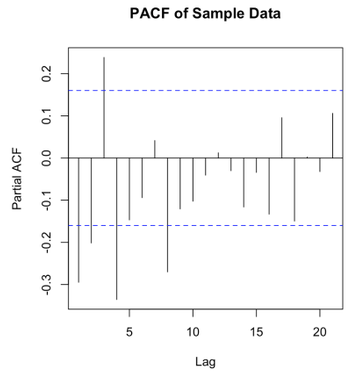
acf(data, type='correlation', na.action=na.pass)



Partial Auto-correlation function: pacf()

pacf(data, na.action=na.pass)

OR: acf(data, type='partial', na.action=na.pass)



Parameter Estimation

Fit an ARMA time series model to the data

ar(): To estimate parameters of an AR model

ar(x=data, aic=T, order.max = NULL, c("yule-walker", "burg", "ols", "mle", "yw"))

```
Call:
ar(x = sqrt1, aic = TRUE, order.max = NULL, method = c("yule-walker",      "burg", "ols", "mle", "yw"))

Coefficients:
      1       2       3       4       5       6       7       8       9
-0.3066  -0.1903   0.0793  -0.5065  -0.1873  -0.1149  -0.0580  -0.3031  -0.1207

Order selected 9  sigma^2 estimated as  1.52
```

arima(): To estimate parameters of an AM or ARMA model, and build model

arima(data, order=c(p, 0, q),method=c('ML'))

```
Call:
arima(x = sqrt1, order = c(2, 0, 6), method = c("ML"))

Coefficients:
          ar1      ar2      ma1     ma2      ma3      ma4     ma5      ma6  intercept
      -0.1658  -0.7951  -0.1536  0.7524  -0.2101  -0.7680  0.0605  -0.6812    -0.0048
s.e.   0.0918   0.0754   0.0916  0.1047   0.0794   0.0743  0.0812   0.0895     0.0062

sigma^2 estimated as 1.193:  log likelihood = -230.78,  aic = 481.55
```

AICc(): Compare models using AICC

AICc(fittedModel)

Forecasting

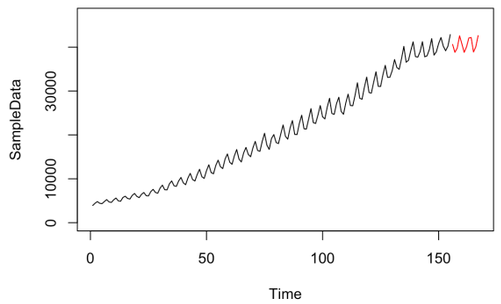
Forecasting future observations given a fitted ARMA model

predict(): Predict future observations given a fitted ARMA model

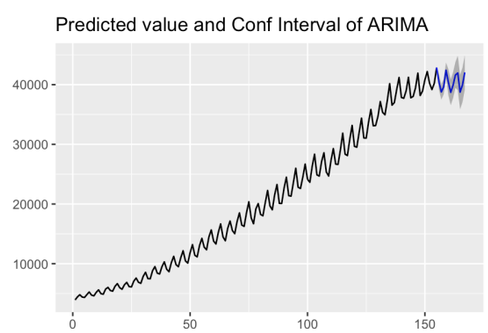
predict(arima_model, number_to_predict)

Plot Predicted values and Confidence Interval:

```
fit<-predict(arima_model, number_to_predict)
ts.plot(data,
        xlim=c(1, length(data)+number_to_predict),
        ylim=c(0, max(fit$pred+1.96*fit$se)))
lines(length(data)+1:length(data)+
      number_to_predict, fit$pred)
```



OR: autoplot(forecast(arima_model, level=c(95), h=number_to_predict))



Deep Learning with Keras : : CHEAT SHEET

Intro

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple back-ends, including TensorFlow, CNTK and Theano.

TensorFlow is a lower level mathematical library for building deep neural network architectures. The `keras` R package makes it easy to use Keras and TensorFlow in R.

Define
- Model
- Sequential model
- Multi-GPU model

Compile
- Optimiser
- Loss
- Metrics

Fit
- Batch size
- Epochs
- Validation split

Evaluate
- Evaluate
- Plot

Predict
- classes
- probability

https://keras.rstudio.com

https://www.manning.com/books/deep-learning-with-r

The "Hello, World!" of deep learning

INSTALLATION

The `keras` R package uses the Python keras library. You can install all the prerequisites directly from R.

https://keras.rstudio.com/reference/install_keras.html

```
library(keras)
install_keras()
```

See ?install_keras for GPU instructions

This installs the required libraries in an Anaconda environment or virtual environment **'r-tensorflow'**.

Working with keras models

DEFINE A MODEL

keras_model() Keras Model

keras_model_sequential() Keras Model composed of a linear stack of layers

multi_gpu_model() Replicates a model on different GPUs

COMPILE A MODEL

compile(object, optimizer, loss, metrics = NULL**)** Configure a Keras model for training

FIT A MODEL

fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, …**)** Train a Keras model for a fixed number of epochs (iterations)

fit_generator() Fits the model on data yielded batch-by-batch by a generator

train_on_batch() test_on_batch() Single gradient update or model evaluation over one batch of samples

EVALUATE A MODEL

evaluate(object, x = NULL, y = NULL, batch_size = NULL**)** Evaluate a Keras model

evaluate_generator() Evaluates the model on a data generator

PREDICT

predict() Generate predictions from a Keras model

predict_proba() and **predict_classes()** Generates probability or class probability predictions for the input samples

predict_on_batch() Returns predictions for a single batch of samples

predict_generator() Generates predictions for the input samples from a data generator

OTHER MODEL OPERATIONS

summary() Print a summary of a Keras model

export_savedmodel() Export a saved model

get_layer() Retrieves a layer based on either its name (unique) or index

pop_layer() Remove the last layer in a model

save_model_hdf5(); load_model_hdf5() Save/ Load models using HDF5 files

serialize_model(); unserialize_model() Serialize a model to an R object

clone_model() Clone a model instance

freeze_weights(); unfreeze_weights() Freeze and unfreeze weights

CORE LAYERS

layer_input() Input layer

layer_dense() Add a densely-connected NN layer to an output

layer_activation() Apply an activation function to an output

layer_dropout() Applies Dropout to the input

layer_reshape() Reshapes an output to a certain shape

layer_permute() Permute the dimensions of an input according to a given pattern

layer_repeat_vector() Repeats the input n times

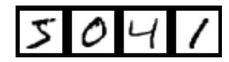
layer_lambda(object, f) Wraps arbitrary expression as a layer

layer_activity_regularization() Layer that applies an update to the cost function based input activity

layer_masking() Masks a sequence by using a mask value to skip timesteps

layer_flatten() Flattens an input

TRAINING AN IMAGE RECOGNIZER ON MNIST DATA



```
# input layer: use MNIST images
mnist <- dataset_mnist()
x_train <- mnist$train$x;  y_train <- mnist$train$y
x_test <- mnist$test$x;  y_test <- mnist$test$y

# reshape and rescale
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255;  x_test <- x_test / 255

y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

# defining the model and layers
model <- keras_model_sequential()
model %>%
 layer_dense(units = 256, activation = 'relu',
             input_shape = c(784)) %>%
 layer_dropout(rate = 0.4) %>%
 layer_dense(units = 128, activation = 'relu') %>%
 layer_dense(units = 10, activation = 'softmax')

# compile (define loss and optimizer)
model %>% compile(
 loss = 'categorical_crossentropy',
 optimizer = optimizer_rmsprop(),
 metrics = c('accuracy')
)

# train (fit)
model %>% fit(
 x_train, y_train,
 epochs = 30, batch_size = 128,
 validation_split = 0.2
)
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test)
```

More layers

CONVOLUTIONAL LAYERS

layer_conv_1d() 1D, e.g. temporal convolution

layer_conv_2d_transpose()
Transposed 2D (deconvolution)

layer_conv_2d() 2D, e.g. spatial convolution over images

layer_conv_3d_transpose()
Transposed 3D (deconvolution)
layer_conv_3d() 3D, e.g. spatial convolution over volumes

layer_conv_lstm_2d()
Convolutional LSTM

layer_separable_conv_2d()
Depthwise separable 2D

layer_upsampling_1d()
layer_upsampling_2d()
layer_upsampling_3d()
Upsampling layer

layer_zero_padding_1d()
layer_zero_padding_2d()
layer_zero_padding_3d()
Zero-padding layer

layer_cropping_1d()
layer_cropping_2d()
layer_cropping_3d()
Cropping layer

POOLING LAYERS

layer_max_pooling_1d()
layer_max_pooling_2d()
layer_max_pooling_3d()
Maximum pooling for 1D to 3D

layer_average_pooling_1d()
layer_average_pooling_2d()
layer_average_pooling_3d()
Average pooling for 1D to 3D

layer_global_max_pooling_1d()
layer_global_max_pooling_2d()
layer_global_max_pooling_3d()
Global maximum pooling

layer_global_average_pooling_1d()
layer_global_average_pooling_2d()
layer_global_average_pooling_3d()
Global average pooling

ACTIVATION LAYERS

layer_activation(object, activation**)**
Apply an activation function to an output

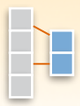
layer_activation_leaky_relu()
Leaky version of a rectified linear unit

layer_activation_parametric_relu()
Parametric rectified linear unit

layer_activation_thresholded_relu()
Thresholded rectified linear unit

layer_activation_elu()
Exponential linear unit

DROPOUT LAYERS



layer_dropout()
Applies dropout to the input

layer_spatial_dropout_1d()
layer_spatial_dropout_2d()
layer_spatial_dropout_3d()
Spatial 1D to 3D version of dropout

RECURRENT LAYERS

layer_simple_rnn()
Fully-connected RNN where the output is to be fed back to input

layer_gru()
Gated recurrent unit - Cho et al

layer_cudnn_gru()
Fast GRU implementation backed by CuDNN

layer_lstm()
Long-Short Term Memory unit - Hochreiter 1997

layer_cudnn_lstm()
Fast LSTM implementation backed by CuDNN

LOCALLY CONNECTED LAYERS

layer_locally_connected_1d()
layer_locally_connected_2d()
Similar to convolution, but weights are not shared, i.e. different filters for each patch

Preprocessing

SEQUENCE PREPROCESSING

pad_sequences()
Pads each sequence to the same length (length of the longest sequence)

skipgrams()
Generates skipgram word pairs

make_sampling_table()
Generates word rank-based probabilistic sampling table

TEXT PREPROCESSING

text_tokenizer() Text tokenization utility

fit_text_tokenizer() Update tokenizer internal vocabulary

save_text_tokenizer(); load_text_tokenizer()
Save a text tokenizer to an external file

texts_to_sequences();
texts_to_sequences_generator()
Transforms each text in texts to sequence of integers

texts_to_matrix(); sequences_to_matrix()
Convert a list of sequences into a matrix

text_one_hot() One-hot encode text to word indices

text_hashing_trick()
Converts a text to a sequence of indexes in a fixed-size hashing space

text_to_word_sequence()
Convert text to a sequence of words (or tokens)

IMAGE PREPROCESSING

image_load() Loads an image into PIL format.

flow_images_from_data()
flow_images_from_directory()
Generates batches of augmented/normalized data from images and labels, or a directory

image_data_generator() Generate minibatches of image data with real-time data augmentation.

fit_image_data_generator() Fit image data generator internal statistics to some sample data

generator_next() Retrieve the next item

image_to_array(); image_array_resize()
image_array_save() 3D array representation

Keras TensorFlow

Pre-trained models

Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

application_xception()
xception_preprocess_input()
Xception v1 model

application_inception_v3()
inception_v3_preprocess_input()
Inception v3 model, with weights pre-trained on ImageNet

application_inception_resnet_v2()
inception_resnet_v2_preprocess_input()
Inception-ResNet v2 model, with weights trained on ImageNet

application_vgg16(); application_vgg19()
VGG16 and VGG19 models

application_resnet50() ResNet50 model

application_mobilenet()
mobilenet_preprocess_input()
mobilenet_decode_predictions()
mobilenet_load_model_hdf5()
MobileNet model architecture

IMAGENET
ImageNet is a large database of images with labels, extensively used for deep learning

imagenet_preprocess_input()
imagenet_decode_predictions()
Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions

Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

callback_early_stopping() Stop training when a monitored quantity has stopped improving
callback_learning_rate_scheduler() Learning rate scheduler
callback_tensorboard() TensorBoard basic visualizations

RStudio

Need to Know

Pattern arguments in stringr are interpreted as regular expressions *after any special characters have been parsed.*

In R, you write regular expressions as *strings*, sequences of characters surrounded by quotes (**""**) or single quotes(**''**).

Some characters cannot be represented directly in an R string . These must be represented as **special characters**, sequences of characters that have a specific meaning., e.g.

Special Character	Represents
\\	\
\"	"
\n	new line

Run **?"'"** to see a complete list

Because of this, whenever a \ appears in a regular expression, you must write it as \\ in the string that represents the regular expression.

Use **writeLines()** to see how R views your string after all special characters have been parsed.

```
writeLines("\\.")
# \.

writeLines("\\ is a backslash")
# \ is a backslash
```

INTERPRETATION

Patterns in stringr are interpreted as regexs. To change this default, wrap the pattern in one of:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...**)** Modifies a regex to ignore cases, match end of lines as well of end of strings, allow R comments within regex's , and/or to have . match everything including \n.
str_detect("I", regex("i", TRUE))

fixed() Matches raw bytes but will miss some characters that can be represented in multiple ways (fast). str_detect("\u0130", fixed("i"))

coll() Matches raw bytes and will use locale specific collation rules to recognize characters that can be represented in multiple ways (slow).
str_detect("\u0130", coll("i", TRUE, locale = "tr"))

boundary() Matches boundaries between characters, line_breaks, sentences, or words.
str_split(sentences, boundary("word"))

Regular Expressions -

Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

MATCH CHARACTERS

see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)

string (type this)	regexp (to mean this)	matches (which matches this)	example	
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
\\.	**\.**	.	see("\\.")	abc ABC 123 .!?\(){}
\\!	**\!**	!	see("\\!")	abc ABC 123 .!?\(){}
\\?	**\?**	?	see("\\?")	abc ABC 123 .!?\(){}
\\\\	****	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	**\(**	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	**\)**	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	**\{**	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	**\}**	}	see("\\}")	abc ABC 123 .!?\(){}
\\n	**\n**	new line (return)	see("\\n")	abc ABC 123 .!?\(){}
\\t	**\t**	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	**\s**	any whitespace (**\S** *for non-whitespaces*)	see("\\s")	abc ABC 123 .!?\(){}
\\d	**\d**	any digit (**\D** *for non-digits*)	see("\\d")	abc ABC 123 .!?\(){}
\\w	**\w**	any word character (**\W** *for non-word chars*)	see("\\w")	abc ABC 123 .!?\(){}
\\b	**\b**	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:][1]	digits	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:][1]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:][1]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:][1]	uppercase letters	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:][1]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:][1]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:][1]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:][1]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:][1]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	every character except a new line	see(".")	abc ABC 123 .!?\(){}

[1] Many base R functions require classes to be wrapped in a second set of **[ ]**, e.g. **[[:digit:]]**

[:space:]
↵ new line

[:blank:]
space
tab

.

[:graph:]

[:punct:]
. , : ; ? ! / * @ #
- _ " ' [ ] { } ( )

[:symbol:]
| ` = + ^
~ < > $

[:alnum:]

[:digit:]
0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:]
a b c d e f
g h i j k l
m n o p q r
s t u v w x
y z

[:upper:]
A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Y Z

ALTERNATES

alt <- function(rx) str_view_all("abcde", rx)

regexp	matches	example	
ab|d	or	alt("ab|d")	abcde
[abe]	one of	alt("[abe]")	abcde
[^abe]	anything but	alt("[^abe]")	abcde
[a-c]	range	alt("[a-c]")	abcde

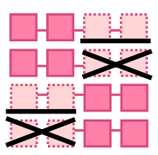
ANCHORS

anchor <- function(rx) str_view_all("aaa", rx)

regexp	matches	example	
^a	start of string	anchor("^a")	aaa
a$	end of string	anchor("a$")	aaa

LOOK AROUNDS

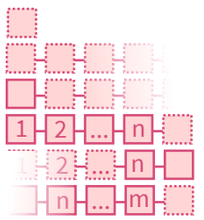
look <- function(rx) str_view_all("bacad", rx)



regexp	matches	example	
a(?=c)	followed by	look("a(?=c)")	bacad
a(?!c)	not followed by	look("a(?!c)")	bacad
(?<=b)a	preceded by	look("(?<=b)a")	bacad
(?<!b)a	not preceded by	look("(?<!b)a")	bacad

QUANTIFIERS

quant <- function(rx) str_view_all(".a.aa.aaa", rx)



regexp	matches	example	
a?	zero or one	quant("a?")	.a.aa.aaa
a*	zero or more	quant("a*")	.a.aa.aaa
a+	one or more	quant("a+")	.a.aa.aaa
a{n}	exactly **n**	quant("a{2}")	.a.aa.aaa
a{n,}	**n** or more	quant("a{2,}")	.a.aa.aaa
a{n, m}	between **n** and **m**	quant("a{2,4}")	.a.aa.aaa

GROUPS

ref <- function(rx) str_view_all("abbaab", rx)

Use parentheses to set precedent (order of evaluation) and create groups

regexp	matches	example	
(ab|d)e	sets precedence	alt("(ab|d)e")	abcde

Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance

string (type this)	regexp (to mean this)	matches (which matches this)	example (the result is the same as ref("abba"))	
\\1	**\1** (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")	abbaab

Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                          ] required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat =<STAT>, position =<POSITION>) +            ] Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5**)** Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

RStudio

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and **a + expand_limits()**
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend = lat + 1, xend = long + 1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin"**)**
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)**
x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES both continuous

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)**
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

j + geom_crossbar(fatten = 2**)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width
Also **geom_errorbarh()**.

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z)**)**
x, y, z, alpha, color, group, linetype, size, weight

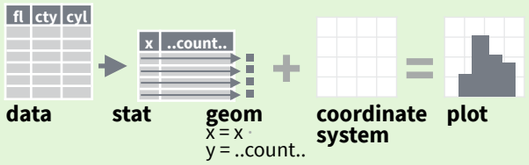
l + geom_contour_filled(aes(fill = z)**)**
x, y, alpha, color, fill, group, linetype, size, subgroup

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density_2d(aes(fill = ..level..), geom = "polygon"**)**

variable created by stat

c + stat_bin(binwidth = 1, boundary = 10**)**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1**) x, y** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian"**)**
x, y | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T**)**
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins = 30**) x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100**)**
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t"**)**

l + stat_contour(aes(z = z)**) x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max**)**
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5**)**
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area"**) x, y**
| ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40**) x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq"**) x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se = T,
level = 0.95**) x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + xlim(-5, 5**) + stat_function(**fun = dnorm,
n = 20, geom = "point"**) x** | ..x.., ..y..

ggplot() + stat_qq(aes(sample = 1:100)**)**
x, y, sample | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot"**)**

h + stat_summary_bin(fun = "mean", geom = "bar"**)**

e + stat_identity()

e + stat_unique()

Scales Override defaults with **scales** package.

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

n <- d + geom_bar(aes(fill = fl))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R")**)**

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()**)** - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d"),
date_breaks = "2 weeks"**)** - Treat data values as dates.
scale_*_datetime() - Treat data values as date times. Same as scale_*_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.

COLOR AND FILL SCALES (DISCRETE)

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2,
end = 0.8, na.value = "red"**)**

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues"**)**

o + scale_fill_gradient(low="red", high="yellow"**)**

o + scale_fill_gradient2(low = "red", high = "blue",
mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colors = topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(),
cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))

p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6)**)**
p + scale_size_area(max_size = 6**)**

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5)**)** - xlim, ylim
The default cartesian coordinate system.

r + coord_fixed(ratio = 1/2**)**
ratio, xlim, ylim - Cartesian coordinates with fixed aspect ratio between x and y units.

ggplot(mpg, aes(y = fl)**) + geom_bar()**
Flip cartesian coordinates by switching x and y aesthetic mappings.

r + coord_polar(theta = "x", direction=1**)**
theta, start, direction - Polar coordinates.

r + coord_trans(y = "sqrt"**)** - x, y, xlim, ylim
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation = c(41, -74, 0)**)** - projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.).

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side.

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height.

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting.

e + geom_label(position = "nudge")
Nudge labels away from points.

s + geom_bar(position = "stack")
Stack elements on top of one another.

Each position adjustment can be recast as a function with manual **width** and **height** arguments:
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines.

r + theme_gray()
Grey background (default theme).

r + theme_dark()
Dark for contrast.

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal theme.

r + theme_void()
Empty theme.

r + theme() Customize aspects of the theme such as axis, legend, panel, and facet properties.
r + ggtitle("Title") + theme(plot.title.postion = "plot")
r + theme(panel.background = element_rect(fill = "blue"))

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
Facet into columns based on fl.

t + facet_grid(rows = vars(year))
Facet into rows based on year.

t + facet_grid(rows = vars(year), cols = vars(fl))
Facet into both rows and columns.

t + facet_wrap(vars(fl))
Wrap facets into a rectangular layout.

Set **scales** to let axis limits vary across facets.

t + facet_grid(rows = vars(drv), cols = vars(fl),
scales = "free")
x and y axis limits adjust to individual facets:
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet label:

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl),
labeller = label_bquote(alpha ^ .(fl)**))**

α^c α^d α^e α^p α^r

Labels and Legends

Use **labs()** to label the elements of your plot.

t + labs(x = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
alt = "Add alt text to the plot",
<AES> = "New <AES> legend title"**)**

t + annotate(geom = "text", x = 8, y = 9, label = "A"**)**
Places a geom with manually selected aesthetics.

p + guides(x = guide_axis(n.dodge = 2)**)** Avoid crowded or overlapping labels with guide_axis(n.dodge or angle).

n + guides(fill = "none"**)** Set legend type for each aesthetic: colorbar, legend, or none (no legend).

n + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right".

n + scale_fill_discrete(name = "Title",
labels = c("A", "B", "C", "D", "E")**)**
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred):
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)**

With clipping (removes unseen data points):
t + xlim(0, 100**) + ylim(**10, 20**)**

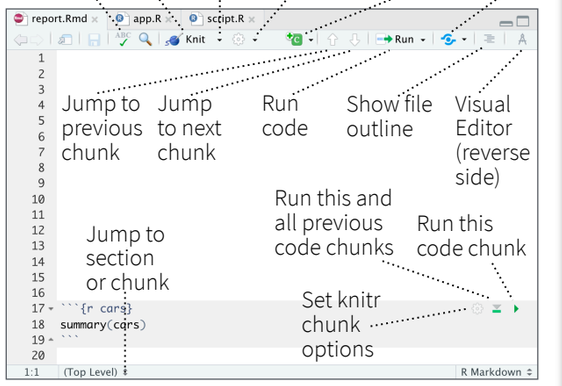
t + scale_x_continuous(limits = c(0, 100)**) +**
scale_y_continuous(limits = c(0, 100)**)**

RStudio

RStudio IDE : : **CHEAT SHEET**

Documents and Apps

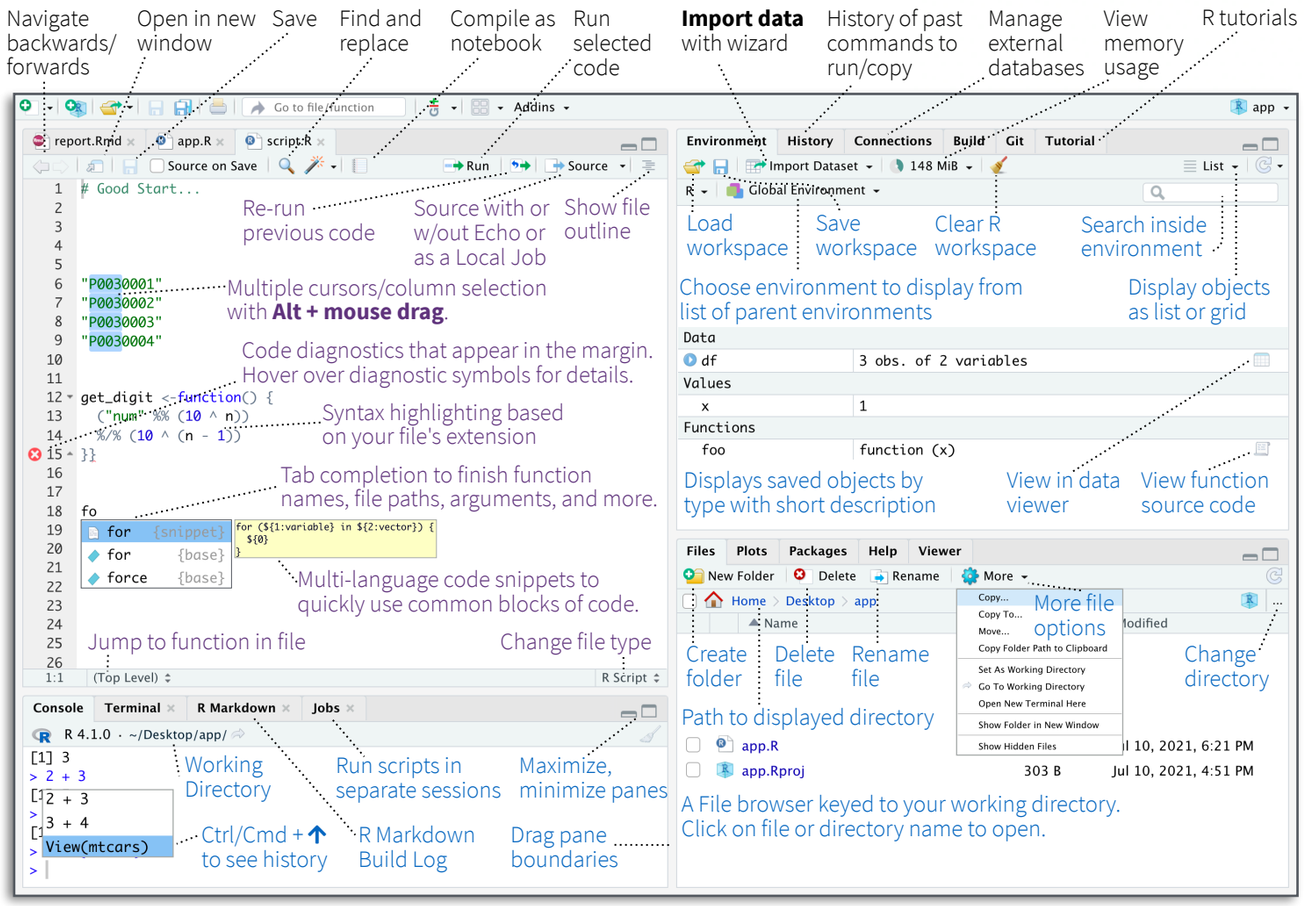
Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane



Access markdown guide at
Help > Markdown Quick Reference
See reverse side for more on **Visual Editor**

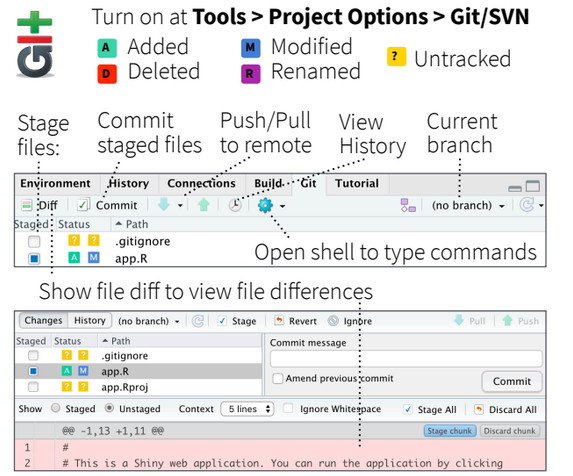
RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

Source Editor



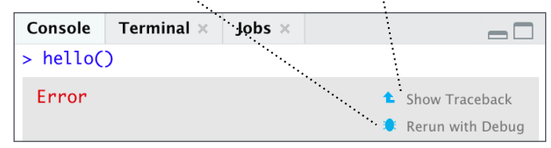
Tab Panes

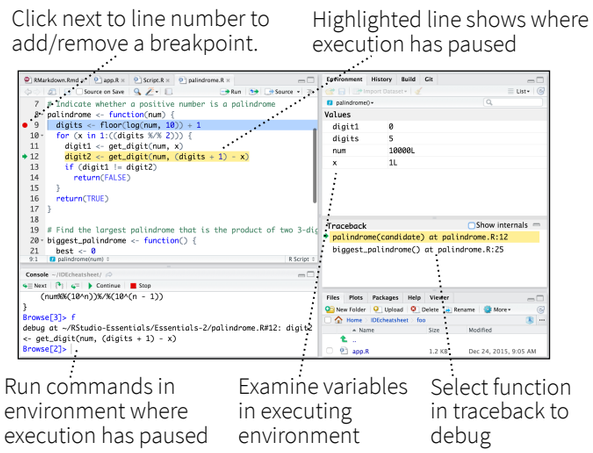
Version Control

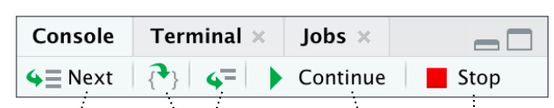


Debug Mode

Use **debug(), browser(),** or a breakpoint and execute your code to open the debugger mode.





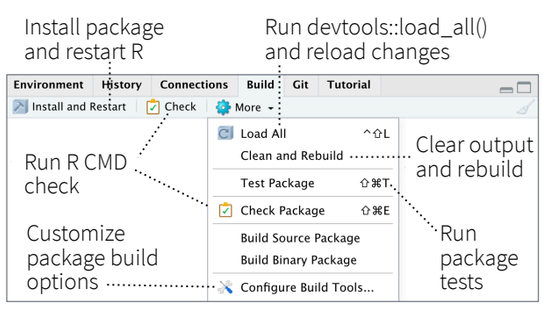


Package Development

Create a new package with
File > New Project > New Directory > R Package
Enable roxygen documentation with
Tools > Project Options > Build Tools

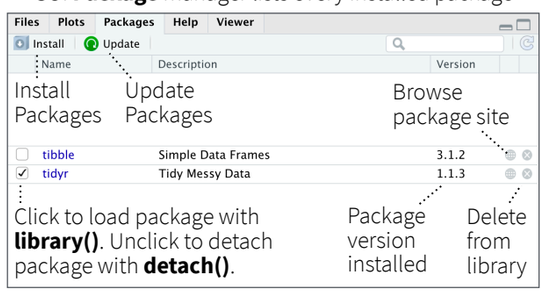
Roxygen guide at **Help > Roxygen Quick Reference**

See package information in the **Build Tab**

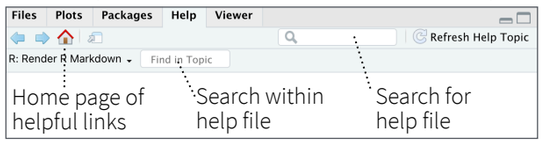


RStudio opens plots in a dedicated **Plots** pane

GUI **Package** manager lists every installed package

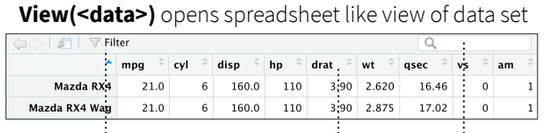


RStudio opens documentation in a dedicated **Help** pane



Viewer pane displays HTML content, such as Shiny apps, RMarkdown reports, and interactive visualizations

View(<data>) opens spreadsheet like view of data set



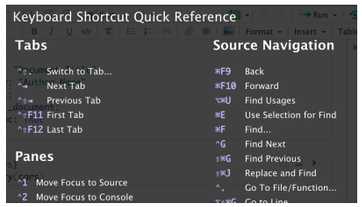
Keyboard Shortcuts

RUN CODE	Windows/Linux	Mac
Search command history	Ctrl+↑	Cmd+↑
Interrupt current command	Esc	Esc
Clear console	Ctrl+L	Ctrl+L
NAVIGATE CODE		
Go to File/Function	Ctrl+.	Ctrl+.
WRITE CODE		
Attempt completion	Tab or Ctrl+Space	Tab or Ctrl+Space
Insert <- (assignment operator)	Alt+-	Option+-
Insert %>% (pipe operator)	Ctrl+Shift+M	Cmd+Shift+M
(Un)Comment selection	Ctrl+Shift+C	Cmd+Shift+C

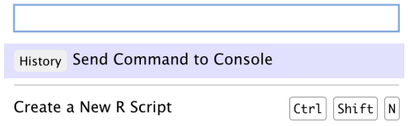
MAKE PACKAGES	Windows/Linux	Mac
Load All (devtools)	Ctrl+Shift+L	Cmd+Shift+L
Test Package (Desktop)	Ctrl+Shift+T	Cmd+Shift+T
Document Package	Ctrl+Shift+D	Cmd+Shift+D

DOCUMENTS AND APPS		
Knit Document (knitr)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave & Knitr)	Ctrl+Alt+I	Cmd+Option+I
Run from start to current line	Ctrl+Alt+B	Cmd+Option+B
MORE KEYBOARD SHORTCUTS		
Keyboard Shortcuts Help	Alt+Shift+K	Option+Shift+K
Show Command Palette	Ctrl+Shift+P	Cmd+Shift+P

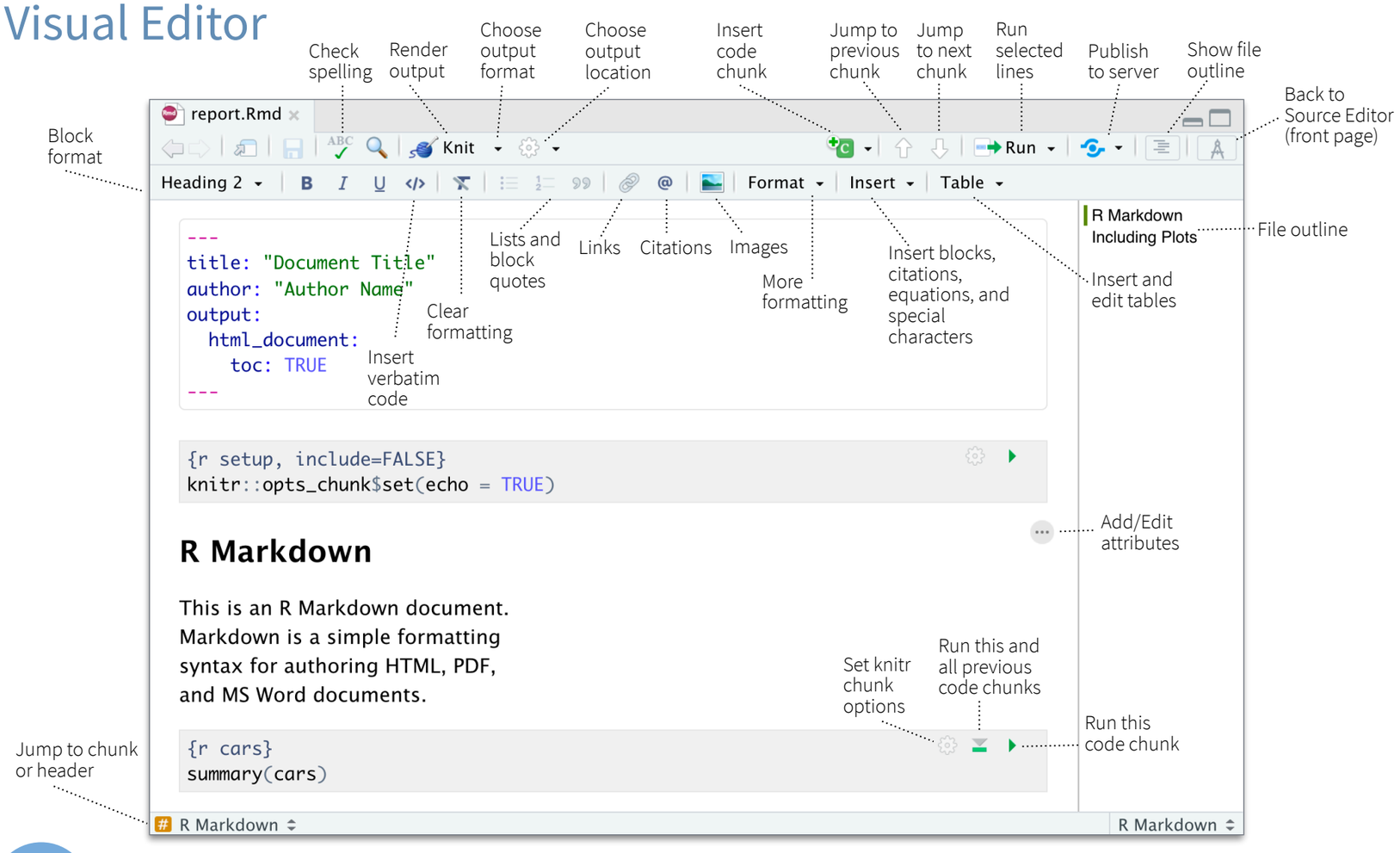
View the Keyboard Shortcut Quick Reference with **Tools > Keyboard Shortcuts** or **Alt/Option + Shift + K**



Search for keyboard shortcuts with **Tools > Show Command Palette** or **Ctrl/Cmd + Shift + P**.



Visual Editor



RStudio Workbench

WHY RSTUDIO WORKBENCH?

Extend the open source server with a commercial license, support, and more:

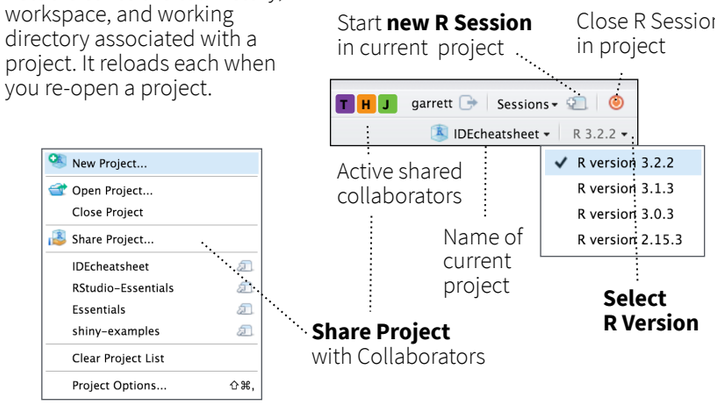
- open and run multiple R sessions at once
- tune your resources to improve performance
- administrative tools for managing user sessions
- collaborate real-time with others in shared projects
- switch easily from one version of R to a different version
- integrate with your authentication, authorization, and audit practices
- work in the RStudio IDE, JupyterLab, Jupyter Notebooks, or VS Code

Download a free 45 day evaluation at
www.rstudio.com/products/workbench/evaluation/

Share Projects

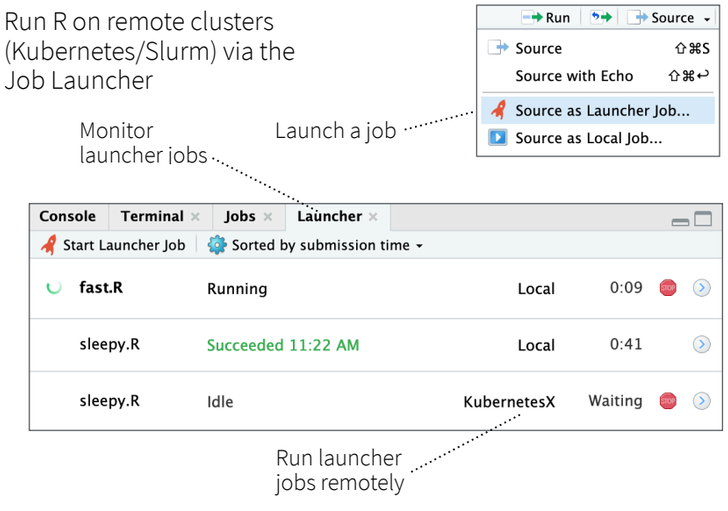
File > New Project

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.



Run Remote Jobs

Run R on remote clusters (Kubernetes/Slurm) via the Job Launcher



6.3 SQL language

Study Guide: Data Retrieval with SQL

Afshine Amidi and Shervine Amidi

August 21, 2020

General concepts

❒ **Structured Query Language** – Structured Query Language, abbreviated as SQL, is a language that is largely used in the industry to query data from databases.

❒ **Query structure** – Queries are usually structured as follows:

SQL

```
-- Select fields                    mandatory
SELECT
    col_1,
    col_2,
     ... ,
    col_n

-- Source of data                   mandatory
FROM table t

-- Gather info from other sources   optional
JOIN other_table ot
  ON (t.key = ot.key)

-- Conditions                       optional
WHERE some_condition(s)

-- Aggregating                      optional
GROUP BY column_group_list

-- Sorting values                   optional
ORDER BY column_order_list

-- Restricting aggregated values    optional
HAVING some_condition(s)

-- Limiting number of rows          optional
LIMIT some_value
```

Remark: the `SELECT DISTINCT` *command can be used to ensure not having duplicate rows.*

❒ **Condition** – A condition is of the following format:

SQL

```
some_col some_operator some_col_or_value
```

where `some_operator` can be among the following common operations:

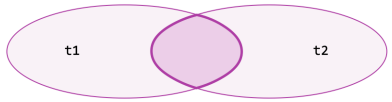
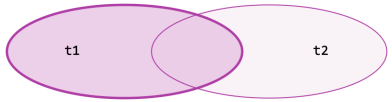
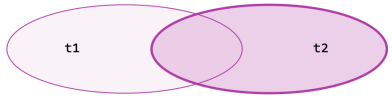
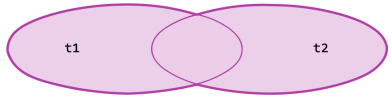
Category	Operator	Command
General	Equality / non-equality	`=` / `!=`, `<>`
	Inequalities	`>=`, `>`, `<`, `<=`
	Belonging	`IN (val_1, ..., val_n)`
	And / or	`AND` / `OR`
	Check for missing value	`IS NULL`
	Between bounds	`BETWEEN val_1 AND val_2`
Strings	Pattern matching	`LIKE '%val%'`

❒ **Joins** – Two tables `table_1` and `table_2` can be joined in the following way:

SQL

```
...

FROM table_1 t1
type_of_join table_2 t2
  ON (t2.key = t1.key)

...
```

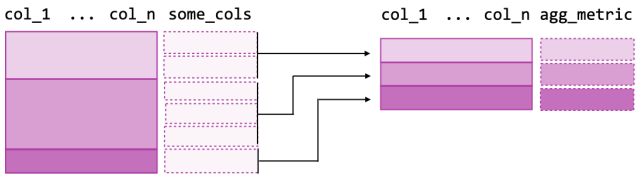
where the different `type_of_join` commands are summarized in the table below:

Type of join	Illustration
`INNER JOIN`	
`LEFT JOIN`	
`RIGHT JOIN`	
`FULL JOIN`	

Remark: joining every row of table 1 with every row of table 2 can be done with the `CROSS JOIN` *command, and is commonly known as the cartesian product.*

Aggregations

❒ **Grouping data** – Aggregate metrics are computed on grouped data in the following way:



The SQL command is as follows:

SQL

```
SELECT
    col_1,
    agg_function(col_2)
FROM table
GROUP BY col_1
```

❒ **Grouping sets** – The GROUPING SETS command is useful when there is a need to compute aggregations across different dimensions at a time. Below is an example of how all aggregations across two dimensions are computed:

SQL

```
SELECT
    col_1,
    col_2,
    agg_function(col_3)
FROM table
GROUP BY (
  GROUPING SETS
    (col_1),
    (col_2),
    (col_1, col_2)
)
```

❒ **Aggregation functions** – The table below summarizes the main aggregate functions that can be used in an aggregation query:

Category	Operation	Command
Values	Mean	AVG(col)
	Percentile	PERCENTILE_APPROX(col, p)
	Sum / # of instances	SUM(col) / COUNT(col)
	Max / min	MAX(col) / MIN(col)
	Variance / standard deviation	VAR(col) / STDEV(col)
Arrays	Concatenate into array	collect_list(col)

Remark: the median can be computed using the PERCENTILE_APPROX *function with* p *equal to* 0.5.

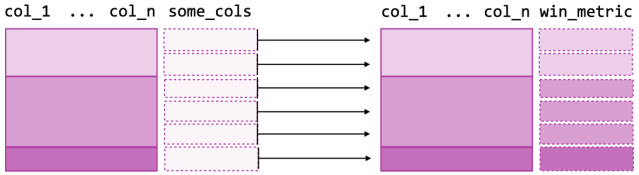
❒ **Filtering** – The table below highlights the differences between the WHERE and HAVING commands:

WHERE	HAVING
- Filter condition applies to individual rows - Statement placed right after FROM	- Filter condition applies to aggregates - Statement placed right after GROUP BY

Remark: if WHERE *and* HAVING *are both in the same query,* WHERE *will be executed first.*

Window functions

❒ **Definition** – A window function computes a metric over groups and has the following structure:



The SQL command is as follows:

SQL

```
some_window_function() OVER(PARTITION BY some_col ORDER BY another_col)
```

Remark: window functions are only allowed in the SELECT *clause.*

❒ **Row numbering** – The table below summarizes the main commands that rank each row across specified groups, ordered by a specific column:

Command	Description	Example
ROW_NUMBER()	Ties are given different ranks	1, 2, 3, 4
RANK()	Ties are given same rank and skip numbers	1, 2, 2, 4
DENSE_RANK()	Ties are given same rank and don't skip numbers	1, 2, 2, 3

❒ **Values** – The following window functions allow to keep track of specific types of values with respect to the partition:

Command	Description
FIRST_VALUE(col)	Takes the first value of the column
LAST_VALUE(col)	Takes the last value of the column
LAG(col, n)	Takes the n^{th} previous value of the column
LEAD(col, n)	Takes the n^{th} following value of the column
NTH_VALUE(col, n)	Takes the n^{th} value of the column

Advanced functions

❒ **SQL tips** – In order to keep the query in a clear and concise format, the following tricks are often done:

Operation	Command	Description
Renaming columns	`SELECT operation_on_column AS col_name`	New column names shown in query results
Abbreviating tables	`FROM table_1 t1`	Abbreviation used within query for simplicity in notations
Simplifying group by	`GROUP BY col_number_list`	Specify column position in `SELECT` clause instead of whole column names
Limiting results	`LIMIT n`	Display only `n` rows

❒ **Sorting values** – The query results can be sorted along a given set of columns using the following command:

SQL
```
... [query] ...
ORDER BY col_list
```

Remark: by default, the command sorts in ascending order. If we want to sort it in descending order, the `DESC` command needs to be used after the column.

❒ **Column types** – In order to ensure that a column or value is of one specific data type, the following command is used:

SQL
```
CAST(some_col_or_value AS data_type)
```

where `data_type` is one of the following:

Data type	Description	Example
`INT`	Integer	`2`
`DOUBLE`	Numerical value	`2.0`
`STRING`	String	`'teddy bear'`
`VARCHAR`		
`DATE`	Date	`'2020-01-01'`
`TIMESTAMP`	Timestamp	`'2020-01-01 00:00:00.000'`

Remark: if the column contains data of different types, the `TRY_CAST()` command will convert unknown types to `NULL` instead of throwing an error.

❒ **Column manipulation** – The main functions used to manipulate columns are described in the table below:

Category	Operation	Command
General	Take first non-`NULL` value	`COALESCE(col_1, col_2, ..., col_n)`
	Create a new column combining existing ones	`CONCAT(col_1, ..., col_n)`
Value	Round value to `n` decimals	`ROUND(col, n)`
String	Converts string column to lower / upper case	`LOWER(col)` / `UPPER(col)`
	Replace occurrences of `old` in col to `new`	`REPLACE(col, old, new)`
	Take the substring of col, with a given `start` and `length`	`SUBSTR(col, start, length)`
	Remove spaces from the left / right / both sides	`LTRIM(col)` / `RTRIM(col)` / `TRIM(col)`
	Length of the string	`LENGTH(col)`
Date	Truncate at a given granularity (`year`, `month`, `week`)	`DATE_TRUNC(time_dimension, col_date)`
	Transform date	`DATE_ADD(col_date, number_of_days)`

❒ **Conditional column** – A column can take different values with respect to a particular set of conditions with the `CASE WHEN` command as follows:

SQL
```
CASE WHEN some_condition THEN some_value
          ...
     WHEN some_other_condition THEN some_other_value
     ELSE some_other_value_n END
```

❒ **Combining results** – The table below summarizes the main ways to combine results in queries:

Category	Command	Remarks
Union	`UNION`	Guarantees distinct rows
	`UNION ALL`	Potential newly-formed duplicates are kept
Intersection	`INTERSECT`	Keeps observations that are in all selected queries

❒ **Common table expression** – A common way of handling complex queries is to have temporary result sets coming from intermediary queries, which are called common table expressions (abbreviated CTE), that increase the readability of the overall query. It is done thanks to the `WITH ... AS ...` command as follows:

SQL
```
WITH cte_1 AS (
SELECT ...
),
```

```
...

cte_n AS (
SELECT ...
)

SELECT ...
FROM ...
```

Table manipulation

❒ **Table creation** – The creation of a table is done as follows:

SQL

```
CREATE [table_type] TABLE [creation_type] table_name(
  col_1 data_type_1,
        ...       ,
  col_n data_type_n
)
[options];
```

where [table_type], [creation_type] and [options] are one of the following:

Category	Command	Description
Table type	Blank	Default table
	EXTERNAL TABLE	External table
Creation type	Blank	Creates table and overwrites current one if it exists
	IF NOT EXISTS	Only creates table if it does not exist
Options	location 'path_to_hdfs_folder'	Populate table with data from hdfs folder
	stored as data_format	Stores the table in a specific data format, e.g. parquet, orc or avro

❒ **Data insertion** – New data can either append or overwrite already existing data in a given table as follows:

SQL

```
WITH ...                          -- optional

INSERT [insert_type] table_name   -- mandatory

SELECT ...;                       -- mandatory
```

where [insert_type] is among the following:

Command	Description
OVERWRITE	Overwrites existing data
INTO	Appends to existing data

❒ **Dropping table** – Tables are dropped in the following way:

SQL

```
DROP TABLE table_name;
```

❒ **View** – Instead of using a complicated query, the latter can be saved as a view which can then be used to get the data. A view is created with the following command:

SQL

```
CREATE VIEW view_name AS complicated_query;
```

Remark: a view does not create any physical table and is instead seen as a shortcut.

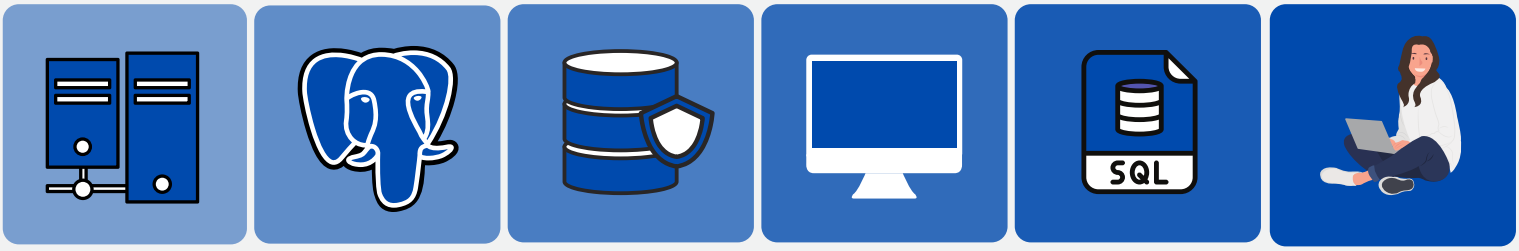
Sources

- W3Schools.com
- DataQuest.io

SQL CHEATSHEET

CONSIDER SUPPORTING ME

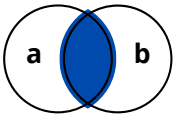
@AbzAaron



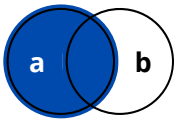
Commands / Clauses

SELECT	Select data from database
FROM	Specify table we're pulling from
WHERE	Filter query to match a condition
AS	Rename column or table with alias
JOIN	Combine rows from 2 or more tables
AND	Combine query conditions. All must be met
OR	Combine query conditions. One must be met
LIMIT	Limit rows returned. See also FETCH & TOP
IN	Specify multiple values when using WHERE
CASE	Return value on a specified condition
IS NULL	Return only rows with a NULL value
LIKE	Search for patterns in column
COMMIT	Write transaction to database
ROLLBACK	Undo a transaction block
ALTER TABLE	Add/Remove columns from table
UPDATE	Update table data
CREATE	Create TABLE, DATABASE, INDEX or VIEW
DELETE	Delete rows from table
INSERT	Add single row to table
DROP	Delete TABLE, DATABASE, or INDEX
GROUP BY	Group data into logical sets
ORDER BY	Set order of result. Use DESC to reverse order
HAVING	Same as WHERE but filters groups
COUNT	Count number of rows
SUM	Return sum of column
AVG	Return average of column
MIN	Return min value of column
MAX	Return max value of column

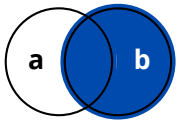
Joins



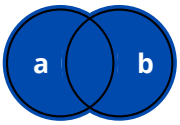
a INNER JOIN b



a LEFT JOIN b



a RIGHT JOIN b



a FULL OUTER JOIN b

Examples

Select all columns with filter applied

```
SELECT * FROM tbl
WHERE col > 5;
```

Select first 10 rows for two columns

```
SELECT col1, col2
FROM tbl LIMIT 10;
```

Select all columns with multiple filters

```
SELECT * FROM tbl
WHERE col1 > 5 OR col2 < 2;
```

Select all rows from col1 & col2 ordering by col1

```
SELECT col1, col2
FROM tbl ORDER BY 1;
```

Return count of rows in table

```
SELECT COUNT(*)
FROM tbl;
```

Return sum of col1

```
SELECT SUM(col1)
FROM tbl;
```

Return max value for col1

```
SELECT MAX(col1)
FROM tbl;
```

Compute summary stats by grouping col2

```
SELECT AVG(col1) FROM tbl
GROUP BY col2;
```

Combine data from 2 tables using left join

```
SELECT * FROM tbl1 AS t1 LEFT JOIN
tbl2 AS t2 ON t2.col1 = t1.col1;
```

Aggregate and filter result

```
SELECT col1,
       COUNT(*) AS total
FROM tbl
GROUP BY col1
HAVING COUNT(*) > 10;
```

Implementation of CASE statement

```
SELECT col1,
CASE
    WHEN col1 > 10 THEN 'more than 10'
    WHEN col1 < 10 THEN 'less than 10'
    ELSE '10'
END AS NewColumnName
FROM tbl;
```

Data Definition Language

CREATE

```
CREATE DATABASE MyDatabase;

CREATE TABLE MyTable (
    id int,
    name varchar(10));

CREATE INDEX IndexName
ON TableName(col1);
```

ALTER

```
ALTER TABLE MyTable
DROP COLUMN col5;

ALTER TABLE MyTable
ADD col5 int;
```

DROP

```
DROP DATABASE MyDatabase;
DROP TABLE MyTable;
```

Data Manipulation Language

UPDATE

```
UPDATE MyTable
SET col1 = 56
WHERE col2 = 'something';
```

INSERT

```
INSERT INTO MyTable (col1, col2)
VALUES ('value1', 'value2');
```

DELETE

```
DELETE FROM MyTable
WHERE col1 = 'something';
```

SELECT

```
SELECT col1, col2
FROM MyTable;
```

Order Of Execution

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT

SQL cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Basic Queries

```
-- filter your columns
   SELECT col1, col2, col3, ... FROM table1
-- filter the rows
   WHERE col4 = 1 AND col5 = 2
-- aggregate the data
   GROUP by ...
-- limit aggregated data
   HAVING count(*) > 1
-- order of the results
   ORDER BY col2
```

Useful keywords for **SELECTS**:

DISTINCT - return unique results
BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates
LIKE - pattern search within the column text
IN (a, b, c) - check if the value is contained among given.

Data Modification

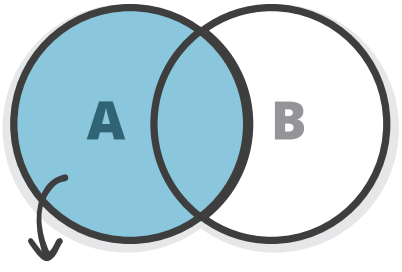
```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT id, last_name, first_name FROM table2
```

Views

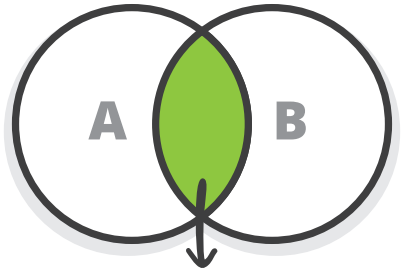
A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

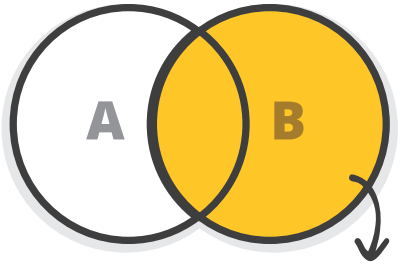
The Joy of JOINs



LEFT OUTER JOIN - *all rows from table A, even if they do not exist in table B*



INNER JOIN - *fetch the results that exist in both tables*



RIGHT OUTER JOIN - *all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
   (SELECT t1_id FROM table2 WHERE date >
      CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

Don't forget:

Avoid overlapping indexes
Avoid indexing on too many columns
Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
   TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
   COALESCE (col1, col2, "default value")
-- return current time:
   CURRENT_TIMESTAMP
-- compute set operations on two result sets
   SELECT col1, col2 FROM table1
   UNION / EXCEPT / INTERSECT
   SELECT col3, col4 FROM table2;
```

Union - returns data from both queries
Except - rows from the first query that are not present in the second query
Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows
SUM - cumulate the values
AVG - return the average for the group
MIN / MAX - smallest / largest value

7. Business Intelligence


CRM
ERP
PIM
Operational System
Server
Flat Files
ETL
Extract
Transform
Load
Data Warehouse
Business Inteligence
Data Mining
Business Analytics
Big Data
Predictive Analysis
Reporting
Planning
Data Visualisation

7.1 Tableau

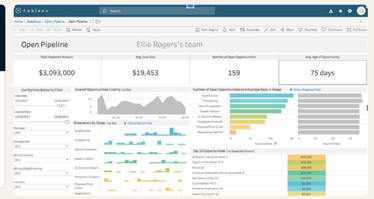
Tableau for Business Intelligence

Tableau Basics Cheat Sheet

Learn Tableau online at www.DataCamp.com

What is Tableau?

Tableau is a business intelligence tool that allows you to effectively report insights through easy-to-use customizable visualizations and dashboards



> Why use Tableau?

- Easy to use—no coding involved
- Integrates seamlessly with any data source
- Fast and can handle large datasets

> Tableau Versions

There are two main versions of Tableau

TABLEAU PUBLIC

A free version of Tableau that lets you connect to limited data sources, create visualizations and dashboards, and publish dashboards online

TABLEAU DESKTOP

A paid version of tableau which lets you connect to all types of data sources, allows you to save work locally, and unlimited data sizes

> Getting started with Tableau

When working with Tableau, you will work with Workbooks. Workbooks contain sheets, dashboards, and stories. Similar to Microsoft Excel, a Workbook can contain multiple sheets. A sheet can be any of the following and can be accessed on the bottom left of a workbook

WORKSHEET

A worksheet is a single view in a workbook. You can add shelves, cards, legends, visualizations, and more in a worksheet

DASHBOARD

A collection of multiple worksheets used to display multiple views simultaneously

STORY

A story is a collection of multiple dashboards and/or sheets that describe a data story

> The Anatomy of a Worksheet

When opening a worksheet, you will work with a variety of tools and interfaces

The Sidebar

In the sidebar, you'll find useful panes for working with data

1. **Data:** The data pane on the left-hand side contains all of the fields in the currently selected data source
2. **Analytics:** The analytics pane on the left-hand side lets you add useful insights like trend lines, error bars, and other useful summaries to visualizations

Tableau Data Definitions

When working with data in Tableau, there are multiple definitions to be mindful of

1. **Fields:** Fields are all of the different columns or values in a data source or that are calculated in the workbook. They show up in the data pane and can either be dimension or measure fields
2. **Dimensions:** A dimension is a type of field that contains qualitative values (e.g. locations, names, and departments). Dimensions dictate the amount of granularity in visualizations and help reveal nuanced details in the data
3. **Measures:** A measure is a type of field that contains quantitative values (e.g. revenue, costs, and market sizes). When dragged into a view, this data is aggregated, which is determined by the dimensions in the view
4. **Data types:** Every field has a data type which is determined by the type of information it contains. The available data types in Tableau include text, date values, date & time values, numerical values, boolean values, geographical values, and cluster groups

The Canvas

The canvas is where you'll create data visualizations

1. **Tableau Canvas:** The canvas takes up most of the screen on Tableau and is where you can add visualizations
2. **Rows and columns:** Rows and columns dictate how the data is displayed in the canvas. When dimensions are placed, they create headers for the rows or columns while measures add quantitative values
3. **Marks card:** The marks card allows users to add visual details such as color, size, labels, etc. to rows and columns. This is done by dragging fields from the data pane into the marks card

> Visualizing Your First Dataset

Upload a dataset to Tableau

1. Launch Tableau
2. In the *Connect* section, under *To a File*, press on the file format of your choice
3. For selecting an Excel file, select *.xlsx* or *.xlsx*

Creating your first visualization

1. Once your file is uploaded, open a *Worksheet* and click on the *Data* pane on the left-hand side
2. Drag and drop at least one field into the *Columns* section, and one field into the *Rows* section at the top of the canvas
3. To add more detail, drag and drop a dimension into the *Marks* card (e.g. drag a dimension over the color square in the marks card to color visualization components by that dimension)
4. To a summary insight like a trendline, click on the *Analytics* pane and drag the trend line into your visualization
5. You can change the type of visualization for your data by clicking on the *Show Me* button on the top right

> Data Visualizations in Tableau

Tableau provides a wide range of data visualizations to use. Here is a list of the most useful visualizations you have in Tableau

- **Bar Charts:** Horizontal bars used for comparing specific values across categories *(e.g. sales by region)*
- **Stacked Bar Chart:** Used to show categorical data within a bar chart *(e.g., sales by region and department)*
- **Side-by-Side Bar Chart:** Used to compare values across categories in a bar chart format *(e.g., sales by region comparing product types)*
- **Line Charts:** Used for looking at a numeric value over time *(e.g., revenue over time)*
- **Scatter Plot:** Used to identify patterns between two continuous variables *(e.g., profit vs. sales volume)*
- **Histogram:** Used to show a distribution of data *(e.g., Distribution of monthly revenue)*
- **Box-and-Whisker Plot:** Used to compare distributions between categorical variables *(e.g., distribution of revenue by region)*
- **Heat Map:** Used to visualize data in rows and columns as colors *(e.g., revenue by marketing channel)*
- **Highlight Table:** Used to show data values with conditional color formatting *(e.g., site-traffic by marketing channel and year)*
- **Symbol Map:** Used to show geographical data *(e.g., Market size opportunity by state)*
- **Map:** Used to show geographical data with color formatting *(e.g., Covid cases by state)*
- **Treemap:** Used to show hierarchical data *(e.g., Show how much revenue subdivisions generate relative to the whole department within an organization)*
- **Dual Combination:** Used to show two visualizations within the same visualization *(e.g., profit for a store each month as a bar chart with inventory over time as a line chart)*

> Customizing Visualizations with Tableau

Tableau provides a deep ability to filter, format, aggregate, customize, and highlight specific parts of your data visualizations

Filtering data with highlights

1. Once you've created a visual, click and drag your mouse over the specific portion you want to highlight
2. Once you let go, you will have the option to ✔ *Keep Only* or ✕ *Exclude the data*
3. Open the *Data* pane on the side bar. Then, you can drag-and-drop a field into the fitlers card just to the left of the pane.

Filtering data with filters

1. Open the *Data* pane on the left-hand-side
2. Drag-and-drop a field you want to filter on and add it to the *Filters* card
3. Fill out in the modal how you would like your visuals to be filtered on the data

Aggregating data

When data is dragged into the Rows and Columns on a sheet, it is aggregated based on the dimensions in the sheet. This is typically a summed value. The default aggregation can be changed using the steps below:

1. Right-click on a measure field in the *Data* pane
2. Go down to *Default properties, Aggregation*, and select the aggregation you would like to use

Changing colors

Color is a critical component of visualizations. It draws attention to details. Attention is the most important component of strong storytelling. Colors in a graph can be set using the marks card.

1. Create a visualization by dragging fields into the *Rows* and *Columns* section at the top of the screen
2. Drag dimensions into the *Marks* field, specifically into the *Color* square
3. To change from the default colors, go to the upper-right corner of the color legend and select *Edit Colors*. This will bring up a dialog that allows you to select a different palette

Changing fonts

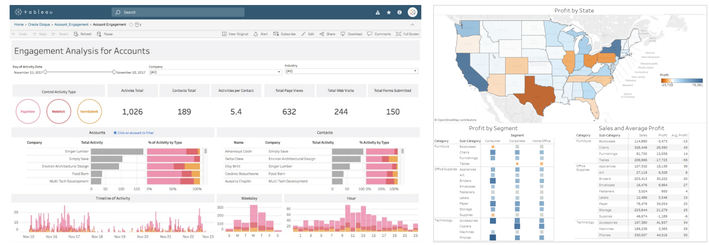
Fonts can help with the aesthetic of the visualization or help with consistent branding. To change the workbook's font, use the following steps

1. In the *Format* menu on the top ribbon, press on *Select Workbook*. This will replace the *Data* pane and allow you to make formatting decisions for the Workbook
2. From here, select the font, font size, and color

> Creating dashboards with Tableau

Dashboards are an excellent way to consolidate visualizations and present data to a variety of stakeholders. Here is a step by step process you can follow to create a dashboard.

1. Launch Tableau
2. In the *Connect* section under *To A File*, press on your desired file type
3. Select your file
4. Click the *New Sheet* at the bottom to create a new sheet
5. Create a visualization in the sheet by following the steps in the previous sections of this cheat sheet
6. Repeat steps 4 and 5 untill you have created all the visualizations you want to include in your dashboard
7. Click the *New Dashboard* at the bottom of the screen
8. On the left-hand side, you will see all your created sheets. Drag sheets into the dashboard
9. Adjust the layout of your sheets by dragging and dropping your visualizations

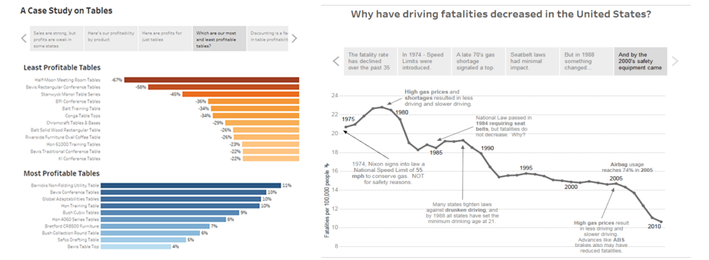


Dashboard examples in Tableau

> Creating stories with Tableau

A story is a collection of multiple dashboards and/or sheets that describe a data story

1. Click the *New Story* at the bottom of the screen
2. Change the size of the story to the desired size in the bottom left-hand corner of the screen under *Size*
3. Edit the title of the story by renaming the story. To do this, right-click on the story sheet at the bottom and press *Rename*
4. A story is made of story points, which lets you cycle through different visualizations and dashboards
5. To begin adding to the story, add a story point from the left-hand side. You can add a blank story point
6. To add a summary text to the story, click *Add a caption* and summarize the story point
7. Add as many story points as you would like to finalize your data story



Stories examples in Tableau

Learn Data Skills Online at www.DataCamp.com

Tableau-Desktop

CHEAT SHEET

Data Sources

File Systems	CSV, Excel, etc.
Relational Systems	Oracle, Sql Server, DB2, etc.
Cloud Systems	Windows Azure, Google BigQuery, etc.
Other Sources	ODBC

Data Extract

- Extraction of data is done by following
  Menu → Data → Extract Data.
- Applying Extract Filters to create subset of data
- To add more data for an already created extract
  Data → Extract → Append Data from File
- Extract History
  Menu - Data → Extract History

Data Joining

- Creating a Join
- Editing a Join Type
- Editing Join Fields

Data Blending

- Preparing Data for Blending
- Adding Secondary Data Source
- Blending the Data

Operators

- General Operators
- Arithmetic Operators
- Relational Operators
- Logical Operators

LOD Expressions

- Fixed LOD , Include LOD and Exclude LOD

Sorting

- **Computed Sorting:** Directly applied on an axis using the sort dialog button.
- **Manual Sorting:** Rearrange the order of dimension fields by dragging them next to each other.

Data Sources

Types	Work
Filter Dimensions	Applied on the dimension fields.
Filter Measures	Applied on the measure fields.
Filter Dates	Applied on the date fields.
Single Value (List)	Select one value at a time in a list.
Single Value (Dropdown)	Select a single value in a drop-down list.

Types	Work
Multiple Values (List)	Select one or more values in a list.
Multiple Values (Dropdown)	Select one or more values in a drop-down list.
Multiple Values (Custom List)	Search and select one or more values.
Single Value (Slider)	Drag a horizontal slider to select a single value.
Wildcard Match	Select values containing the specified characters.

Tableau Charts

Type	Description
Text Table (Crosstab)	To see your data in rows and columns.
Heat Map	Just like Crosstab, but it uses size and color as visual cues to describe the data.
Highlight Table	Just like Excel table, but the cells here are colored.
Symbol Map	Visualize and highlight geographical data.
Filled Map	Color filled geographical data visualization.
Pie Chart	Represents data as slices of a circle with different sizes and colors.
Horizontal Bar Chart	Represents data in horizontal bars, visually digestible.
Stacked Bar Chart	Visualize data of a category having sub-categories.
Side-by-Side Bar Chart	Side by side comparison of data, vertical representation.
Treemap	Similar to a heat map, but the boxes are grouped by items that are close in hierarchy.
Circle View	Shows the different values that are within the categories.
Side-by-Side Circle View fields)	Combination of Circle view and Side-by-Side Bar Chart
Line Chart (Continuous)	Several number of lines in the view to show continuous flow of data, must have a date.
Line Chart (Discrete)	This allows slicing and dicing of the graph, graph not continuous.
Dual Line Chart	Comparing two measures over a period.
Scatter Plot	Scatter plot shows many points scattered in the Cartesian plane
Histogram	A histogram represents the frequencies of values of a variable bucketed into ranges
Gantt Chart	It illustrates a project schedule.
Bullet Graph	Two bars drawn upon one another to indicate their individual values at the same position in the graph
Waterfall Chart	It shows where a value starts, ends and how it gets there incrementally

+ableau

FURTHERMORE:

[Tableau Training and Certification - Tableau 10 Desktop Course]

IntelliPaat

Tableau-Desktop Shortcuts & Terminologies CHEAT SHEET

What is Tableau?

A powerful Data visualization and Business intelligence tool with a strong and intuitive interface. No coding knowledge or experience needed to work with Tableau.

Data Short Cuts:

Data	Operation
ALT+D+A	Automatic updates
ALT+D+D	Connect to data
CTRL+D	Connect to data source
ALT+D+C+D	Duplicate data connection
ALT+D+A	Extract
ALT+D+C+P	Properties of data connection
ALT+D+R	Refresh data
F5	Refreshes the data source
F9	Run query
ALT+D+U	Toggle use extract
F10	Toggles automatic updates on and off
ALT+D+P+D	Update dashboard
ALT+D+P+Q	Update quick filters
ALT+D+P+W	Update worksheet

File Short Cuts:

File	Operation
ALT+F4	Closes the current workbook
ALT+F+E+I	Export to image
ALT+F+E+P	Export to packaged workbook
CTRL+N	New workbook
CTRL+O	Open file
CTRL+P	Print
CTRL+S	Save file

File Short Cuts:

Analysis	Operation
ALT+A+A	Aggregate measures
ALT+A+C	Create calculated field
ALT+A+B	Describe trend model
ALT+A+U	Edit calculated field
ALT+A+L	Edit trend lines
ALT+A+F	Filter
CTRL+1	Show me!
ALT+A+S	Sort
ALT+A+M+F	Stack marks off
ALT+A+M+O	Stack marks on

Get started with Tableau

- Tableau is BI software
- Allows users to connect to data, visualize data and create interactive and sharable dashboards.

Tableau Terminologies

Alias: Refers to a field or to a dimension member

Bin: User-defined grouping of measures in the data source

Bookmark: .tbm file in the Bookmarks folder contains a single worksheet of the Tableau repository

Calculated Field: New field created by using a formula to modify the existing fields in data source

Crosstab: Text table view to display the numbers associated with dimension members

Dashboard: Use dashboards to compare and monitor a variety of data simultaneously

Data Pane: Displays the fields (divided into dimensions and measures) of the data sources to which Tableau is connected

Data Source Page: A page to set up your data source consists of – left pane, join area, preview area, and metadata area

Dimension: Categorical data field holds discrete data such as hierarchies and members that cannot be aggregated

Extract: A saved subset of a data source that can be used to improve performance and analyze offline.

Filters Shelf: Used to exclude data from a view by filtering it using measures and dimensions

Format Pane: Contains formatting settings that control the entire worksheet, as well as individual fields in the view

Level of Detail (LOD) Expression: A syntax that supports aggregation at dimensionalities other than the view level.

Marks: A part of the view that visually represents one or more rows in a data source. A mark can be, for example, a bar, line, or square. You can control the type, color, and size of marks

Marks Card: A card to the left of the view, where you can drag fields to control mark properties such as type, color, size, shape, label, tooltip, and detail

Pages Shelf: Used to split a view into a sequence of pages based on the members and values in a discrete or continuous field

Rows Shelf: Used to create the rows of a data table, also accepts any number of dimensions and measures

Worksheet: A sheet to build views of the data

Workbook: Contains one or more worksheets

Design Flow

Connect to Data Source → Build Data Views → Enhance Data Views → Worksheets → Create and Organize Dashboards → Story Telling

7.2 Power BI

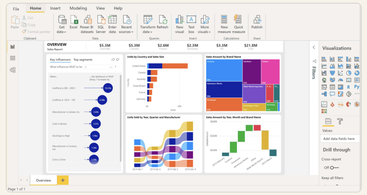
Power BI for Business Intelligence

Power BI Cheat Sheet

Learn Power BI online at www.DataCamp.com

What is Power BI?

Power BI is a business intelligence tool that allows you to effectively report insights through easy-to-use customizable visualizations and dashboards.



> Why use Power BI?

- Easy to use—no coding involved
- Integrates seamlessly with any data source
- Fast and can handle large datasets

> Power BI Components

There are three components to Power BI—each of them serving different purposes

POWER BI DESKTOP
Free desktop application that provides data analysis and creation tools.

POWER BI SERVICE
Cloud-based version of Power BI with report editing and publishing features.

POWER BI MOBILE
A mobile app of Power BI, which allows you to author, view, and share reports on the go.

> Getting started with Power BI

There are three main views in Power BI

REPORT VIEW
This view is the default view, where you can visualize data and create reports

DATA VIEW
This view lets you examine datasets associated with your reports

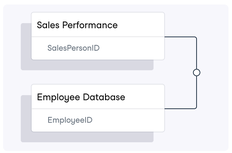
MODEL VIEW
This view helps you establish different relationships between datasets

> Visualizing your first dataset

Upload datasets into Power BI

- Underneath the *Home* tab, click on *Get Data*
- Choose any of your datasets and double click
- Click on *Load* if not prior data needs processing
- If you need to transform the data, click *Transform* which will launch Power Query. Keep reading this cheat sheet for how to apply transformations in Power Query.
- Inspect your data by clicking on the *Data View*

Create relationships in Power BI



- If you have different datasets you want to connect. First, upload them into Power BI
- Click on the *Model View* from the left-hand pane
- Connect key columns from different datasets by dragging one to another (e.g., EmployeeID to e.g., SalespersonID)

Create your first visualization

- Click on the *Report View* and go to the *Visualizations* pane on the right-hand side
- Select the type of visualization you would like to plot your data on. Keep reading this cheat to learn different visualizations available in Power BI.
- Under the *Field* pane on the right-hand side, drag the variables of your choice into *Values* or *Axis*.

Values let you visualize aggregate measures (e.g. Total Revenue)
Axis let you visualize categories (e.g. Sales Person)

Aggregating data

Power BI sums numerical fields when visualizing them under ***Values***. However, you can choose different aggregations

- Select the visualization you just created
- Go to the *Visualizations* section on the right-hand side
- Go to *Values*—the visualized column should be there
- On the selected column—click on the dropdown arrow and change the aggregation (i.e., `AVERAGE`, `MAX`, `COUNT`, etc..)

> Data Visualizations in Power BI

Power BI provides a wide range of data visualizations. Here is a list of the most useful visualizations you have in Power BI

- Bar Charts: Horizontal bars used for comparing specific values across categories (e.g. sales by region)
- Column Charts: Vertical columns for comparing specific values across categories
- Line Charts: Used for looking at a numeric value over time (e.g. revenue over time)
- Area Chart: Based on the line chart with the difference that the area between the axis and line is filled in (e.g. sales by month)
- Scatter: Displays one set of numerical data along the horizontal axis and another set along the vertical axis (e.g. relation age and loan)
- Combo Chart: Combines a column chart and a line chart (e.g. actual sales performance vs target)
- Treemaps: Used to visualize categories with colored rectangles, sized with respect to their value (e.g. product category based on sales)
- Pie Chart: Circle divided into slices representing a category's proportion of the whole (e.g. market share)
- Donut Chart: Similar to pie charts; used to show the proportion of sectors to a whole (e.g. market share)
- Maps: Used to map categorical and quantitative information to spatial locations (e.g. sales per state)
- Cards: Used for displaying a single fact or single data point (e.g. total sales)
- Table: Grid used to display data in a logical series of rows and columns (e.g. all products with sold items)

> Power Query Editor in Power BI

Power Query is Microsoft's data transformation and data preparation engine. It is part of Power BI Desktop, and lets you connect to one or many data sources, shape and transform data to meet your needs, and load it into Power BI.

Open the Power Query Editor

While loading data
- Underneath the *Home* tab, click on *Get Data*
- Choose any of your datasets and double click
- Click on *Transform* Data

When data is already loaded
- Go to the *Data View*
- Under *Queries* in the *Home* tab of the ribbon, click on *Transform Data* drop-down, then on the *Transform Data* button

Using the Power Query Editor

Removing rows
You can remove rows dependent on their location, and properties
- Click on the *Home* tab in the *Query* ribbon
- Click on *Remove Rows* in the *Reduce Rows* group
- Choose which option to remove, whether *Remove Top Rows*, *Remove Bottom Rows*, etc..
- Choose the number of rows to remove
- You can undo your action by removing it from the *Applied Steps* list on the right-hand side

Adding a new column
You can create new columns based on existing or new data
- Click on the *Add Column* tab in the *Query* ribbon
- Click on *Custom Column* in the *General* group
- Name your new column by using the *New Column Name* option
- Define the new column formula under the custom column formula using the available data

Replace values
You can replace one value with another value wherever that value is found in a column
- In the Power Query Editor, select the cell or column you want to replace
- Click on the column or value, and click on *Replace Values* under the *Home* tab under the *Transform* group
- Fill the *Value to Find* and *Replace With fields* to complete your operation

Appending datasets
You can append one dataset to another

- Click on *Append Queries* under the *Home* tab under the *Combine* group
- Select to append either *Two tables* or *Three or more tables*
- Add tables to append under the provided section in the same window

Merge Queries
You can use merge tables based on a related column

- Click on *Merge Queries* under the *Home tab* under the *Combine group*
- Select the first table and the second table you would like to merge
- Select the columns you would like to join the tables on by clicking on the column from the first dataset, and from the second dataset
- Select the *Join Kind* that suits your operation:

- Click on *Ok*—new columns will be added to your current table

Data profiling
Data Profiling is a feature in Power Query that provides intuitive information about your data

- Click on the *View* tab in the *Query* ribbon
- In the *Data Preview* tab—tick the options you want to visualize
- Tick *Column Quality* to see the amount of missing data
- Tick *Column Distribution* to see the statistical distribution under every column
- Tick *Column Profile* to see summary statistics and more detailed frequency information of columns

> DAX Expressions

Data Analysis Expressions (DAX) is a calculation language used in Power BI that lets you create calculations and perform data analysis. It is used to create calculated columns, measures, and custom tables. DAX functions are predefined formulas that perform calculations on specific values called arguments.

Sample data
Throughout this section, we'll use the columns listed in this sample table of `sales_data`

deal_size	sales_person	date	customer_name
1,000	Maria Shuttleworth	30-03-2022	Acme Inc.
3,000	Nuno Rocha	29-03-2022	Spotflix
2,300	Terence Mickey	13-04-2022	DataChamp

Simple aggregations
- `SUM(<column>)` adds all the numbers in a column
- `AVERAGE(<column>)` returns the average (arithmetic mean) of all numbers in a column
- `MEDIAN(<column>)` returns the median of numbers in a column
- `MIN/MAX(<column>)` returns the smallest/biggest value in a column
- `COUNT(<column>)` counts the number of cells in a column that contain non-blank values
- `DISTINCTCOUNT(<column>)` counts the number of distinct values in a column.

EXAMPLES
- Sum of all deals — `SUM('sales_data'[deal_size])`
- Average deal size — `AVERAGE('sales_data'[deal_size])`
- Distinct number of customers — `DISTINCTCOUNT('sales_data'[customer_name])`

Logical functions
- `IF(<logical_test>, <value_if_true>[, <value_if_false>])` check the result of an expression and create conditional results

EXAMPLES
- Create a column called `large_deal` that returns "Yes" if `deal_size` is bigger than 2,000 and "No" otherwise
  `large_deal = IF( 'sales_data'[deal_size] > 2000, "Yes", "No")`

Text Functions
- `LEFT(<text>, <num_chars>)` returns the specified number of characters from the start of a text
- `LOWER(<text>)` converts a text string to all lowercase letters
- `UPPER (<text>)` converts a text string to all uppercase letters
- `REPLACE(<old_text>, <start_num>, <num_chars>, <new_text>)` replaces part of a text string with a different text string.

EXAMPLES
- Change column `customer_name` be only lower case
  `customer_name = LOWER('sales_data'[customer_name])`

Date and time functions
- `CALENDAR(<start date>, <end date>)` generates a column of continuous sets of dates
- `DATE(<year>, <month>, <day>)` returns the specified date in the datetime format
- `WEEKDAY(<date>, <return_type>)` returns 1-7 corresponding to the day of the week of a date (`return_type` indicates week start and end (1: Sunday-Saturday, 2: Monday-Sunday)

EXAMPLES
- Return the day of week of each deal
  `week_day = WEEKDAY('sales_data'[date], 2)`

OVERVIEW

What is **Power BI**?

"It is Microsoft's Self-Service Business Intelligence tool for processing and analyzing data."

Components

› **Power BI Desktop**—Desktop application
› **Report**—Multi-page canvas visible to end users. It serves for the placement of visuals, buttons, images, slicers, etc.
› **Data**—Preview pane for data loaded into a model.
› **Model**—Editable scheme of relationships between tables in a model. Pages can be used in a model for easier navigation.
› **Power Query**—A tool for connecting, transforming, and combining data.

"Apart from the standard version, there is also a version for Report Server."

› **Power BI Service**—A cloud service enabling access to, and sharing and administration of, output data.
› **Workspace**—There are three types of workspaces: **Personal**, **Team**, and **Develop a template app**. They serve as storage and enable controlled access to output data.
› **Dashboard**—A space consisting of **tiles** in which visuals and report pages are stored.*
› **Report**—A **report** of pages containing visuals.*
› **Worksheet**—A published Excel worksheet. Can be used as a tile on a dashboard.
› **Dataset**—A published sequence for fetching and transforming data from **Power BI Desktop**.
› **Dataflow**—Online Power Query representing a special dataset outside of Power BI Desktop.*
› **Application**—A single location combining one or more reports or dashboards.*
› **Admin portal**—Administration portal that lets you configure capacities, permissions, and capabilities for individual users and workspaces.

**Can be created and edited in the Power BI Service environment.*

› **Data Gateway**—On-premises data gateway that lets you transport data from an internal network or a custom device to the **Power BI Service**.
› **Power BI Mobile**—Mobile app for viewing reports. Mobile view is applied, if it exists, otherwise the desktop view is used.
› **Report Server**—On-premises version of Power BI Service.
› **Report Builder**—A tool for creating page reports.

Built-in and additional languages

Built-in languages

› **M/Query Language**—Lets you transform data in Power Query.
› **DAX** (Data Analysis Expressions)—Lets you define custom calculated tables, columns, and measures in Power BI Desktop.

"Both languages are natively available in Power BI, which eliminates the need to install anything."

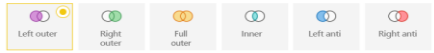
Additional languages

› **Python**—Lets you fetch data and create visuals. Requires installation of the Python language on your computer and enabling Python scripting.
› **R**—Lets you fetch and transform data and create visuals. Requires installation of the R language on your computer and enabling R scripting.

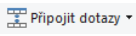
Power Query

Works with data fetched from data sources using connectors. This data is then processed at the Power BI app level and stored to an **in-memory** database in the program background. This means that data is not processed at the source level. The basic unit in Power Query is **query**, which means one sequence consisting of **steps**. A step is a data command that dictates what should happen to the data when it is loaded into Power BI. The basic definition of each step is based on its use:

› **Connecting data**—Each query begins with a function that provides data for the subsequent steps. E.g., data can be loaded from **Excel**, **SQL database**, **SharePoint** etc. Connection steps can also be used later.
› **Transforming data**—Steps that modify the structure of the data. These steps include features such as Pivot Column, converting columns to rows, grouping data, splitting columns, removing columns, etc. Transformation steps are necessary in order to clean data from not entirely clean data sources.
› **Combining data**—Data split into multiple source files needs to be combined so that it can be analyzed in bulk. Functions include merging queries and appending queries.
› **Merge queries**—This function merges queries based on the selected key. The primary query then contains a column which can be used to extract data from a secondary query. Supports typical join types:



› **Append query**—Places the resulting data from one or more selected queries under the primary query. In this case, data is placed in columns with names that are an exact match. Non-matching columns form new columns with a unique name in the primary query.



› **Custom function**—A query intended to apply a pre-defined sequence of steps so that the author does not need to create them repeatedly. The custom function can also accept input data (values, sheets, etc.) to be used in the sequence.
› **Parameter**—Values independent of datasets. These values can then be used in queries. Values enable the quick editing of a model because they can be changed in the Power BI Service environment.

Dataflow

The basic unit is a table or **Entity** consisting of columns or **Fields**. Just like Queries in Power Query, Entities in Dataflows consist of sequences of steps. The result of such steps is stored in native Azure Data Lake Gen 2.

"You can connect a custom Data Lake where the data will be stored."

There are three types of entities:
Standard entity—It only works with data fetched directly from a data source or with data from non-stored entities within the same dataflow.
Computed entity*—It uses data from another stored entity within the same dataflow.
Linked entity*—Uses data from an entity located in another dataflow. If data in the original entity is updated, the new data is directly passed to all linked entities.

**Can only be used in a dedicated Power BI Premium workspace.*

"It supports custom functions as well as parameters."

DAX

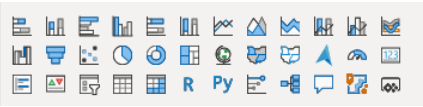
Language developed for data analysis. It enables the creation of the following objects using expressions:
› **Measures**
› **Calculated Columns**
› **Calculated Tables**

Each expression starts with the = sign, followed by links to tables/columns/functions/measures and operators. The following operators are supported:
› **Arithmetic** { + , - , / , * , ^ }
› **Comparison** { = , == , > , < , >= , <= , <> }
› **Text concatenation** { & , && , II , IN }
› **Precedence** { (,) }

Operators and functions require that all values/columns used are of the same data type or of a type that can be freely converted; such as a date or a number.

Visualization

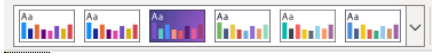
Visualizations or visuals let you present data in various graphical forms, from graphs to tables, maps, and values. Some visuals are linked to other services outside Power BI, such as Power Apps.



In addition to basic visuals, Power BI supports creating custom visuals. Custom visuals can be added using a file import or from a free Marketplace offering certified and non-certified visuals. Certification is optional, but it verifies whether, among other things, a visual accesses external services and resources.

Themes

Serves as a single location for configuring all native graphical settings for visuals and pages.



By default, you can choose from 19 predefined themes. Custom themes can be added.

A custom theme can be applied in two different ways:
› **Modification of an existing theme**—A native window that lets you modify a theme directly in the Power BI environment.
› **Importing a JSON file**—Any file you create only defines the formatting that should change. Everything else remains the same. The advantage of this approach is that you can customize any single visual.

*"The resulting theme **can be exported** in the JSON format and **used** in any report without the need to create a theme from scratch."*

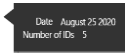
Drill Down

The Visual that supports the embedding of hierarchies enables drilling down to the embedded hierarchy's individual levels using the following symbols:

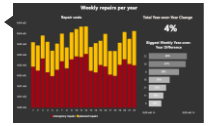
Drill up to a higher-level hierarchy
Drill down to a specific field
Drill down to the next level in the hierarchy
Expand next-level hierarchy

Tooltip/Custom Tooltip

› **Tooltip** —A default detail preview pane which appears above a visual when you hover over its values.

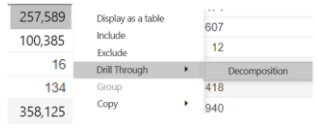


› **Custom Tooltip** —A custom tooltip is a custom-designed report page identified as descriptive. When you hover over visual, a page appears with content filtered based on criteria specified by the value in the visual.



Drill-through

Drill-through lets you pass from a data overview visual to a page with specific details. The target page is displayed with all the applied filters affecting the value from which the drill-through originated.



Bookmarks

Bookmarks capture the currently configured view or a report page visual. Later, you can go back to that state by selecting the saved bookmark. Setting options:
› **Data**—Stores filters, applied sort order in visuals and slicers. By selecting the bookmark, you can re-apply the corresponding settings.
› **Display**—Stores the state of the display for visuals and report elements (buttons, images, etc.). By selecting the bookmark, you can go back to the previously stored state of the display.
› **Current page**—Stores the currently displayed page. By selecting the bookmark, you can go back the to stored page.

License

Per-user License
› **Free**—Can be obtained for any Microsoft work or school email account. Intended for personal use. Users with this license can only use the personal workspace. They cannot share or consume shared content.

"If it is not available in Premium workspace"

› **Pro**—It is associated with a work/school account priced at €8.40 per month or it is included in the E5 license. Intended for team collaboration. Let's users access team workspaces, consume shared content, and use apps.
› **Premium per User** – Includes all Power BI Pro license capabilities, and adds features such as paginated reports, AI, greater frequency for refresh rate, XMLA endpoint and other capabilities that are only available to Premium subscribers.
Per-tenant License
› **Premium**—Premium is set up for individual workspaces. 0 to N workspaces can be used with a single version of this license. It provides dedicated server computing power based on license type: P1, P2, P3, P4*, P5*. It offers more space for datasets, extended metrics for individual workspaces, managed consumption of dedicated capacity, linking of Azure AI features with datasets, and access for users with **Free** licenses to shared content. Prices start at €4,212.30.

**Only available upon special request. Intended for models larger than 100GB.*

› **Embedded**—Supports embedding dashboards and reports in custom apps.
› **Report Server**—Included in Premium or SQL Server Enterprise licenses.

Administration

› **Use metrics**—Usage metrics let you monitor Power BI usage for your organization.
› **Users**—The Users tab provides a link to the Microsoft 365 admin center.
› **Audit logs**—The Audit logs tab provides a link to the Security & Compliance center.
› **Tenant settings**—Tenant settings enable fine-grained control over features made available to your organization. It controls which features will be enabled or disabled and for which users and groups.
› **Capacity settings**—The Power BI Premium tab enables you to manage any Power BI Premium and Embedded capacities.
› **Embed codes**—You can view the embed codes that are generated for your tenant to share reports publicly. You can also revoke or delete codes.
› **Organization visuals**—You can control which type of Power BI visuals users can access across the organization.
› **Azure connections**—You can control workspace-level storage permissions for Azure Data Lake Gen 2.
› **Workspaces**—You can view the workspaces that exist in your tenant on the Workspaces tab.
› **Custom branding**—You can customize the look of Power BI for your whole organization.
› **Protection metrics**—The report shows how sensitivity labels help protect your content.
› **Featured content**—You can manage all the content promoted in the Featured section.

External Tools

They simplify the use of Power BI and extend the capabilities offered in Power BI. These tools are mostly developed by the community. Recommended external tools:
› Tabular Editor
› DAX studio
› ALM Toolkit
› VertiPaq Analyzer

DAX

What is **DAX**?

“ Data Analysis Expressions (DAX) is a library of functions and operators combined to create formulas and expressions “

Introduction to DAX

› **Where to find**
› Power BI, Power Pivot for Excel, Microsoft Analysis Services
› **Purpose**
› DAX was created to enumerate formulas across the data model, where the data is stored in the form of tables, which can be linked together through the sessions. They may have a cardinality of either 1: 1, 1: N, or M: N and your direction, which decides which table filters which. These sessions are either active or inactive. The active session is automatically and participates in the calculation. The inactive is involved in this when it is activated, for example, by a function **USERELATIONSHIP()**

Basic concepts

› **Constructs and their notation**
› Table – ‘Table‘
› Column – [Column] -> ‘Table‘[Column]
› Measure – [NameOfMeasure]
› **Comments**
› Single-line (CTRL + ´) – // or --
› Multi-line – /* */
› **Data types**
› INTEGER
› DECIMAL
› CURRENCY
› DATETIME
› BOOLEAN
› STRING
› VARIANT (not implemented in Power BI)
› BINARY

› DAX can work very well with some types as well combined as if it were the same type. If so, for example, the DATETIME and INTEGER data types are supported operator "+" then it is possible to use them together.

Example: DATETIME ([Date]) + INTEGER (1) = DATETIME ([Date] + 1)

Operators

› **Arithmetic** { + , - , / , * , ^ }
› **Comparative** { = , == , > , < , >= , <= , <> }
› **Joining text** { & }
› **Logic** { && , II , IN, NOT }
› **Prioritization** { (,) }

Calculated Columns

› They behave like any other column in the table. Instead of coming from a data source, they are created through a DAX expression evaluated based on the current context line, and we cannot get values of another row directly.
› **Import mode.** Their evaluation and storage is in progress when processing the model.
› **DirectQuery mode.** They are evaluated at runtime, which may slow down the model.

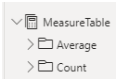
Profit = Trades[Quantity]*Trades[UnitPrice]

Measures

› They do not compare row-based calculations, but they perform aggregation of row-based values input contexts that the environment passes to the calculation. Because of this, there can be no pre-counting result. **It must be evaluated only at the moment when Measure is called.**
› The condition is that they must always be linked to the table to store their code, which is possible at any time alter. Because their calculation is no longer directly dependent, it is common practice to have one separate **Measure Table**, which groups all Measures into myself. For clarity, they are therefore further divided into **folders**.

Example of **Measure**:

SalesVolume = **SUM** (Trades[Quantity])



Variables

› Variables in DAX calculations allow avoiding repeated recalculations of the same procedure. Which might look like this:

```
NumberSort =
VAR _selectedNumber =
  SELECTEDVALUE( Table[Number] )
RETURN
IF( _selectedNumber < 4, _selectedNumber, 5 )
```

› Their declaration uses the word VAR after followed by the **name "="** and the **expression**. The first using the word VAR creates a section for DAX where possible declare such variables 1 to X. Individual variables always require a comment for their declaration VAR before setting the name. To end this section, the word **RETURN** that it defines is a necessary return point for calculations.
› **Variables are local only.**
› If there is a variable in the formula that is not used to get the result, this variable does not evaluate. **(Lazy Evaluation)**
› Evaluation of variables is **performed based on evaluated context instead of the context in which the variable is used directly.** Within one, The expression can be multiple VAR / RETURN sections that always serve to evaluate the currently evaluated context.
› They can store both the value and the whole table

Calculation contexts

› All calculations are evaluated on a base basis some context that the environment brings to the calculation. (**Evaluation context**)
› **Context Filter -**
The following calculation calculates the profit forindividual sales.

```
Revenue =
    SUMX( Trades,
      Trades[Quantity]*
      Trades[UnitPrice]
    )
```

Country	Revenue
Australia	2.838.077.18
Canada	73,959.95
Germany	340.392.76
Japan	1.833.026.16
Mexico	320.833.27
Nigeria	378,202.44
Total	**5,784,491.77**

If I place this calculation in a table without a **Country** column, then the result will be 5,784,491.77. With this column, we get "**Total**" the same as the previous calculation. Still, the individual records provide us with a **FILTER** context that filters in calculating the input the **SUMX** function's input. They behave the same way, for example, **AXES** in the chart.
› The filter context is can be adjusted with various functions, such as **FILTER,ALL, ALLSELECTED**
› **Row context -** Unlike the previous one, this context does not filter the table. It is used to iterate over tables and evaluate values columns. They are typical, but at the same time, specific example calculated columns that are calculated from data that are valid for the table row being evaluated. In particular that, manual creation is not required when creating the line context because **DAX** makes it. Above the mentioned example with the use of **SUMX** also hides in itself line context. Because **SUMX** is the function for that specified, the table in the first argument performs an iterative pass and evaluates the calculation line by line. The line context is possible to use even nested. Or, for each row of the table, evaluates each row of a different table.

Calculate type function

› **CALCULATE,** and **CALCULATETABLE** are functions that can programmatically set the **context filter**. In addition to this feature converts any existing line context to a context filter.
› **Calculate** and **Calculatetable** syntax:

```
CALCULATE / CALCULATETABLE (
                <expression> [, <filter1> [, ... ]]
                )
```

› The section filter within the **Calculate** expression is NOT of type **boolean** but **Table** type. Nevertheless, boolean can be used as an argument.
› Example of using the calculate function in a cumulative calculation the sum of sales for the last 12 months:

```
CALCULATE (
  SUM ( Trades[Quantity] ),
  DATESINPERIOD(
      DateKey[Date],
      MAX ( DateKey[Date] ),
      -1,
      YEAR
  ) )
```

› Syntax Sugar:

```
› [TradeVolume](Trades[Dealer] = 1)
                =
CALCULATE ( [TradeVolume], Trades[Dealer] = 1)
                =
CALCULATE ( [TradeVolume], FILTER (
                ALL (Trades[Dealer]) ,
                Trades[Dealer] = 1) )
```

Calcuation Groups

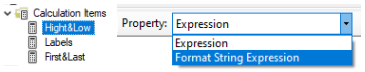
› They are very similar to **Calculated members** from MDX. In Power BI, it is not possible to create them directly in the **Desktop** application environment, but an External Tool **Tabular Editor** is required.
› This is a set of **Calculation Items** grouped according to their purpose and whose purpose is to prepare an expression, which can be used for different input measures, so it doesn’t have to write the same expression multiple times. To where she would be, but the input measure is placed **SELECTEDMEASURE()**.
Example:

```
CALCULATE ( SELECTEDMEASURE(),
                Trades[Dealer] = 1)
```

Name	Ordinal
Hight&Low	0
Labels	1
First&Last	2

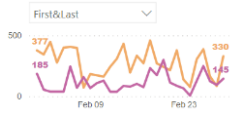
› From a visual point of view, the Calculation Group looks like a table with just two columns, "**Name**," "**Ordinal,**" and rows that indicate the individual Calculation Items.
› In addition to facilitating the reusability of the prepared expressions also provide the ability to modify the output format of individual calculations. Within this section, “**Format String Expression** ”often uses the **DAX** function **SELECTEDMEASUREFORMATSTRING**(), which returns a format string associated with the **Measures** being evaluated.

Example:




```
VAR _selectedCurrency = SELECTEDVALUE( Trades[Currency] )
RETURN
  SELECTEDMEASUREFORMATSTRING() & „ “ & _selectedCurrency
```

› In Power BI, they can all be evaluated pre-prepared items, or it is possible, for example, to use the cross-section to define items that are currently being evaluated
› Sometimes, however, it is necessary to enable the evaluation of Calculation Items only for Specific Measures. In that case, it is possible to use the **ISSELECTEDMEASURE**() function, whose output is a value of type boolean or the **SELECTEDMEASURENAME**() function that returns the name of the currently inserted measure as a string.



Conditions

› Like most languages, DAX uses the IF function. Within this language, it is defined by syntax:

```
IF ( <logical_test>, <value_if_true>[, <value_if_false>])
```

Where false, the branch is optional. The IF function explicitly evaluates only a branch that is based on the result of a logical test relevant.
› If both branches need to be evaluated, then there is a function **IF.EAGER()** whose syntax is the same as IF itself but evaluates as:

```
VAR _value_if_true = <value_if_true>
VAR _value_if_false = <value_if_false>
RETURN
IF (<logical_test>, _value_if_true, _value_if_false)
```

› **IF** has an alternative as **IFERROR**. Evaluates the expression and return the output from the <value_if_error> branch only if the expression returns an error. Otherwise, it returns the value of the expression itself.

› **DAX** supports concatenation of conditions, both using submerged ones **IF**, so thanks to the **SWITCH** function. It evaluates the expression against the list values and returns one of several possible result expressions.

Hierarchy

› DAX itself has no capability within the hierarchy to automatically convert your calculations to parent or child levels. Therefore, each level must Prepare Your Measures, which are then displayed based on the **ISINSCOPE** function. She tests which level to go just evaluating. Evaluation takes place from the bottom to the top level.
› The native data model used by DAX does not directly support its **parent/child** hierarchy. On the other hand, DAX contains functions that can convert this hierarchy to separate columns.
› **PATH** - It accepts two parameters, where the first parameter is the key ID column tables. The second parameter is the column that holds the parent ID of the row. The result of this function then looks like this: **1|2|3|4**
Syntax: **PATH**(<ID_columnName>, <parent_columnName>)
› **PATHITEM** – Returns a specific item based on the specified position from the string, resulting from the PATH function. Positions are counted from left to right. The inverted view uses the PATHITEMREVERSE function.
Syntax: **PATHITEM**(<path>, <position>[, <type>])
› **PATHLENGTH** – Returns the number of parent elements to the specified item in given the **PATH** result, including itself.
Syntax: **PATHLENGTH**(<path>)
› **PATHCONTAINS** – Returns **true** if the specified item is specified exists in the specified PATH path.
Syntax: **PATHCONTAINS**(<path>, <item>)

DAX Queries

› The basic building block of DAX queries is the expression EVALUATE followed by any expression whose output is a table.
Example:

```
EVALUATE
ALL (Trades[Dealer] )
```

› The **EVALUATE** statement can be divided into three primary sections. Each section has its specific purpose and its introductory word.
› **Definition –** It always starts with the word **DEFINE**. This section defines local entities such as tables, columns, variables, and measures. There can be one section definition for an entire query, although a query can contain multiple EVALUATEs
› **Query –** It always starts with the word **EVALUATE**. This section contains the table expression to evaluate and return as a result.
› **Result –** This is a section that is optional and starts with the word ORDER BY. It contains the possibility to sort the result based on the inserted inputs.
Example:

```
DEFINE
    VAR _tax = 0.79
EVALUATE
    ADDCOLUMNS(
    Trades,
    „AdjustedpProfit“,
        ( Trades[Quantity] * Trades[UnitPrice] ) * _tax
    )
ORDER BY [AdjustedpProfit]
```

› This type of notation is used, for example, in DAX Studio (**daxstudio.org**). It is a publicly available tool that provides free access to query validation, code debugging, and query performance measurement.
› **DAX studio** has the ability to connect directly to **Analysis Services**, **Power BI** a **Power Pivot for Excel**

Recommended sources

› **Marco Russo & Alberto Ferrari**
› Daxpatterns.com
› dax.guide
› The Definitive Guide to DAX

POWER QUERY

What is **Power Query**?

"An IDE for M development"

Components

› **Ribbon –** A ribbon containing settings and pre-built features by Power Query itself rewrites in M language for user convenience.
› **Queries –** simply a named M expression. Queries can be moved into groups
› **Primitive –** A primitive value is a single-part value, such as a number, logical, date, text, or null. A **null** value can be used to indicate the absence of any data.
› **List –** The list is an ordered sequence of values. M supports endless lists. Lists define the characters **"{"** and **"}"** indicate the beginning and the end of the list.
› **Record –** A record is a set of fields, where the field is a pair of which form the name and value. The name is a text value that is in the field record unique.
› **Table –** A table is a set of values arranged in named columns and rows.
Table can be operated on as if it is a list of records, or as if it is a record of lists. Table[Field]` (field reference syntax for records) returns a list of values in that field. `Table{i}` (list index access syntax) returns a record representing a row of the table.
› **Function –** A function is a value that when called using arguments creates a new value. Functions are written by listing the function argumets in parentheses, followed by the transition symbol "=>" and the expression defining the function. This expression usually refers to argumets by name. There are also functions without argumets.
› **Parameter –** The parameter stores a value that can be used for transformations. In addition to the name of the parameter and the value it stores, it also has other properties that provide metadata. The undeniable advantage of the parameter is that it can be changed from the **Power BI Service** environment without the need for direct intervention in the data set. Syntax of parameter is as regular query only thing that is special is that the metadata follows a specific format.
› **Formula Bar –** Displays the currently loaded step and allows you to edit it.To be able to see formula bar, It has to be enabled in the ribbon menu inside **View** category.
› **Query settings –** Settings that include the ability to edit the name and description of the query. It also contains an overview of all currently applied steps. Applied Steps are the variables defined in a let expression and they are represented by varaibles names.
› **Data preview –** A component that displays a preview of the data in the currently selected transformation step.
› **Status bar –** This is the bar located at the bottom of the screen. The row contains information about the approximate state of the rows, columns, and time the data was last reviewed. In addition to this information, there is profiling source information for the columns. Here it is possible to switch the profiling from 1000 rows to the entire data set.

Functions in Power Query

Knowledge of functions is your best helper when working with a functional language such as **M**. Functions are called with parentheses.
› **Shared –** Is a keyword that loads all functions (including help and example) and enumerators in result set. The call of function is made inside empty query using by **= # shared**

```
= #shared
```

Functions can be divided into two categories:
› Prefabricated – Example: Date.From()
› Custom – these are functions that the user himself prepares for the model by means of the extension of the notation by „()=> ", where the argumets that will be required for the evaluation of the function can be placed in parentheses. When using multiple argumets, it is necessary to separate them using a delimiter.

Data values

Each value type is associated with a literal syntax, a set of values of that type, a set of operators defined above that set of values, and an internal type attributed to the newly created values.
› **Null –** null
› **Logical –** true, false
› **Number –** 1, 2, 3, ...
› **Time –** #time(HH,MM,SS)
› **Date –** #date(yyyy,mm,ss)
› **DateTime –** #datetime(yyyy,mm,dd,HH,MM,SS)
› **DateTimeZone –** #datetimezone(yyyy,mm,dd,HH,MM,SS, 9,00)
› **Duration –** #duration(DD,HH,MM,SS)
› **Text –** "text"
› **Binary –** #binary("link")
› **List –** {1, 2, 3}
› **Record –** [A = 1, B = 2]
› **Table –** #table({columns},{{first row contenct},{}...})*
› **Function –** (x) => x + 1
› **Type –** type { number }, type table [A = any, B = text]
* The index of the first row of the table is the same as for the records in sheet 0

Operators

There are several operators within the M language, but not every operator can be used for all types of values.
› **Primary operators**
› **(x)** – Parenthesized expression
› **x[i]** – Field Reference. Return value from record, list of values from table.
› **x{i}** – Item access. Return value from list, record from table. "Placing the "**?**" Character after the operator returns null if the index is not in the list "
› **x(...)** – Function invocation
› **{1 .. 10}** – Automatic list creation from 1 to 10
› **...** – Not implemented
› **Mathematical operators** – **+, -, *, /**
› **Comparative operators**
› **>** , **>=** – Greater than, greater than or equal to
› **<** , **<=** – Less than, less than or equal to
› **=** , **<>** – is equal, is not equal. Equal returns true even for null = null
› **Logical operators**
› **and** – short-circuiting conjunction
› **or** – short-circuiting disjunction
› **not** – logical negation
› **Type operators**
› **as** – Is compatible nullable-primitive type or error
› **is** – Test if compatible nullable-primitive type
› **Metadata** - The word **meta** assigns metadata to a value.
Example of assigning metadata to variable **x:**
"x meta y" or **"x meta [name = x, value = 123,...]"**
Within Power Query, the priority of the operators applies, so for example "X + Y * Z" will be evaluated as "X + (Y * Z)"

Comments

M language supports **two** versions of comments:
› Single-line comments – can be created by **//** before code
› Shortcut: **CTRL + ´**
› Multi-line comments – can be created by **/*** before code and ***/** after code
› Shortcut: **ALT + SHIFT + A**

let expression

The expression let is used to capture the value from an intermediate calculation in a named variable. These named variables are local in scope to the `let` expression. The construction of the term let looks like this:

```
let
    name_of_variable = <expression>,
   returnVariable = <function>(name_of_variable)
in
returnVariable
```

When it is evaluated, the following always applies:

› Expressions in variables define a new range containing identifiers from the production of the list of variables and must be present when evaluating terms within a list variables. The expressions in the list of variables are they can refer to each other
› All variables must be evaluated before the term let is evaluated.
› If expressions in variables are not available, let will not be evaluated
› Errors that occur during query evaluation propagate as an error to other linked queries.

Conditions

Even in Power Query, there is an "**If**" expression, which, based on the inserted condition, decides whether the result will be a true-expression or a false-expression.

Syntactic form of If expression:
if <predicate> **then** < true-expression > **else** < false-expression >
"**else** is required in M's conditional expression "

Condition entry:
If x > 2 **then** 1 **else** 0
If [Month] > [Fiscal_Month] **then** true **else** false

If expression is the only conditional in M. If you have multiple predicates to test, you must chain together like:

```
if <predicate>
  then < true-expression >
  else if <predicate>
    then < false-true-expression >
    else  < false-false-expression >
```

When evaluating the conditions, the following applies:

› If the value created by evaluating the if a condition is not a logical value, then an error with the reason code "**Expression.Error**„ is raised
› A true-expression is evaluated only if the **if** condition evaluates to **true. Otherwise**, false-expression is evaluated**.**
› If expressions in variables are not available, they must not be evaluated
› The error that occurred during the evaluation of the condition will spread further either in the form of a failure of the entire query or "**Error**" value in the record.

The expression try... otherwise

Capturing errors is possible, for example, using the **try** expression. An attempt is made to evaluate the expression after the word **try**. If an error occurs during the evaluation, the expression after the word **otherwise** is applied

Syntax example:
try Date.From([textDate]) **otherwise** null

Custom function

Example of custom function entries:
(x, y) => Number.From(**x**) + Number.From(**y**)

```
(x) =>
let
   out = Number.From(x) +
   Number.From(Date.From(DateTime.LocalNow()))
in
   out
```

The input argumets to the functions are of two types:
› **Required –** All commonly written argumets in (). Without these argumets, the function cannot be called.
› **Optional –** Such a parameter may or may not be to function to enter. Mark the parameter as **optional** by placing text before the argument name "**Optional**". For example **(optional x)**. If it does not happen fulfillment of an optional argument, so be the same for for calculation purposes, but its value will be null.
Optional arguments must come after required arguments.

Arguments can be annotated with `as <type>` to indicate required type of the argument. The function will throw a type error if called with arguments of the wrong type. Functions can also have annotated return of them. This annotation is provided as:
(x as number, y as text) as logical => <expression>

The return of the functions is very different. The output can be a sheet, a table, one value but also other functions. This means that one function can produce another function. Such a function is written as follows:

let first = **(x)=> () =>** let out = {1..x} in out in first

When evaluating functions, it holds that:

› Errors caused by evaluating expressions in a list of expressions or in a function expression will propagate further either as a failure or as an "**Error**" value
› The number of arguments created from the argument list must be compatible with the formal argumets of the function, otherwise an error will occur with reason code "**Expression.Error**"

Recursive functions

For recursive functions is necessary to use the character "**@**" which refers to the function within its calculation. A typical recursive function is the factorial. The function for the factorial can be written as follows:

```
let
  Factorial = (x) =>
    if x = 0 then 1 else x * @Factorial(x - 1),
  Result = Factorial(3)
in
  Result // = 6
```

Each

Functions can be called against specific arguments. However, if the function needs to be executed for each record, an entire sheet, or an entire column in a table, it is necessary to append the word **each** to the code. As the name implies, for each context record, it applies the procedure behind it. **Each** is never required! It simply makes it easier to define a function in-line for functions which require a function as their argument.

Syntax Sugar

› Each is essentially a syntactic abbreviation for declaring non-type functions, using a single formal parameter named. Therefore, the following notations are semantically equivalent:

```
let
  Source = ...,
  addColumn = Table.AddColumn(Source, „NewName", each [field1] + 1)
in
  addColumn
```

```
let
  Source = ...,
  add1ToField1 = (_) => [field1] + 1,
  addColumn(Source,"NewName",add1ToField1)
in
```

The second piece of syntax sugar is that bare square brackets are syntax sugar for field access of a Record named `_`.

Query Folding

As the name implies, it is about composing. Specifically, the steps in Power Query are composed into a single query, which is then implemented against the data source. Data sources that supports Query folding are resources that support the concept of query languages as relational database sources. This means that, for example, a CSV or XML file as a flat file with data will definitely not be supported by Query Folding. Therefore, the transformation does not have to take place until after the data is loaded, but it is possible to get the data ready immediately. Unfortunately, not every source supports this feature.
› Valid functions
› Remove, Rename columns
› Row filtering
› Grouping, summarizing, pivot and unpivot
› Merge and extract data from queries
› Connect queries based on the same data source
› Add custom columns with simple logic
› Invalid functions
› Merge queries based on different data sources
› Adding columns with Index
› Change the data type of a column

DEMO

› Operators can be combined. For example, as follows:
› LastStep[Year]{[ID]}

*This means that you can get the value from another step based on the index of the column

› Production of a DateKey dimension goes like this:

```
#table(
   type table [Date=date, Day=Int64.Type, Month=Int64.Type,
MonthName=text, Year=Int64.Type,Quarter=Int64.Type],
   List.Transform(
      List.Dates(start_date, (start_date-endd_ate),
                  #duration(1, 0, 0 ,0)),
      each {_, Date.Day(_), Date.Month(_),
Date.MonthName(_), Date.Year(_), Date.QuarterOfYear(_)}
))
```

Keywords

and, as, each, else, error, false, if, in, is, let, meta, not, otherwise, or, section, shared, then, true, try, type, #binary, #date, #datetime, #datetimezone, #duration, #infinity, #nan, #sections, #shared, #table, #time

7.3 Data Visualization

datacamp

The Data Visualization Cheat Sheet

Learn Data Visualization online at www.DataCamp.com

How to use this cheat sheet

Use this cheat sheet for inspiration when making your next data visualizations. For more data visualization cheat sheets, check out our cheat sheets repository here.

> Part-to-whole charts

Pie chart

One of the most common ways to show part to whole data. It is also commonly used with percentages

USE CASES

1. Voting preference by age group
2. Market share of cloud providers

Donut pie chart

The donut pie chart is a variant of the pie chart, the difference being it has a hole in the center for readability

USE CASES

1. Android OS market share
2. Monthly sales by channel

Heat maps

Heatmaps are two-dimensional charts that use color shading to represent data trends.

USE CASES

1. Average monthly temperatures across the year
2. Departments with the highest amount of attrition over time

Stacked column chart

Best to compare subcategories within categorical data. Can also be used to compare percentages

USE CASES

1. Quarterly sales per region
2. Total car sales by producer

Treemap charts

2D rectangles whose size is proportional to the value being measured and can be used to display hierarchically structured data

USE CASES

1. Grocery sales count with categories
2. Stock price comparison by industry and company

> Capture a trend

Line chart

The most straightforward way to capture how a numeric variable is changing over time

USE CASES

1. Revenue in $ over time
2. Energy consumption in kWh over time
3. Google searches over time

Multi-line chart

Captures multiple numeric variables over time. It can include multiple axes allowing comparison of different units and scale ranges

USE CASES

1. Apple vs Amazon stocks over time
2. Lebron vs Steph Curry searches over time
3. Bitcoin vs Ethereum price over time

Area chart

Shows how a numeric value progresses by shading the area between line and the x-axis

USE CASES

1. Total sales over time
2. Active users over time

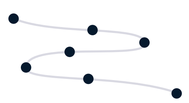
Stacked area chart

Most commonly used variation of area charts, the best use is to track the breakdown of a numeric value by subgroups

USE CASES

1. Active users over time by segment
2. Total revenue over time by country

Spline chart



Smoothened version of a line chart. It differs in that data points are connected with smoothed curves to account for missing values, as opposed to straight lines

USE CASES

1. Electricity consumption over time
2. CO2 emissions over time

> Visualize a single value

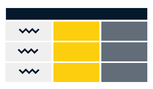
Card

Cards are great for showing and tracking KPIs in dashboards or presentations

USE CASES

1. Revenue to date on a sales dashboard
2. Total sign-ups after a promotion

Table chart



Best to be used on small datasets, it displays tabular data in a table

USE CASES

1. Account executive leaderboards
2. Registrations per webinar

Gauge chart

This chart is often used in executive dashboard reports to show relevant KPIs

USE CASES

1. NPS scores
2. Revenue to target

> Capture distributions

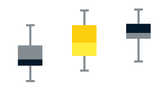
Histogram

Shows the distribution of a variable. It converts numerical data into bins as columns. The x-axis shows the range, and the y-axis represents the frequency

USE CASES

1. Distribution of salaries in an organization
2. Distribution of height in one cohort

Box plot



Shows the distribution of a variable using 5 key summary statistics—minimum, first quartile, median, third quartile, and maximum

USE CASES

1. Gas efficiency of vehicles
2. Time spent reading across readers

Violin plot

A variation of the box plot. It also shows the full distribution of the data alongside summary statistics

USE CASES

1. Time spent in restaurants across age groups
2. Length of pill effects by dose

Density plot

Visualizes a distribution by using smoothing to allow smoother distributions and better capture the distribution shape of the data

USE CASES

1. Distribution of price of hotel listings
2. Comparing NPS scores by customer segment

> Visualize relationships

Bar chart

One of the easiest charts to read which helps in quick comparison of categorical data. One axis contains categories and the other axis represents values

USE CASES

1. Volume of google searches by region
2. Market share in revenue by product

Column chart

Also known as a vertical bar chart, where the categories are placed on the x-axis. These are preferred over bar charts for short labels, date ranges, or negatives in values

USE CASES

1. Brand market share
2. Profit Analysis by region

Scatter plot

Most commonly used chart when observing the relationship between two variables. It is especially useful for quickly surfacing potential correlations between data points

USE CASES

1. Display the relationship between time-on-platform and churn
2. Display the relationship between salary and years spent at company

Connected scatterplot

A hybrid between a scatter plot and a line plot, the scatter dots are connected with a line

USE CASES

1. Cryptocurrency price index
2. Visualizing timelines and events when analyzing two variables

Bubble chart

Often used to visualize data points with 3 dimensions, namely visualized on the x-axis, y-axis, and with the size of the bubble. It tries to show relations between data points using location and size

USE CASES

1. Adwords analysis: CPC vs Conversions vs Share of total conversions
2. Relationship between life expectancy, GDP per capita, & population size

Word cloud chart

A convenient visualization for visualizing the most prevalent words that appear in a text

USE CASES

1. Top 100 used words by customers in customer service tickets

> Visualize a flow

Sankey chart

Useful for representing flows in systems. This flow can be any measurable quantity

USE CASES

1. Energy flow between countries
2. Supply chain volumes between warehouses

Chord chart

Useful for presenting weighted relationships or flows between nodes. Especially useful for highlighting the dominant or important flows

USE CASES

1. Export between countries to showcase biggest export partners
2. Supply chain volumes between the largest warehouses

Network chart

Similar to a graph, it consists of nodes and interconnected edges. It illustrates how different items have relationships with each other

USE CASES

1. How different airports are connected worldwide
2. Social media friend group analysis

Visual vocabulary

Designing with data

There are so many ways to visualise data - how do we know which one to pick? Use the categories across the top to decide which data relationship is most important in your story, then look at the different types of chart within the category to form some initial ideas about what might work best. This list is not meant to be exhaustive, nor a wizard, but is a useful starting point for making informative and meaningful data visualisations.

FT graphic: Alan Smith; Chris Campbell; Ian Bott; Liz Faunce;
Graham Parrish; Billy Ehrenberg; Paul McCallum; Martin Stabe
Inspired by the Graphic Continuum by Jon Schwabish and Severino Ribecca

ft.com/vocabulary

Deviation

Emphasise variations (+/-) from a fixed reference point. Typically the reference point is zero but it can also be a target or a long-term average. Can also be used to show sentiment (positive/neutral/negative).

Example FT uses
Trade surplus/deficit, climate change

Diverging bar

A simple standard bar chart that can handle both negative and positive magnitude values.

Diverging stacked bar

Perfect for presenting survey results which involve sentiment (eg disagree/neutral/agree).

Spine chart

Splits a single value into 2 contrasting components (eg Male/Female).

Surplus/deficit filled line

The shaded area of these charts allows a balance to be shown – either against a baseline or between two series.

Correlation

Show the relationship between two or more variables. Be mindful that, unless you tell them otherwise, many readers will assume the relationships you show them to be causal (i.e. one causes the other).

Example FT uses
Inflation & unemployment, income & life expectancy

Scatterplot

The standard way to show the relationship between two continuous variables, each of which has its own axis.

Line + Column

A good way of showing the relationship between an amount (columns) and a rate (line).

Connected scatterplot

Usually used to show how the relationship between 2 variables has changed over time.

Bubble

Like a scatterplot, but adds additional detail by sizing the circles according to a third variable.

XY heatmap

A good way of showing the patterns between 2 categories of data, less good at showing fine differences in amounts.

Ranking

Use where an item's position in an ordered list is more important than its absolute or relative value. Don't be afraid to highlight the points of interest.

Example FT uses
Wealth, deprivation, league tables, constituency election results

Ordered bar

Standard bar charts display the ranks of values much more easily when sorted into order.

Ordered column

See above.

Ordered proportional symbol

Use when there are big variations between values and/or seeing fine differences between data is not so important.

Dot strip plot

Dots placed in order on a strip are a space-efficient method of laying out ranks across multiple categories.

Slope

Perfect for showing how ranks have changed over time or vary between categories.

Lollipop chart

Lollipops draw more attention to the data value than standard bar/column and can also show rank and value effectively.

Distribution

Show values in a dataset and how often they occur. The shape (or 'skew') of a distribution can be a memorable way of highlighting the lack of uniformity or equality in the data.

Example FT uses
Income distribution, population (age/sex) distribution

Histogram

The standard way to show a statistical distribution - keep the gaps between columns small to highlight the 'shape' of the data.

Boxplot

Summarise multiple distributions by showing the median (centre) and range of the data

Violin plot

Similar to a box plot but more effective with complex distributions (data that cannot be summarised with simple average).

Population pyramid

A standard way for showing the age and sex breakdown of a population distribution; effectively, back to back histograms.

Dot strip plot

Good for showing individual values in a distribution, can be a problem when too many dots have the same value.

Dot plot

A simple way of showing the change or range (min/max) of data across multiple categories.

Barcode plot

Like dot strip plots, good for displaying all the data in a table,they work best when highlighting individual values.

Cumulative curve

A good way of showing how unequal a distribution is: y axis is always cumulative frequency, x axis is always a measure.

Change over Time

Give emphasis to changing trends. These can be short (intra-day) movements or extended series traversing decades or centuries: Choosing the correct time period is important to provide suitable context for the reader.

Example FT uses
Share price movements, economic time series

Line

The standard way to show a changing time series. If data are irregular, consider markers to represent data points.

Column

Columns work well for showing change over time - but usually best with only one series of data at a time.

Line + column

A good way of showing the relationship over time between an amount (columns) and a rate (line).

Stock price

Usually focused on day-to-day activity, these charts show opening/closing and hi/low points of each day.

Slope

Good for showing changing data as long as the data can be simplified into 2 or 3 points without missing a key part of story.

Area chart

Use with care – these are good at showing changes to total, but seeing change in components can be very difficult.

Fan chart (projections)

Use to show the uncertainty in future projections - usually this grows the further forward to projection.

Connected scatterplot

A good way of showing changing data for two variables whenever there is a relatively clear pattern of progression.

Calendar heatmap

A great way of showing temporal patterns (daily, weekly, monthly) – at the expense of showing precision in quantity.

Priestley timeline

Great when date and duration are key elements of the story in the data.

Circle timeline

Good for showing discrete values of varying size across multiple categories (eg earthquakes by continent).

Seismogram

Another alternative to the circle timeline for showing series where there are big variations in the data.

Part-to-whole

Show how a single entity can be broken down into its component elements. If the reader's interest is solely in the size of the components, consider a magnitude-type chart instead.

Example FT uses
Fiscal budgets, company structures, national election results

Stacked column

A simple way of showing part-to-whole relationships but can be difficult to read with more than a few components.

Proportional stacked bar

A good way of showing the size and proportion of data at the same time – as long as the data are not too complicated.

Pie

A common way of showing part-to-whole data – but be aware that it's difficult to accurately compare the size of the segments.

Donut

Similar to a pie chart – but the centre can be a good way of making space to include more information about the data (eg. total).

Treemap

Use for hierarchical part-to-whole relationships; can be difficult to read when there are many small segments.

Voronoi

A way of turning points into areas – any point within each area is closer to the central point than any other centroid.

Sunburst

Another way of visualisaing hierarchical part-to-whole relationships. Use sparingly (if at all) for obvious reasons.

Arc

A hemicycle, often used for visualising political results in parliaments.

Gridplot

Good for showing % information, they work best when used on whole numbers and work well in multiple layout form.

Venn

Generally only used for schematic representation.

Waterfall

Can be useful for showing part-to-whole relationships where some of the components are negative.

Magnitude

Show size comparisons. These can be relative (just being able to see larger/bigger) or absolute (need to see fine differences). Usually these show a 'counted' number (for example, barrels, dollars or people) rather than a calculated rate or per cent.

Example FT uses
Commodity production, market capitalisation

Column

The standard way to compare the size of things. Must always start at 0 on the axis.

Bar

See above. Good when the data are not time series and labels have long category names.

Paired column

As per standard column but allows for multiple series. Can become tricky to read with more than 2 series.

Paired bar

See above.

Proportional stacked bar

A good way of showing the size and proportion of data at the same time – as long as the data are not too complicated.

Proportional symbol

Use when there are big variations between values and/or seeing fine differences between data is not so important.

Isotype (pictogram)

Excellent solution in some instances – use only with whole numbers (do not slice off an arm to represent a decimal).

Lollipop chart

Lollipop charts draw more attention to the data value than standard bar/column – does not HAVE to start at zero (but preferable).

Radar chart

A space-efficient way of showing value pf multiple variables– but make sure they are organised in a way that makes sense to reader.

Parallel coordinates

An alternative to radar charts – again, the arrngement of the variables is important. Usually benefits from highlighting values.

Spatial

Used only when precise locations or geographical patterns in data are more important to the reader than anything else.

Example FT uses
Locator maps, population density, natural resource locations, natural disaster risk/impact, catchment areas, variation in election results

Basic choropleth (rate/ratio)

The standard approach for putting data on a map – should always be rates rather than totals and use a sensible base geography.

Proportional symbol (count/magnitde)

Use for totals rather than rates – be wary that small differences in data will be hard to see.

Flow map

For showing unambiguous movement across a map.

Contour map

For showing areas of equal value on a map. Can use deviation colour schemes for showing +/- values

Equalised cartogram

Converting each unit on a map to a regular and equally-sized shape – good for representing voting regions with equal value.

Scaled cartogram (value)

Stretching and shrinking a map so that each area is sized according to a particular value.

Dot density

Used to show the location of individual events/locations – make sure to annotate any patterns the reader should see.

Heat map

Grid-based data values mapped with an intensity colour scale. As choropleth map – but not snapped to an admin/political unit.

Flow

Show the reader volumes or intensity of movement between two or more states or conditions. These might be logical sequences or geographical locations.

Example FT uses
Movement of funds, trade, migrants, lawsuits, information; relationship graphs.

Sankey

Shows changes in flows from one condition to at least one other; good for tracing the eventual outcome of a complex process.

Waterfall

Designed to show the sequencing of data through a flow process, typically budgets. Can include +/- components.

Chord

A complex but powerful diagram which can illustrate 2-way flows (and net winner) in a matrix.

Network

Used for showing the strength and inter-connectdness of relationships of varying types.

8. Emerging Technologies

Top 10 Technology Trends

Artificial Intelligence (AI)

Simulation of human intelligence processes by machines, which includes machine, deep, and reinforcement learning, natural language processing and computer vision

Internet of Things (IoT)

Networking capability that allows information to be sent to and received from objects and devices (things).

Robotic Process Automation (RPA)

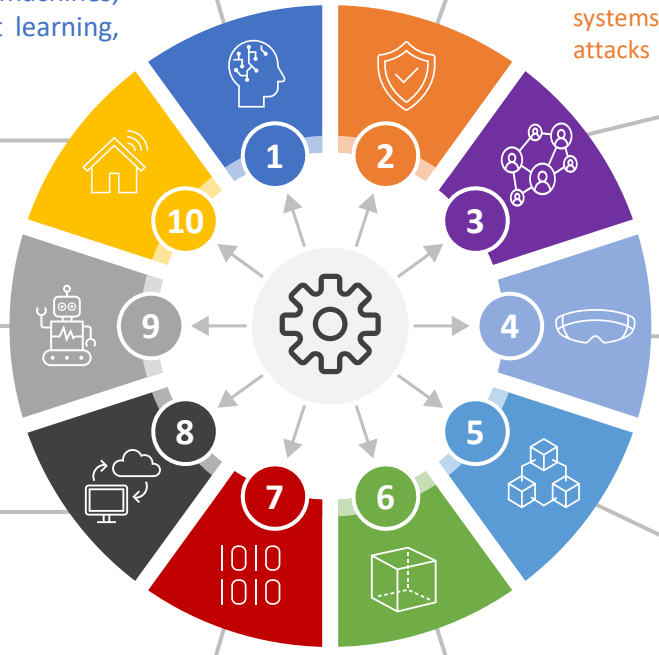
Technology that mimics the way humans interact with software to perform high-volume, repeatable tasks

Cloud Computing

Delivery of different services through the Internet, including data storage, servers, databases, networking, and software.

Quantum Computing

Machines that use the properties of quantum physics to store data and perform computations, which can result in a computer 158 million times faster than the most sophisticated supercomputer we have in the world today



Cyber Security

Application of technologies, processes and controls to protect systems, networks, programs, devices and data from cyber attacks

Blockchain

Technology that facilitates the process of recording transactions and tracking assets in a business network, where an asset can be tangible (house, car, land) or intangible (patents, copyrights, branding)

Extended Reality (XR)

Technology that represents all human-machine interactions, which includes augmented reality (AR), mixed reality (MR) and virtual reality (VR)

5th Generation Network (5G)

5G is the fifth generation of wireless technology, which can provide higher speed, lower latency and greater capacity than 4G LTE networks

3D Printing

Also known as additive manufacturing, is a method of creating a three dimensional object layer-by-layer using a computer created design.

Prepered by: Dr. Mejdal

LinkedIn

8.1 Artificial Intelligence (AI)

Machine Learning Algorithm Cheat Sheet

This cheat sheet helps you choose the best machine learning algorithm for your predictive analytics solution.
Your decision is driven by both the nature of your data and the goal you want to achieve with your data.

Microsoft Azure

What do you want to do?

Extract information from text

Text Analytics

Derives high-quality information from text

Answers questions like: What info is in this text?

- **Latent Dirichlet Allocation** ← Unsupervised topic modeling, group texts that are similar
- **Extract N-Gram Features from Text** ← Creates a dictionary of n-grams from a column of free text
- **Feature Hashing** ← Converts text data to integer encoded features using the Vowpal Wabbit library
- **Preprocess Text** ← Performs cleaning operations on text, like removal of stop-words, case normalization
- **Word2Vector** ← Converts words to values for use in NLP tasks, like recommender, named entity recognition, machine translation

Predict between several categories

Multiclass Classification

Answers complex questions with multiple possible answers

Answers questions like: Is this A or B or C or D?

- **Multiclass Logistic Regression** ← Fast training times, linear model
- **Multiclass Neural Network** ← Accuracy, long training times
- **Multiclass Decision Forest** ← Accuracy, fast training times
- **One-vs-All Multiclass** ← Depends on the two-class classifier
- **One-vs-One Multiclass** ← Depends on binary classifier, less sensitive to an imbalanced dataset with larger complexity
- **Multiclass Boosted Decision Tree** ← Non-parametric, fast training times and scalable

Predict between two categories

Two-Class Classification

Answers simple two-choice questions, like yes or no, true or false

Answers questions like: Is this A or B?

- **Two-Class Support Vector Machine** ← Under 100 features, linear model
- **Two-Class Averaged Perceptron** ← Fast training, linear model
- **Two-Class Decision Forest** ← Accurate, fast training
- **Two-Class Logistic Regression** ← Fast training, linear model
- **Two-Class Boosted Decision Tree** ← Accurate, fast training, large memory footprint
- **Two-Class Neural Network** ← Accurate, long training times

Generate recommendations

Recommenders

Predicts what someone will be interested in

Answers the question: What will they be interested in?

- **Use the Train Wide & Deep Recommender module** ← Hybrid recommender, both collaborative filtering and content-based approach
- **SVD Recommender** ← Collaborative filtering, better performance with lower cost by reducing dimensionality

Predict values

Regression

Makes forecasts by estimating the relationship between values

Answers questions like: How much or how many?

- **Fast Forest Quantile Regression** ← Predicts a distribution
- **Poisson Regression** ← Predicts event counts
- **Linear Regression** ← Fast training, linear model
- **Bayesian Linear Regression** ← Linear model, small data sets
- **Decision Forest Regression** ← Accurate, fast training times
- **Neural Network Regression** ← Accurate, long training times
- **Boosted Decision Tree Regression** ← Accurate, fast training times, large memory footprint

Discover structure

Clustering

Separates similar data points into intuitive groups

Answers questions like: How is this organized?

- **K-Means** ← Unsupervised learning

Find unusual occurrences

Anomaly Detection

Identifies and predicts rare or unusual data points

Answers the question: Is this weird?

- **One Class SVM** ← Under 100 features, aggressive boundary
- **PCA-Based Anomaly Detection** ← Fast training times

Classify images

Image Classification

Classifies images with popular networks

Answers questions like: What does this image represent?

- **ResNet** ← Modern deep learning neural network
- **DenseNet** ← Modern deep learning neural network

Share this poster: aka.ms/mlcheatsheet

8.2 Internet of Things (IoT)

DIGITAL SURGE PROGRAMME

Demystifying Industry 4.0: the case for transformative technologies.

Cheat sheet

Internet of Things

The Internet of Things (IoT) refers to a system of interrelated, internet-connected objects that can collect and transfer data over a wireless network without human intervention. IoT devices are basically "smart" devices that make "smart" systems (for e.g., smart home, smart factory, smart farms, smart cities, etc.).

Overview	Challenge	Solution	Results
If machines could talk - and tell you when they were going to break down.	When a machine is not functioning properly, productivity drops. How can you tell, quickly, that something is wrong?	IoT sensors on agricultural machinery were connected to the cloud. Managers & workers received immediate notifications when the machine was not functioning properly.	This also resulted in predictive data analysis from the information provided. Engineers started to build a picture and predict when the machine would break and plan maintenance. This increased productivity to 100%.
Remote Monitoring - Improving work performance.	Staff shortages for a civil engineering company reduced the number of onsite visits that could be carried out.	Rather than manually taking readings and producing reports, data was collected on IoT sensors, uploaded to the cloud and this generated a report.	Improved efficiencies, enabling the company to take on more projects and increase their customer base.
Blinding the competition - how to get a USP with IoT.	Consumers want smarter, simpler everyday devices.	Blinds that are IoT enabled - features include a camera, thermometer, lighting and sensors, allowing for seamless integration into the home environment.	"Work still on-going on IoT products at Bloc Blinds as it is an exciting space with loads of potential!" **Read how Magherafelt's Bloc Blinds has incorporated IoT technologies**

Invest Northern Ireland

European Union
European Regional Development Fund
Investment for Growth and Jobs

8.3 Robotic Process Automation (RPA)

ROBOTIC PROCESS AUTOMATION CHEAT SHEET

RPA Basics

RPA

RPA is a **Robotic Process Automation** which is used for automating the current workflows with the help of robots to reduce human intervention at every point

Robotic: Machines that mimics the human activities and actions are called as **robots**

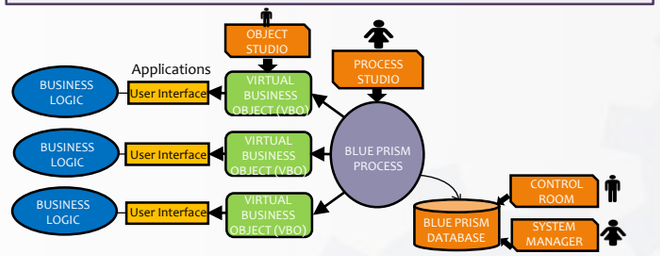
Process: Sequence of steps which is used to perform a particular task

Automation: Any process which is done by robot without any human intervention and provides high degree of accuracy

VENDOR	Free Version	Pricing	Usability	Global Coverage
Another Monday	----	----	----	Europe
AntWorks	----	----	----	----
Arago	----	----	----	----
Automation Anywhere	----	Per Process	Drag & Drop, Macro Recording	Global
BluePrism	----	Per bot	Drag & Drop	----
Contextor	----	----	----	EMEA & North America
Jidoka	----	----	----	----
Kofax	----	----	----	Global
Kryon Systems	----	----	----	----
Nice systems	----	----	----	Global
Pega	----	----	----	Global
Redwood software	----	----	----	----
UiPath	UiPath Community Edition	Per bot	Drag & Drop, Macro Recording	Global
Visual Cron	45 day free trial	Per server	Drag & Drop	----
WorkFusion	WorkFusion RPA Express	Per process	Drag & Drop, Macro Recording	Global

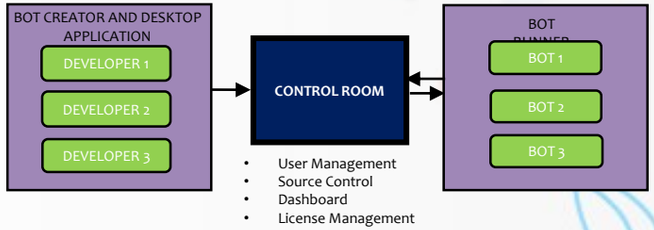
Blue Prism

It is built with a simple concept by replicating user activity. **SQL Server** is the tool where data is stored



Automation Anywhere

The architecture consists of **client, bots** and **control room**. Automation Anywhere is a software designed to virtually automate any computer process. **SQL express** is a tool where data is stored.



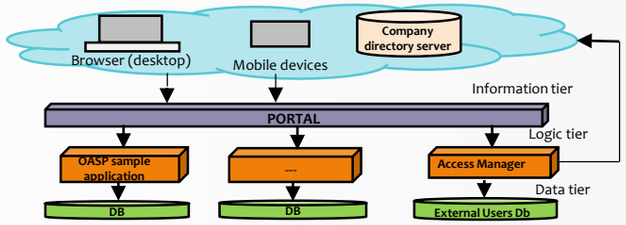
- User Management
- Source Control
- Dashboard
- License Management

Ui Path

It is a simple software automation and application integration expert.

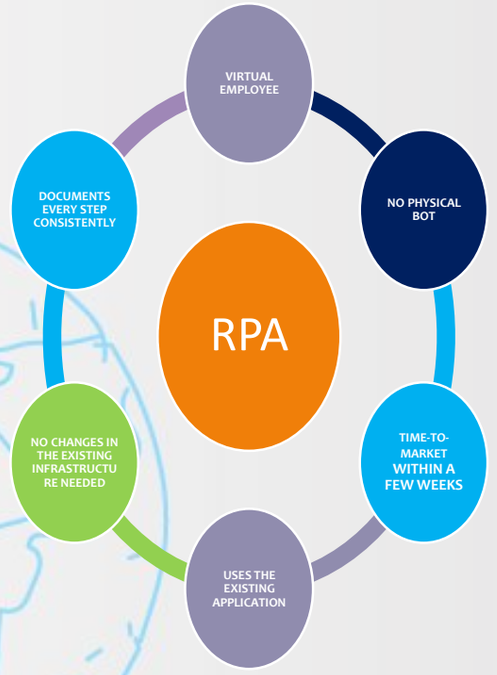
UiPath consists of three parts

- **UiPath** studio
- **UiPath** robots
- **UiPath** orchestrator



RPA Tool Factors

- Technology: Many organizations perform the tasks outside the local system using Virtual Machines or Citrix. Hence, the tool must be platform independent and support any type of application
- Scalability: How quickly and easily the tool responds to the business requirements
- Security: Implementations of security controls must be measured
- Total Cost: The initial set-up cost, vendor license fee, maintenance cost must be taken into consideration while selecting the tool
- Ease of use and control: The tool must be user friendly to increase the efficiency and employee satisfaction
- Vendor Experience: It improves the speed of implementation by reducing the work required to implement RPA software
- Maintenance and support: To make sure that the required service level agreements are met
- Quick Deployment: The tool must be able to work as a real end-user by interacting with applications.



Benefits of Using RPA

- Reduces burden on IT: It does not disturb underlying legacy systems
- Reliability: As the bots can work 24*7 effectively
- Cost cutting technology: It reduces the costs by reducing the size of manual workforce
- No coding required: To use RPA tools, a person need not have the programming skills
- Accuracy: It functions with accuracy and is less prone to errors
- Productivity rate: Execution time is much faster than the manual process approach
- Compliance: It follows the rules to provide audit free trail
- Consistency: Repetitive tasks are performed in the same way
- Increase employee engagement: It lets the employee to focus on value-added activities

Applications of RPA

- It is used in customer service, to automate service order management and quality reporting
- Travel and logistics: for ticket booking, passenger details and accounting
- Human Resource: New employees joining formalities, payroll process and hiring shortlisted candidates
- Health care: In patient registrations and billing
- Banking and Financial services: It can be used for card activation and fraud claims and discovery
- Government: Change of address and license renewal

IntelliPaat

FURTHERMORE:

RPA Training using UiPath

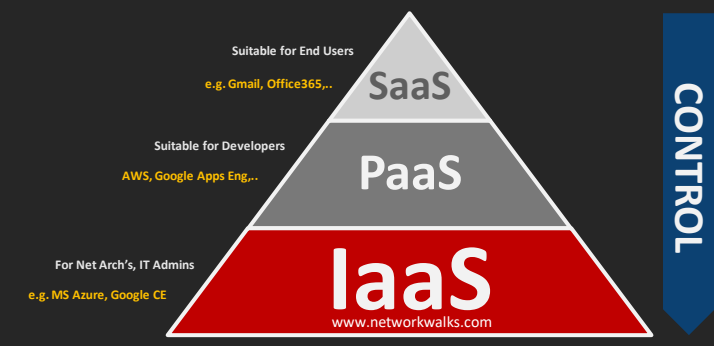
8.4 Cloud Computing

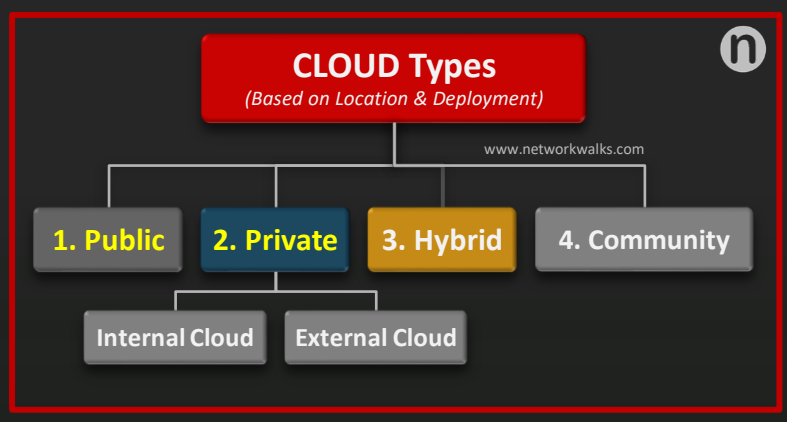
Cloud Computing – Summary Cheat sheet

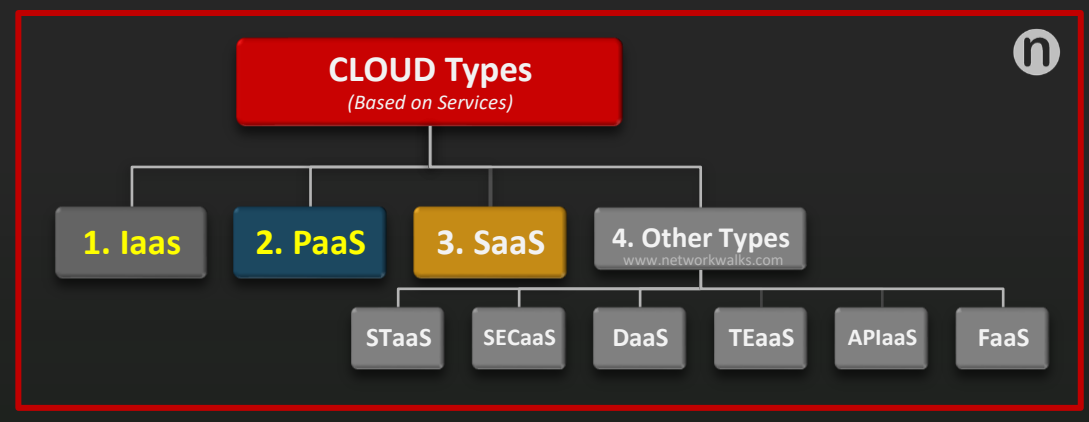
Version 2

What is Cloud Computing ???

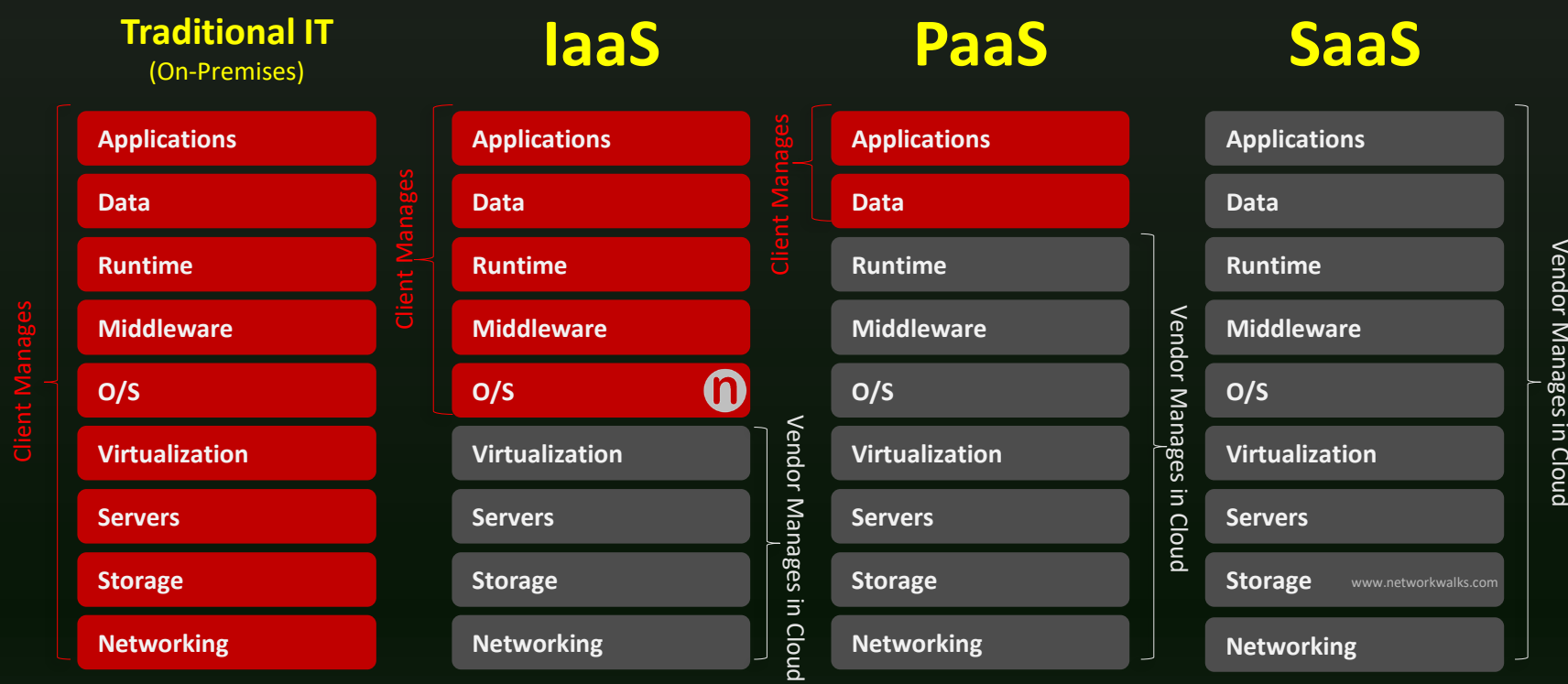
"Cloud Computing is an approach to offer IT Services to customers remotely is called Cloud Computing"







Cloud Computing Models



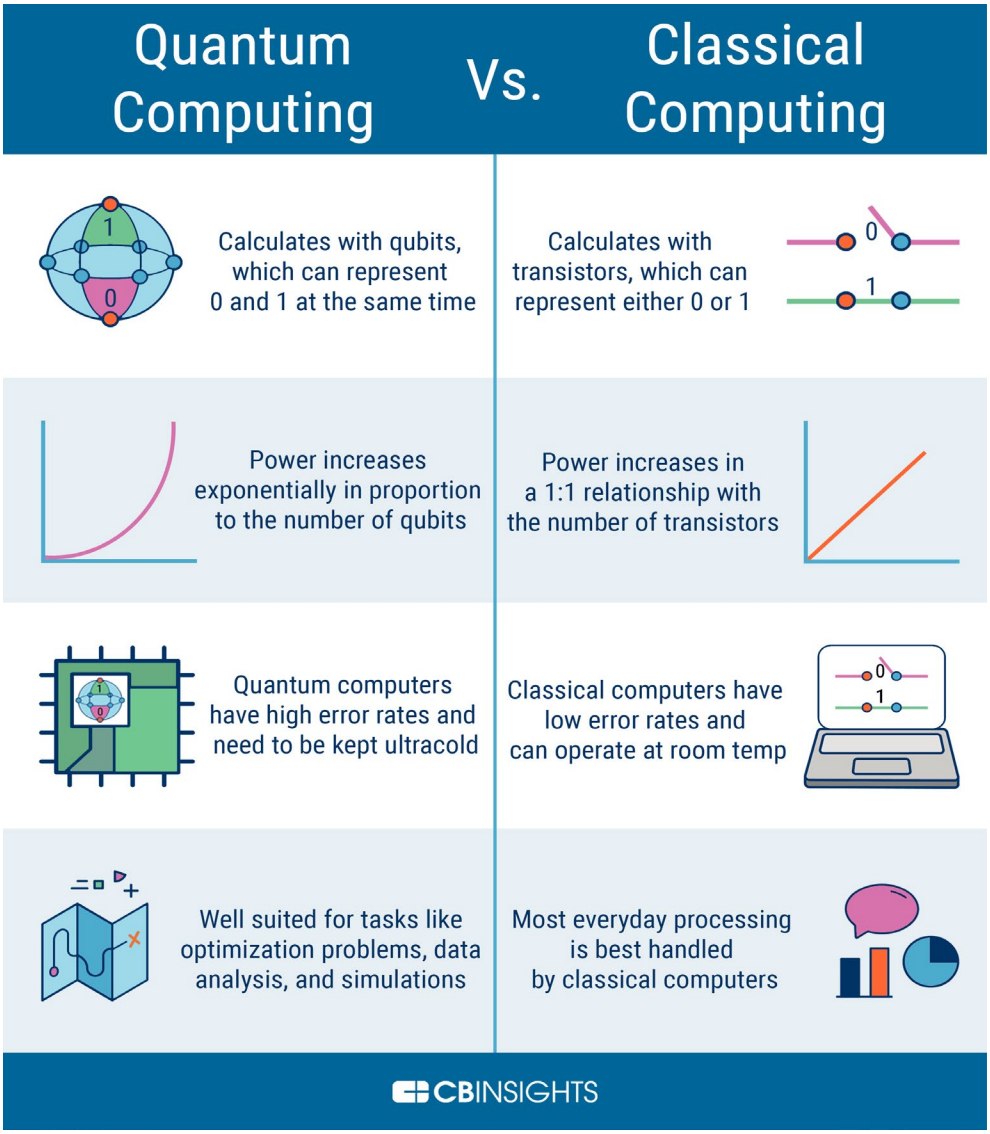
Advantages of Cloud Computing	Disadvantages of Cloud Computing
↑ Cost Effective (Low CAPEX/OPEX)	↓ Less Secure
↑ Scalable, Standardized Model, Highly Reliable	↓ Limited Control (especially SaaS/Paas Models)
↑ Quick in Deployment, Easy Disaster Recovery, Flexible	↓ Requires Extra Bandwidth

www.networkwalks.com

8.5 Quantum Computing

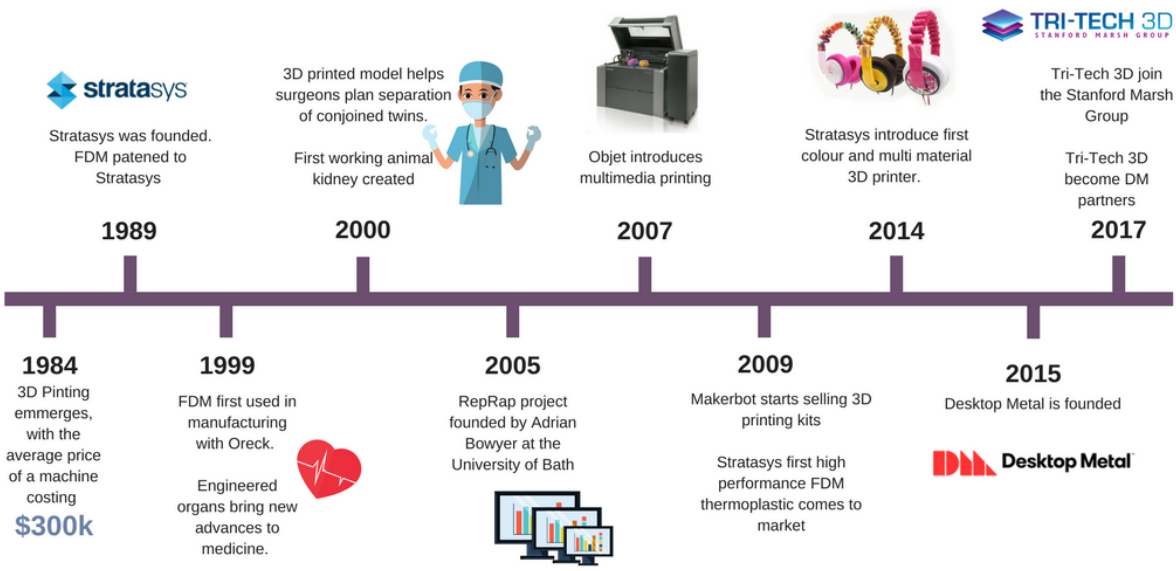
- **History of Quantum computing**

- **Classic computing vs Quantum computing**

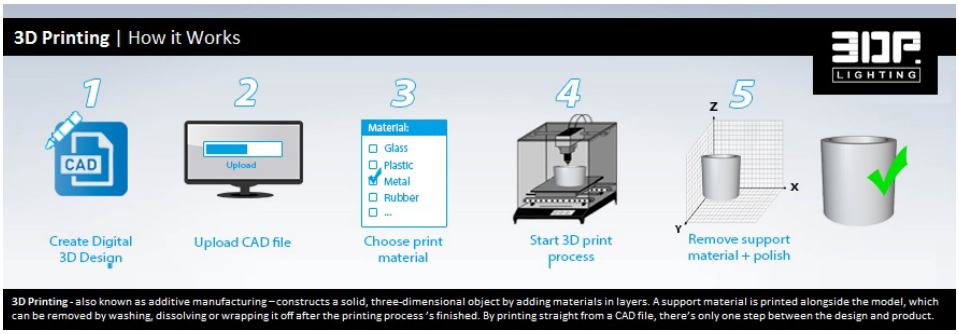


8.6 3D Printing

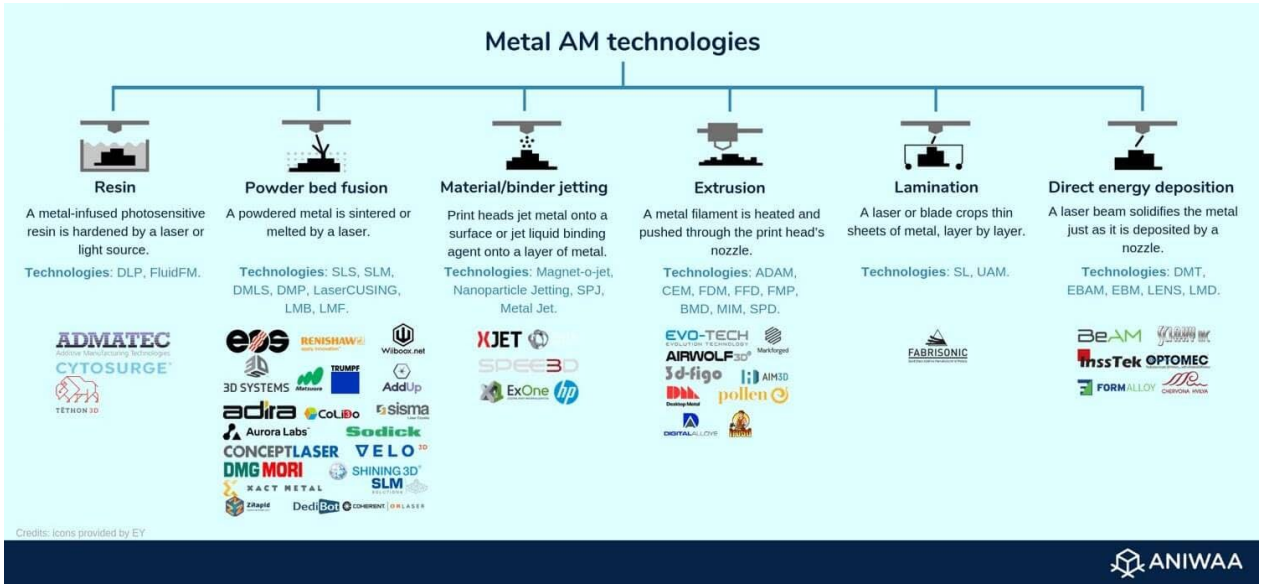
- **History of 3D printing Technology**



- **How 3D printing works?**



- **Type of 3D Printing Technologies**



Cheatography

3D Printing Options Cheat Sheet

by [deleted] via cheatography.com/2754/cs/16505/

Introduction

Our manufacturing industry generally – and 3D printing specifically – is driven by innovation. Indeed, key technological developments and new applications in industrial-grade 3D printing, or additive manufacturing, continue to advance this technology, which has only been around for a little more than 30 years.

Designers and engineers can now choose from several distinct classes of 3D printing technologies. Your choice of “tool” just depends on what it is you’re designing and what its final application is. Here’s a brief roundup of some of the main industrial-grade 3D printing options:

Source: https://www.medicaldesignandoutsourcing.com/3d-printing-options-medical-device-development/

Stereolithography (SL)

Stereolithography (SL) uses an ultraviolet laser that draws on the surface of a liquid thermoset resin to create thousands of thin layers until final parts are formed. SL is used to create concept models, cosmetic prototypes, and complex parts with intricate geometries.

Selective laser sintering (SLS)

Selective laser sintering (SLS) uses a CO2 laser that lightly fuses nylon-based powder, layer by layer, until final thermoplastic parts are created. SLS produces accurate prototypes and functional production parts.

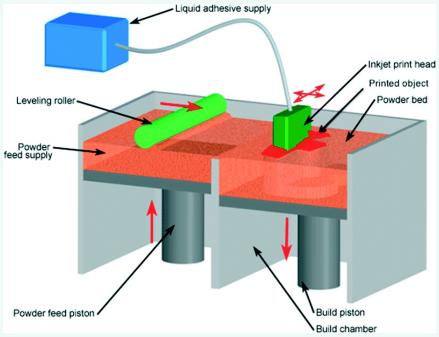
Direct metal laser sintering (DMLS)

Direct metal laser sintering (DMLS) uses a fiber laser system that draws onto a surface of atomized metal powder, welding the powder into fully dense metal parts. DMLS builds fully functional metal prototypes and production parts and works well to reduce metal components in multipart assemblies.

PolyJet

PolyJet uses a jetting process in which small droplets of liquid photopolymer are sprayed from multiple jets onto a build platform and cured in layers that form elastomeric parts. PolyJet produces multi-material prototypes with flexible features at varying durometers and is often used to concept overmolding designs.

3D Printing



Fused Deposition Modeling (FDM)

Fused deposition modeling (FDM) works by feeding a filament or spool of plastic into a heated nozzle, which then extrudes successive layers of thermoplastics onto the workpiece. FDM offers a wide thermoplastic material selection and is leveraged for iterative prototyping.

Continuous Liquid Interface Production (CLIP)

Carbon is the name of the company that is using a process called CLIP, Continuous Liquid Interface Production, which builds parts from the top down, unlike other additive technologies that work from the bottom up. Final plastic parts exhibit excellent mechanical properties and surface finishes.

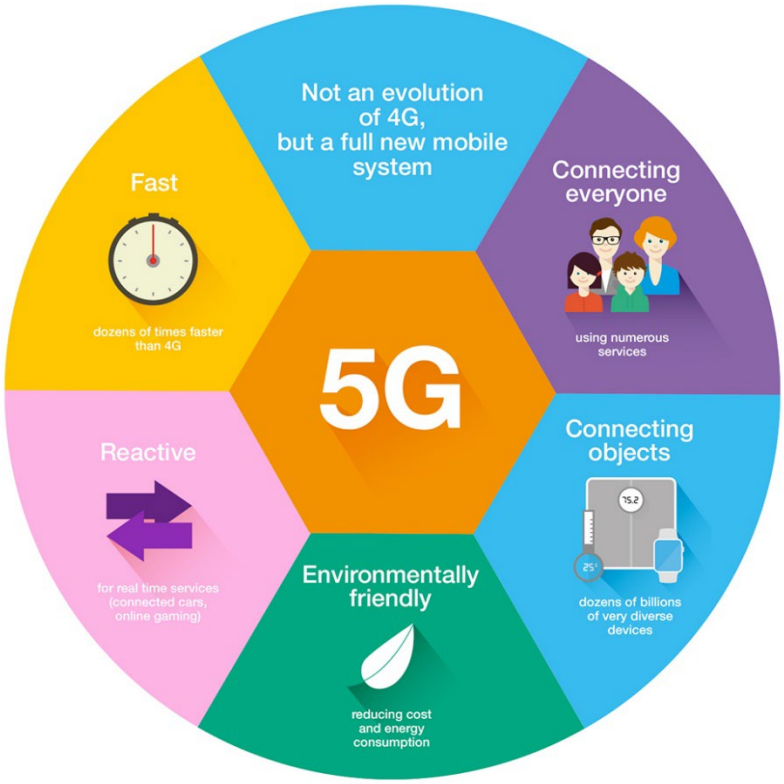
Multi Jet Fusion

Multi Jet Fusion process selectively applies fusing and detailing agents across a bed of nylon powder, which are fused in thousands of layers by heating elements into a solid functional component. Final parts exhibit improved surface roughness, fine feature resolution, and more isotropic mechanical properties when compared to processes like SLS.

8.7 5th Generation Network (5G)

- **History of 5G network**

- **Advantages of 5G network**



8.8 Extended Reality (XR)

DIGITAL SURGE PROGRAMME

Demystifying Industry 4.0: the case for transformative technologies.

Cheat sheet

Immersive Tech

Immersive technologies create experiences by merging the physical world with a digital / simulated reality. Augmented reality (AR) and virtual reality (VR) are the two main types of immersive tech. AR blends computer-generated information onto the user's real environment. VR uses computer-generated information to provide a full sense of immersion.

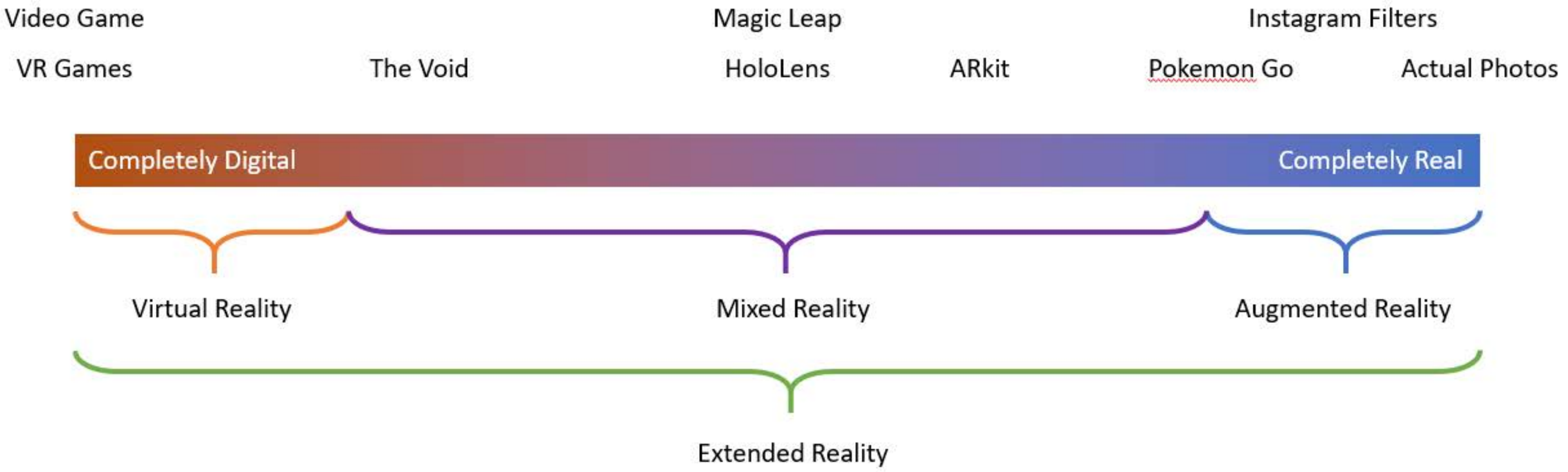
Overview	Challenge	Solution	Results
VR versus Covid-19 - Continuing business remotely.	Due to Covid-19, restrictions a company who delivered forklift driver training was unable to provide practical instruction sessions.	Use of virtual reality to train construction workers to use a forklift remotely. *Also being used to deliver Health & Safety training to oil rig workers before they are deployed.* *VR is being used to show potential apprentices what experience they would receive without distracting workers from their activities.*	Training occurred remotely using VR for the first two days, then onsite with the forklift on the 3rd. This improved productivity, as the forklift became available for work purposes for two days, when it was previously required for training.
Adding value through AR – Business growth in a post Covid-19 environment.	An interior design consultancy was struggling with a lack of business due to Covid-19 restrictions. Designers were unable to visit customers' homes.	Customers used AR to take measurements, and designers were able to create and place items within their 3D world.	They were able to survive Covid-19 by bringing their business online. Also, they can now access a wider client base and scale their business, as with AR their market is no longer restricted by distance.
Creating Tourist Destinations using AR – increasing visitor numbers & revenue.	Tourists, day trippers, and locals are all looking for new experiences.	An interactive AR app to create a huge scale solar system trail. There is also an app for those following the trail giving information on the planets and solar system.	Will reach millions, bring people together and showcase UK creativity globally. **Read more about Northern Ireland's *Our Place in Space* - a 3 dimensional sculpture trail, interactive AR app!**

Invest Northern Ireland

European Union
European Regional Development Fund
Investment for Growth and Jobs

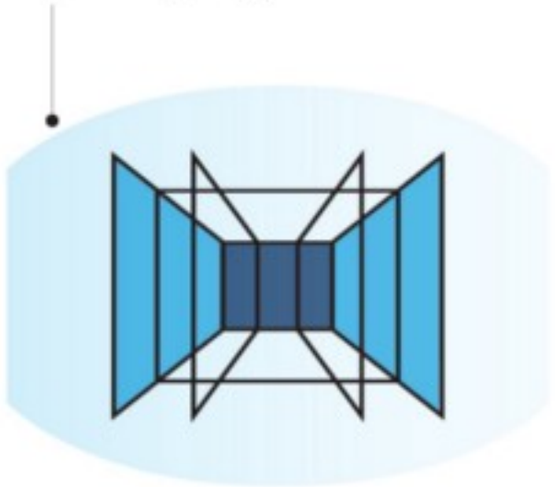
The Digital Surge Programme is part funded by Invest Northern Ireland and the European Regional Development Fund under the Investment for Growth & Jobs Northern Ireland (2014-2020) Programme.

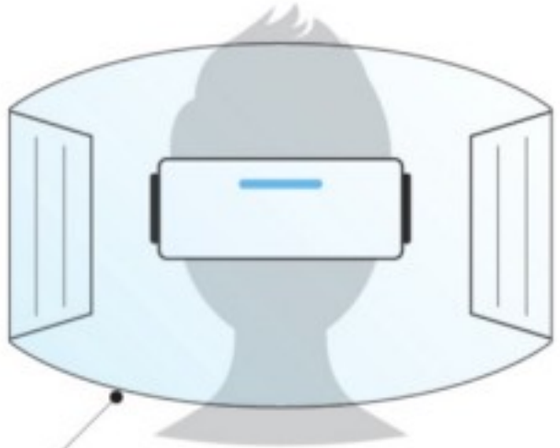
Reality – Virtuality Spectrum



VIRTUAL REALITY (VR)

Completely digital environment

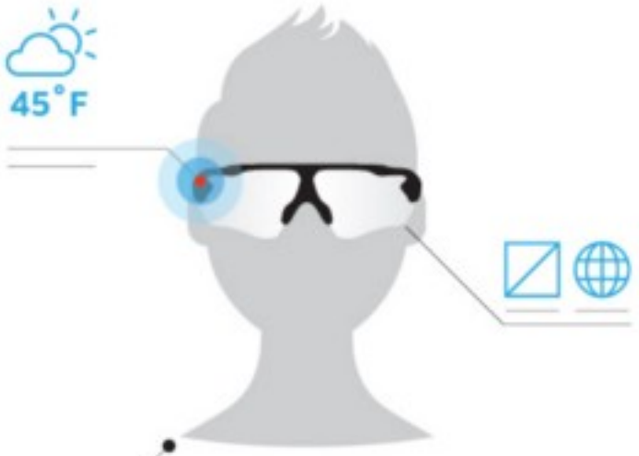




Fully enclosed, synthetic experience with no sense of the real world.

AUGMENTED REALITY (AR)

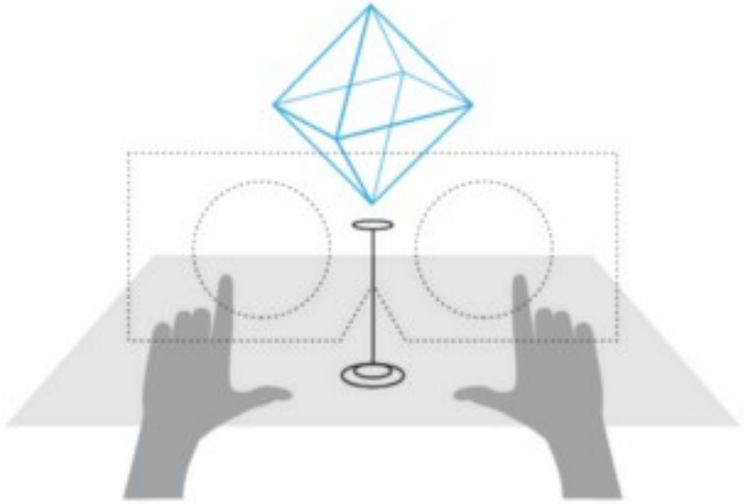
Real world with digital information overlay

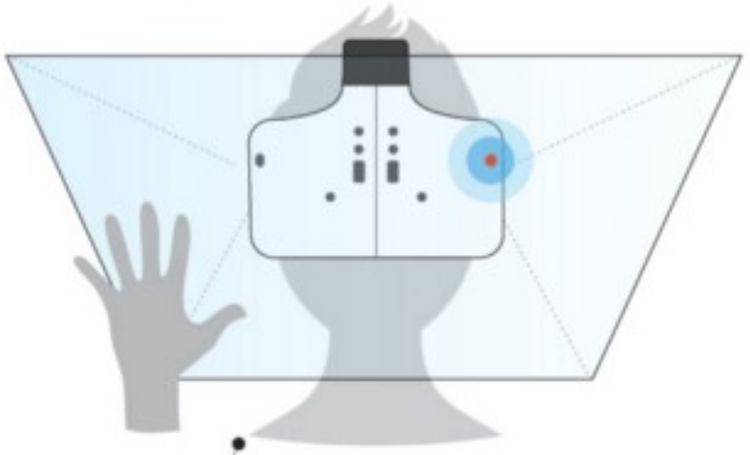


Real world remains central to the experience, enhanced by virtual details.

MERGED REALITY (MR)

Real and the virtual are intertwined





Interaction with and manipulation of both the physical and virtual environment.

8.9 Blockchain

DIGITAL SURGE PROGRAMME

Demystifying Industry 4.0: the case for transformative technologies.

Cheat sheet

Blockchain

Blockchain is a shared, immutable ledger for recording transactions and tracking assets in a business network. With blockchain technology, companies can establish a transparent supply chain, verify the authenticity of product claims and gain consumer trust.

Overview	Challenge	Solution	Results
Connecting through Blockchain – Increasing productivity and profitability.	How to track devices through a supply chain in the medical devices sector?	Each supplier was invited to the blockchain, added data about the parts they were making, and added status.	Able to view the origin of raw materials, have an estimated delivery of parts from 30 different suppliers, and guarantee provenance of medical grade products. This led to a significant increase in productivity, approximately 5-10% increase in turnover due to increased capacity, and 70% reduction in manual report creation.
Food and Beverage Traceability – Showcasing the truth of your product.	How to create a safer and more transparent supply chain?	Used blockchain technology and included a QR code on food and beverage products to allow the customer to see exactly how the product was made.	Using blockchain enhanced the transparency of product information, reduced food fraud, and waste. It increased food safety and profitability among food suppliers. And it improved consumer information, boosting trust & sales. **See how a County Down enterprise uses blockchain to reveal everything that a consumer would want to know about their beer!**

Invest Northern Ireland

European Union
European Regional Development Fund
Investment for Growth and Jobs

8.10 Cyber Security

Cyber Security
Quick Reference Guide

Security Risks

Businesses worldwide are at risk for security breaches. While large, well-known companies seem like a likely target, small and medium-sized organizations and individuals are also at risk. There are many ways data can be compromised, including viruses, phishing scams, hardware and software vulnerabilities, and network security holes.

Did you know?

11% of U.S. adults have had personal information stolen

1 in 5 people have had an email or social media account hacked

98% of software applications are vulnerable

Only **40%** of adults know how to protect themselves online

Confidential Information

When dealing with security, confidentiality means private information is never viewed by unauthorized parties. Confidential information must only be accessible to those authorized to view the sensitive data. Confidential information includes:

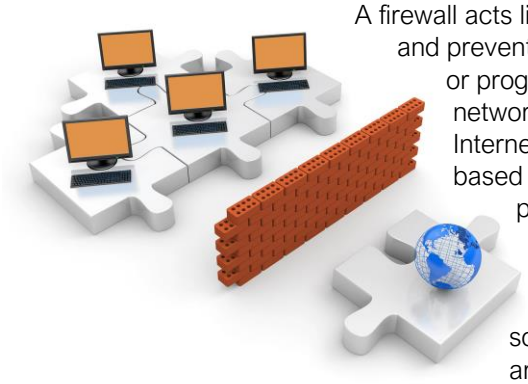
Personal Information

- Social Security Number
- Home Address
- Salary History
- Performance Issues
- Credit Card Numbers

Corporate Information

- Processes
- Customer Lists
- Research and Development
- Business Strategies
- Objectives and Projections

Firewalls



A firewall acts like a security guard and prevents unauthorized people or programs from accessing a network or computer from the Internet. There are hardware-based firewalls, which create a protective barrier between internal networks and the outside world, and software firewalls, which are often part of your operating system.

Passwords

The first line of defense in maintaining system security is using complex passwords. Use passwords that are at least 8 characters long and include a combination of numbers, upper and lowercase letters, and special characters. Hackers have tools that can break easy passwords in just a few minutes.

How Long Does it Take to Crack a Password?

There are 2 kinds of passwords:

Simple

Lowercase letters only

Complex

Upper & lowercase letters, numbers, & special characters

Complex passwords are **EXPONENTIALLY** More difficult to crack

Use them!

Here's how long it takes to crack a password when it's **simple** vs. **complex**

Characters	Password	Time to crack
8	ghiouhel	4 hours, 7 min
	ghiouH3l	6 months
9	houtheouh	4 days, 11 hours
	Houtheo!2	1060 years
10	ghotuhilhg	112 days
	gh34uhilh!	1500 years
11	wopthiendhf	8 years, 3 months
	w3pthi7ndh!	232,800 years
12	whithgildnzq	210 years
	@hi3hg5ldnq!	15,368,300 years

Source: mywot.com

Malware

Malware is short for "malicious software." It is written to infect the host computer. Common types of malware include:

VIRUS

Replicating computer program that infects computers

ADWARE

Hijacks your computer or browser and displays annoying advertisements

SPYWARE

Secretly tracks your internet activities and information

TROJAN

Malicious program that tries to trick you into running it

Online Browsing

Browsers communicate to websites with a protocol called HTTP, which stands for Hyper Text Transfer Protocol. HTTPS is the secure version of HTTP. Websites that use HTTPS encrypt all communication between your browser and the site.

Secure sites have an indicator, like a padlock, in the address bar to show the site is secure. You should always ensure security when logging in or transferring confidential information.

Sites without HTTPS are not secure and should never be used when dealing with personal data. If you are simply reading an article or checking the weather, HTTP is acceptable.

Network Security

- Use Wi-Fi password security and change the default password
- Set permissions for shared files
- Only connect to known, secure public Wi-Fi and ensure HTTPS-enabled sites are used for sensitive data
- Keep your operating system updated
- Perform regular security checks
- Browse smart!

Email and Phishing

A phishing email tries to trick consumers into providing confidential data to steal money or information. These emails appear to be from a credible source, such as a bank, government entity, or service provider. Here are some things to look for in a phishing email:

Grammatical Errors

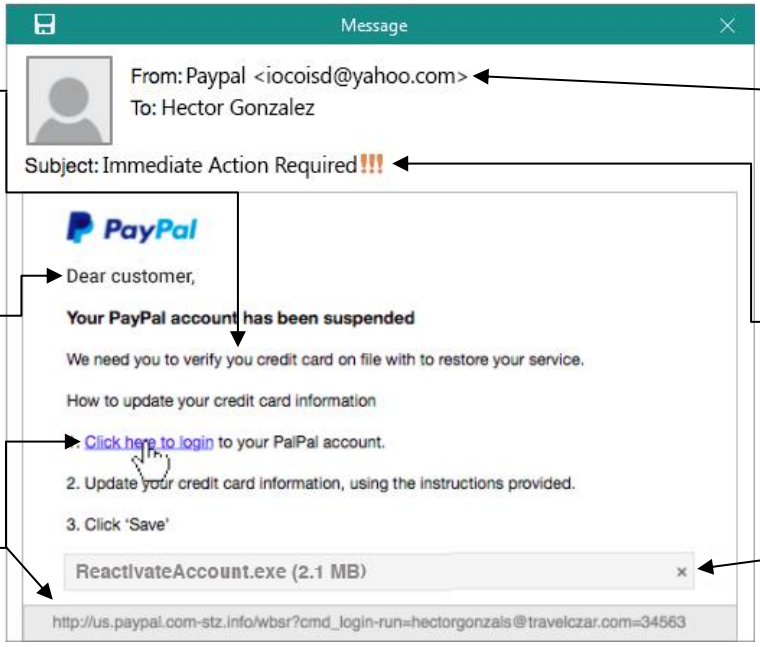
Spelling mistakes and poor grammar

Generic References

Not being addressed by your name

Hover Link

Always check where links lead before clicking



Sender's Address

The address should be correlated with the sender

Immediate Action

Beware of anything that calls for urgent action

Attachments

Never open an attachment you aren't expecting