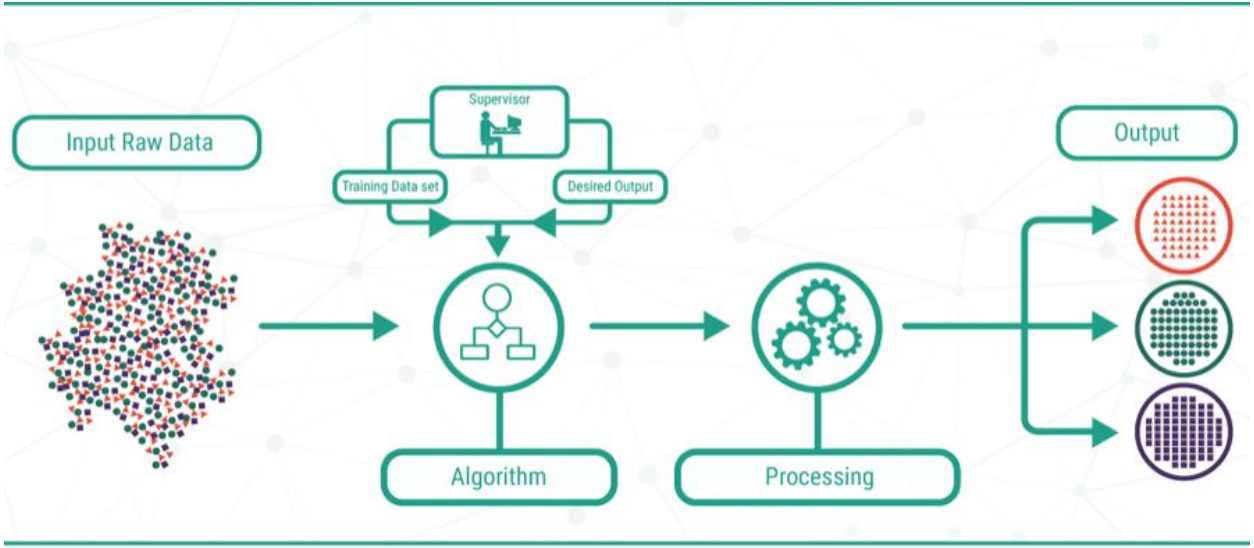
Supervised Learning

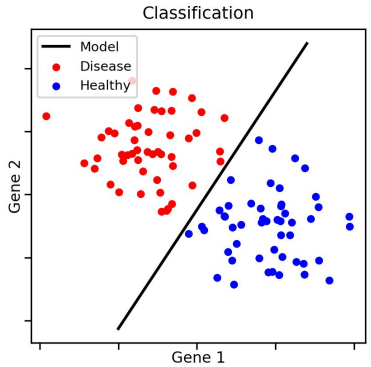
Data Science & Machine Learning
ENSAE - ITS4 & ISE2

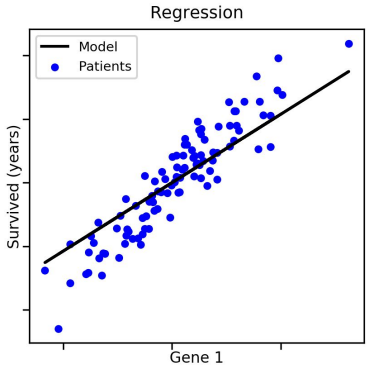
Mously DIAW
2021 - 2022

Summary

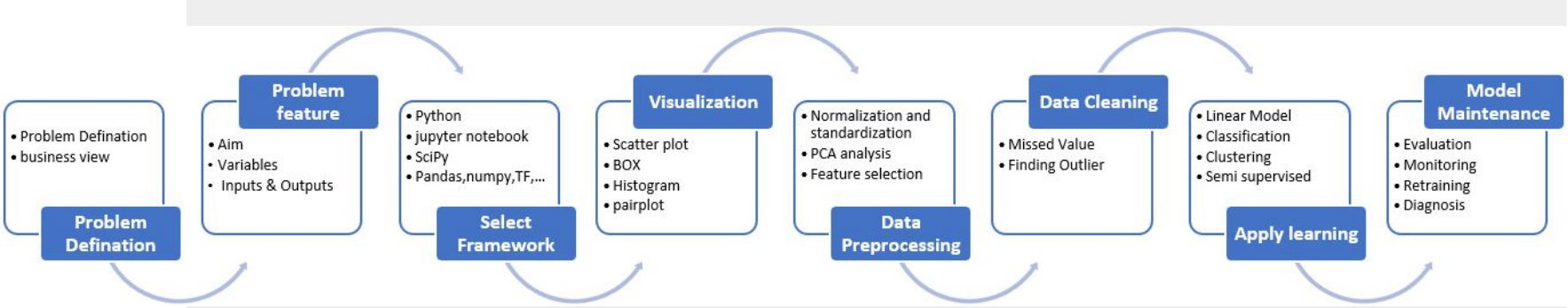
What is Supervised Learning?

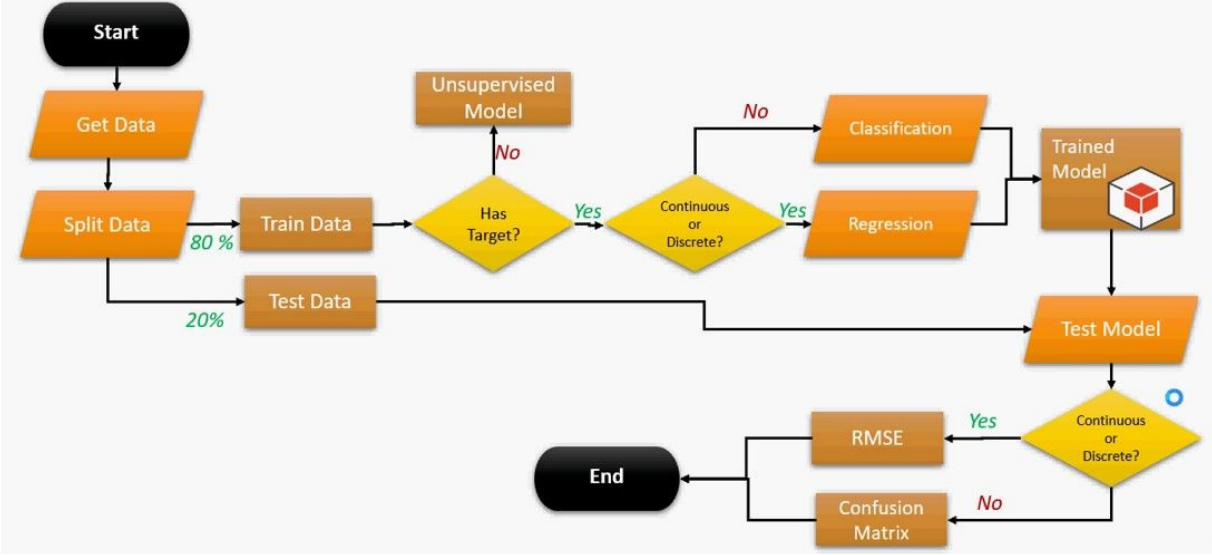






Machine Learning - Workflow





I. Regression

What is Regression ?

Regression searches for relationships among variables.

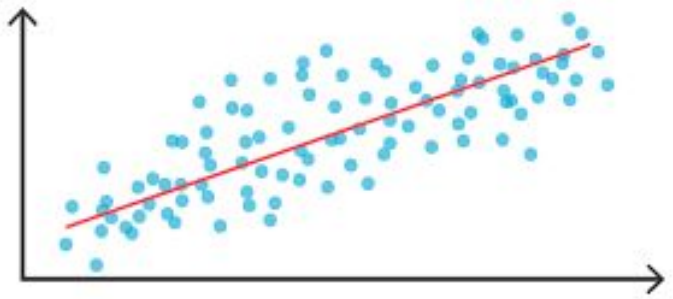
target: quantitative

independent variables: quantitative / categorical

For example, you can **observe several employees of some company** and try to understand how their salaries depend on the features, such as experience, level of education, role, city they work in, and so on.

Similarly, you can try to establish a **mathematical dependence of the prices of houses** on their areas, numbers of bedrooms, distances to the city center, and so on.

Regression is used in many different fields: economy, computer science, social sciences, and so on.



What is Regression ?

In other words, you need to **find a function that maps some features or variables to others sufficiently well**.

The **dependent features** are called the **dependent variables, outputs, or responses.**

The **independent features** are called the **independent variables, inputs, or predictors.**

Regression problems usually have **one continuous and unbounded dependent variable**. The **inputs**, however, can be **continuous, discrete, or even categorical data** such as gender, nationality, brand, and so on.

It is a common practice to denote the **outputs with y** and **inputs with x.** If there are two or more independent variables, they can be represented as the vector $\mathbf{x} = (x_1, \ldots, x_r)$, where r **is the number of inputs.**

Problem formulation: $y = f(\mathbf{x})$

f: estimated regression function, capture the dependencies between the inputs and output sufficiently well

Regression: Linear Regression

Linear regression is probably one of the most important and widely used regression techniques. It's among the simplest regression methods. One of its main advantages is the ease of interpreting results.

Problem Formulation

you assume a linear relationship between y and $\mathbf{x}$:

$$y = f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$$

Regression: Linear Regression (LR)

$$y = f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$$

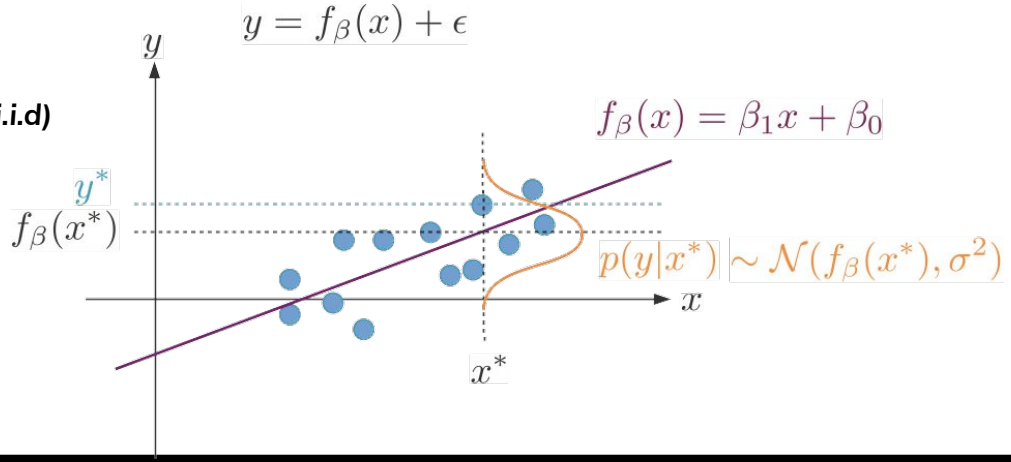
Simple LR	Multiple LR	Polynomial LR
$f(x) = \beta_0 + \beta_1 x$	$f(x_1, \ldots, x_r) = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r$	degree 2: $f(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1 x_2 + \beta_5 x_2^2$

How do we find the values of the coefficients $\boldsymbol{\beta_0, \beta_1, \ldots, \beta_r}$?

Linear Regression: Hypothesis

1. ***linearity***, *is that we can approximate, for each point of our data set, its label by a linear combination of the value of the variables that describe it*
2. ***normality assumption***, *is that the noise is Gaussian, or normally distributed: the* ***ϵ****i are the realization of a Gaussian random variable, centered at 0 and with standard deviation* ***σ****:* $\varepsilon \sim \mathcal{N}(0, \sigma 2)$.
3. ***the independence hypothesis***, *is that the n pairs (x(i),y(i)) are mutually independent realizations of the variables x and y.*

=> *independent and identically distributed (i.i.d)*



Linear Regression: How to find the values of the coefficients?

- **Maximization of the likelihood**: maximize the *log* likelihood
- **Least squares method** (OLS): **minimize the sum of squares** of the differences between each label and its predicted value

$$\arg\min_{\beta \in \mathbb{R}^{p+1}} (y - X\beta)^\top (y - X\beta)$$

minimize the gradient

$$\nabla_\beta (y - X\beta)^\top (y - X\beta) = 0$$

$$\nabla_\beta (y - X\beta)^\top (y - X\beta) = -2X^\top (y - X\beta)$$

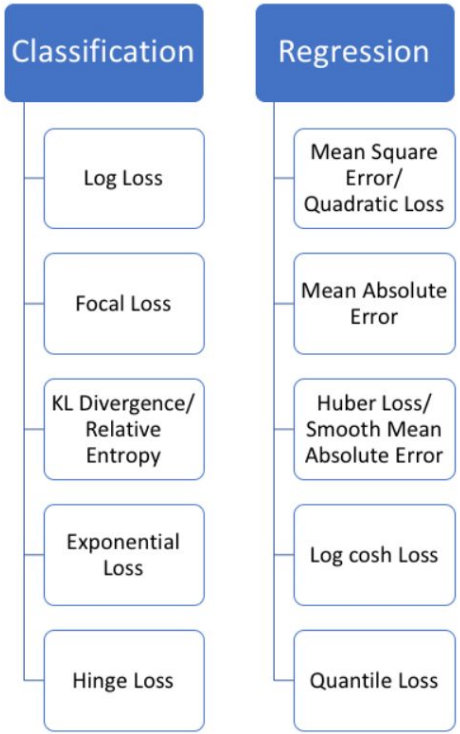
$$X^\top y - X^\top X\beta = 0$$

If the matrix $X^\top X$ is invertible, the solution is unique & explicite

$$\beta = (X^\top X)^{-1} X^\top y$$

Notes: for more details => linear-regression

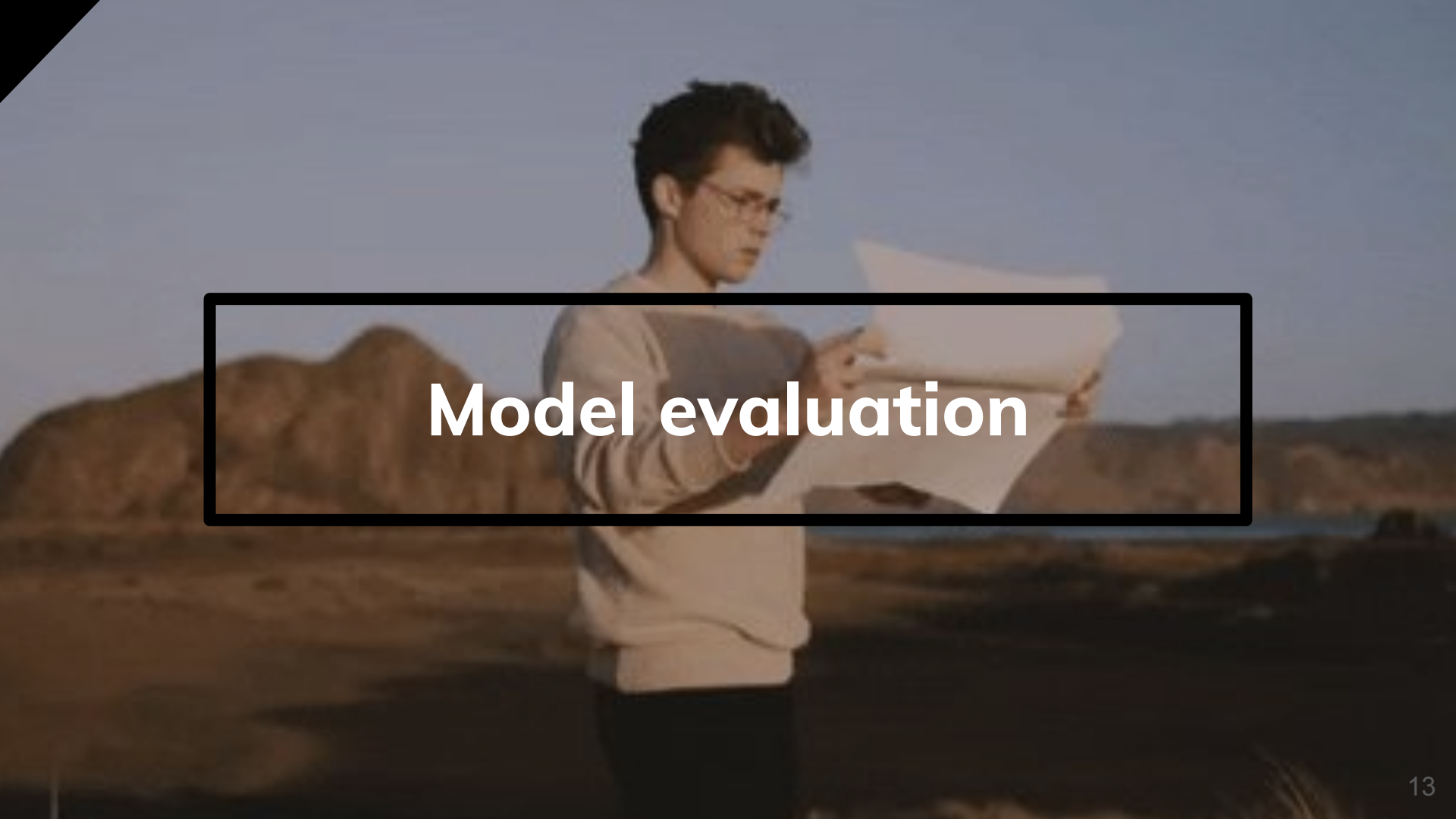
Objective function overview



The **function we want to minimize or maximize** is called the **objective function, or criterion**.

When we are **minimizing** it, we may also call it the **cost function, loss function, or error function**

- **Loss function:** is usually a function defined on a data point, prediction and label, and measures the penalty
- **Cost function:** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty
- **Objective function** is the most general term for any function that you optimize during training


Model evaluation

Regression: Quality of predictions

R^2 score, Coefficient of Determination

It represents the proportion of variance (of y) that has been explained by the independent variables in the model

it can be negative (because the model can be arbitrarily worse)

The best possible score is 1.0, lower values are worse

The **R^2 score** can be calculated as follows:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$.

Regression: Quality of predictions

Adjusted R^2 score

R^2 increases with every predictor added to a model. The adjusted **R^2** will penalize this for adding independent variables

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n-1)}{n-k-1} \right]$$

where:

- ***n*** is the number of points in your data sample.
- ***k*** is the number of independent variables

While **values are usually positive**, they can be **negative** as well

Adjusted R-squared, adds precision and reliability by considering the impact of additional independent variables that tend to skew the results of R-squared measurements.

Regression: Quality of predictions

Mean Squared Error (MSE)

It computes a risk metric corresponding to the expected value of the squared (quadratic) error or loss

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

"*least squares*" refers to minimizing the mean squared error between predictions and expected values

A perfect MSE value is 0.0, which means that all predictions matched the expected values exactly.

The **units of the MSE are squared units**.

- For example, if your target value represents "*dollars*," then the MSE will be "*squared dollars*." This can be confusing for stakeholders; therefore, when reporting results, often the root mean squared error is used instead

A good MSE is relative to your specific dataset.

Regression: Quality of predictions

Root Mean Squared Error (RMSE)

It is an extension of the mean squared error

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

RMSE = sqrt(**MSE**)

Importantly, the square root of the error is calculated, which means that the units of the **RMSE are the same as the original units of the target value** that is being predicted.

A **perfect RMSE value is 0.0**, which means that all predictions matched the expected values exactly.

Regression: Quality of predictions

Mean squared logarithmic error (MSLE) 1/2

It computes a risk metric corresponding to the expected value of the squared logarithmic (quadratic) error or loss.

It is a variation of the Mean Squared Error

MSLE only care about the percentual difference

This means that MSLE will treat small differences between small true and predicted values approximately the same as big differences between large true and predicted values.

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (\log_e(1 + y_i) - \log_e(1 + \hat{y}_i))^2$$

Regression: Quality of predictions

Mean squared logarithmic error (MSLE) 2/2

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (\log_e(1 + y_i) - \log_e(1 + \hat{y}_i))^2$$

True value	Predicted value	MSE loss	MSLE loss
30	20	100	0.02861
30000	20000	100 000 000	0.03100
	Comment	**big difference**	**small difference**

Regression: Quality of predictions

Mean Absolute Error (MAE)

It computes a risk metric corresponding to the expected value of the absolute error loss or l1-norm loss

In a perfectly fitted output regression model, **MAE would be 0**

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over n_{samples} is defined as

Regression: Quality of predictions

Mean Absolute Percentage Error (MAPE)

... also known as mean absolute percentage deviation (MAPD), is an evaluation metric for regression problems.

The idea of this metric is to be sensitive to relative errors. It is for example not changed by a global scaling of the target variable.

$$\text{MAPE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{|y_i - \hat{y}_i|}{max(\epsilon, |y_i|)}$$

where ϵ is an arbitrary small yet strictly positive number to avoid undefined results when y is zero.

Regression: Quality of predictions

Median Absolute Error (MedAE)

... is particularly interesting because it is **robust to outliers**

It's the median of all of the absolute values of the residuals

$$\text{MedAE}(y, \hat{y}) = \text{median}(|\ y_1 - \hat{y}_1\ |, \ldots, |\ y_n - \hat{y}_n\ |)$$

MedAE output is non-negative floating point. **The best value is 0.0**.

Regression: Quality of predictions

Max error

It computes the maximum residual error

It captures the worst case error between the predicted value and the true value

$$\text{Max Error}(y, \hat{y}) = max(|y_i - \hat{y}_i|)$$

In a perfectly fitted output regression model, **max_error would be 0**

Regression: Quality of predictions

AIC (Akaike's Information Criterion)

BIC (Bayesian information criterion)

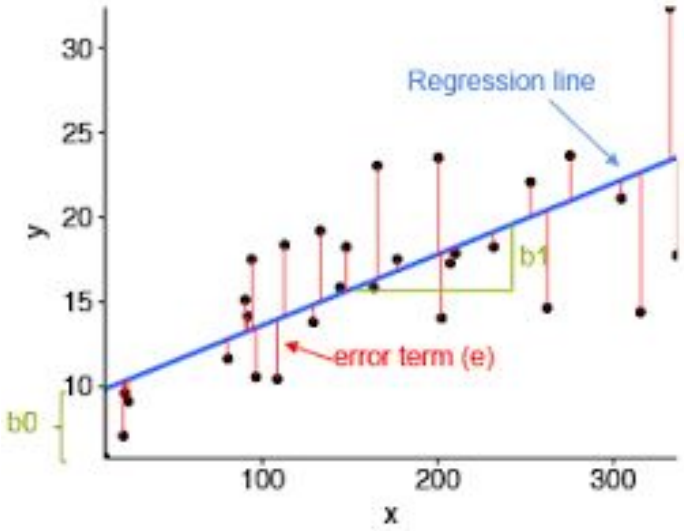
VIF (Variance Inflation Factor)

...


Model Selection

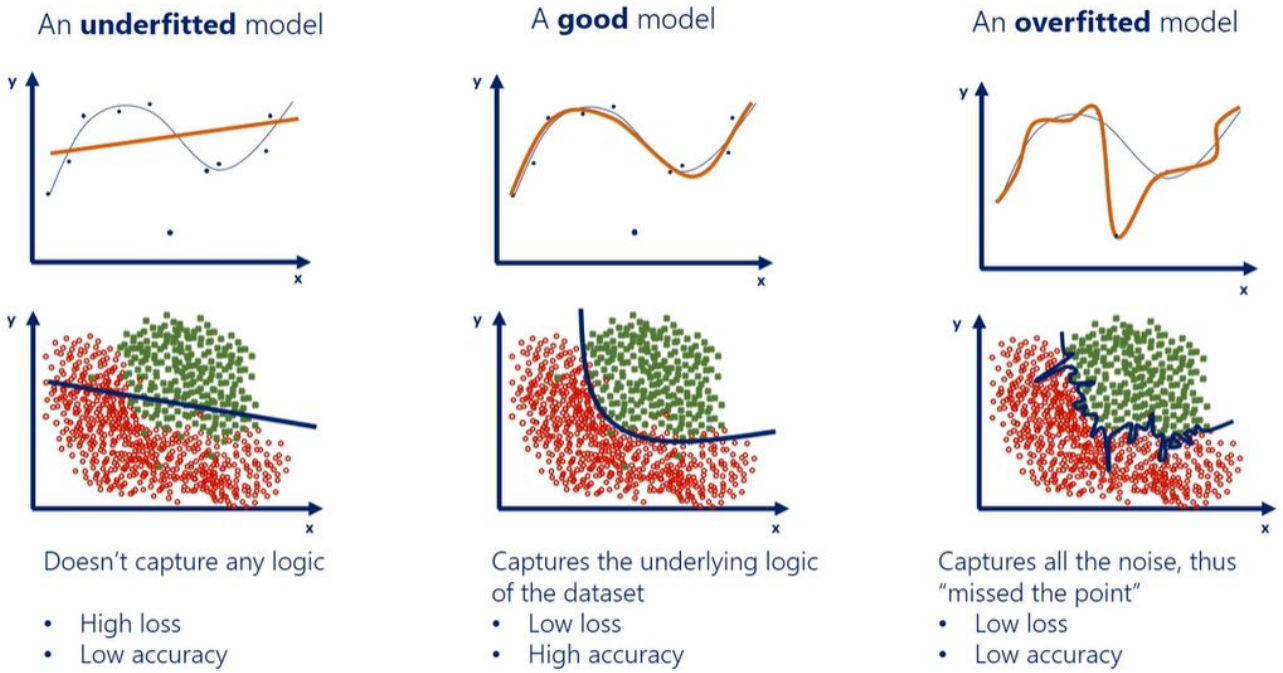
Linear regression: limitations

- **Variables are correlated:** the solution is not unique and the coefficients have a large variability, and the interpretation is more difficult
- **Correlation** is not always **causality**
- Assumes **linear relationship** between variables
- **Multicollinearity** problem
- **Prone to underfitting**
- **Sensitive to Outliers**
- **Data Must Be Independent**
- The **number of variables** is **greater than** the **number of observations:** There is therefore a risk of overfitting



Supervised Learning: overfitting & underfitting

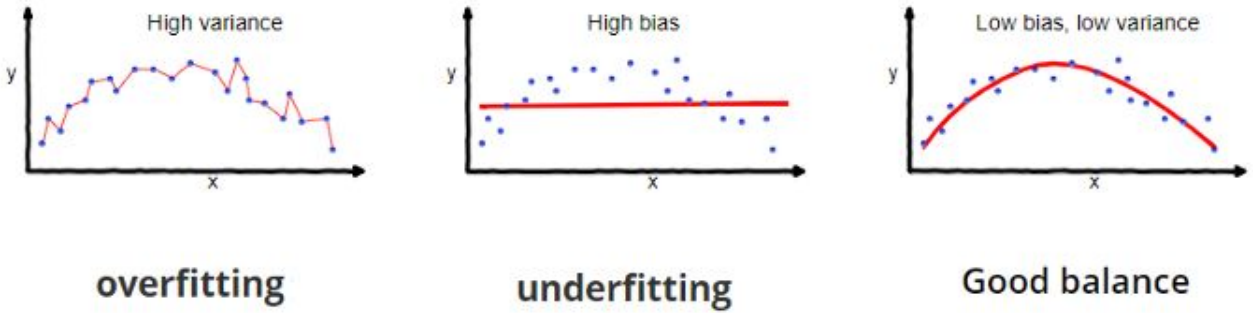
- **Overfitting:** a model that is too specialized on training set data and will not generalize well
- **Underfitting**: the model performs poorly on the training data



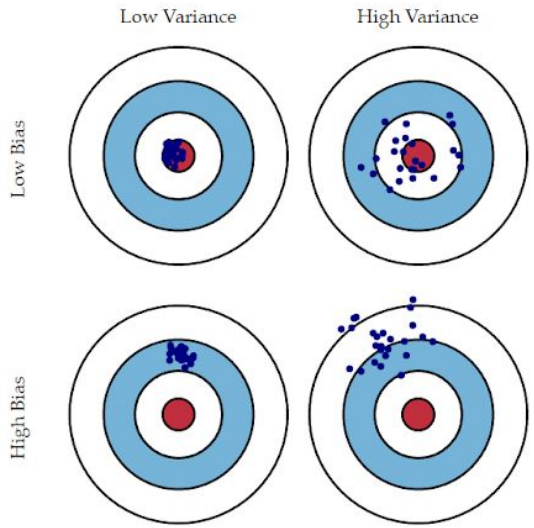
Supervised Learning: overfitting & underfitting

- **Overfitting:** a model that is too specialized on training set data and will not generalize well
  - Increase training data
  - Reduce model complexity / Increase the amount of regularization used
  - Feature selection
  - Use a resampling technique (cross validation) or ensemble methods (bagging/boosting)
  - Increase the amount of regularization used
  - Increase the number of passes or iterations on the existing training data

- **Underfitting**: the model performs poorly on the training data
  - Increase the number of features, performing feature engineering
  - Try alternate machine learning algorithms
  - Decrease the amount of regularization used / Increase model complexity
  - Increase the amount of training data examples
  - Remove noise from the data.



Supervised Learning: Good fit, Bias Variance Trade Off

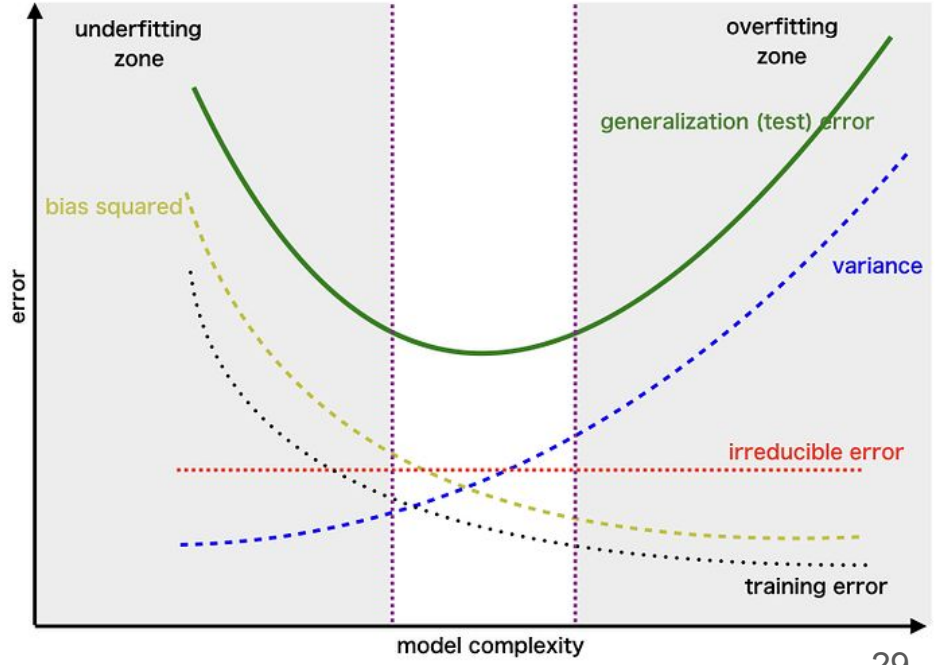


- **Bias:** difference between the average prediction of our model and the correct value
- **Variance:** variability of model prediction

$$TotalError = Bias^2 + Variance + IrreducibleError$$

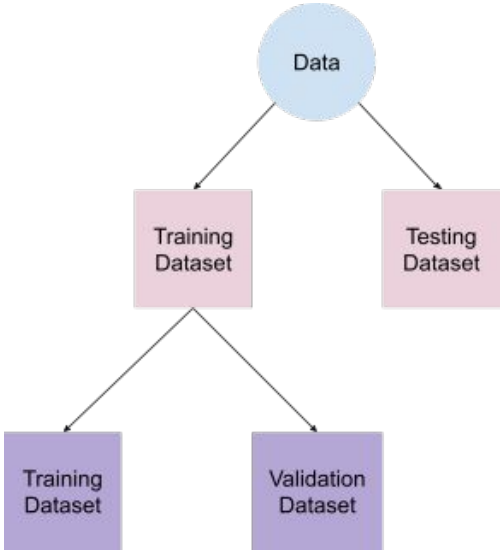
The balance between underfitting and overfitting

A "**good"** model not be perfect, but would be very close to the actual relationship

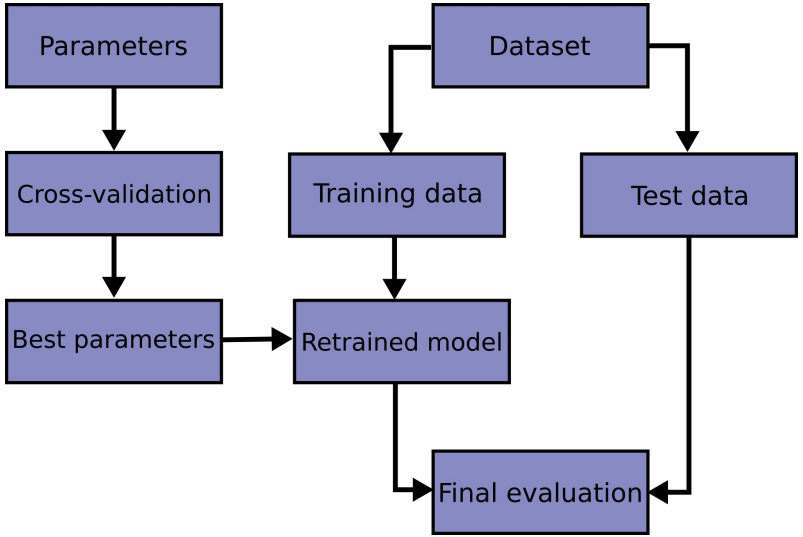


Supervised Learning: Cross validation

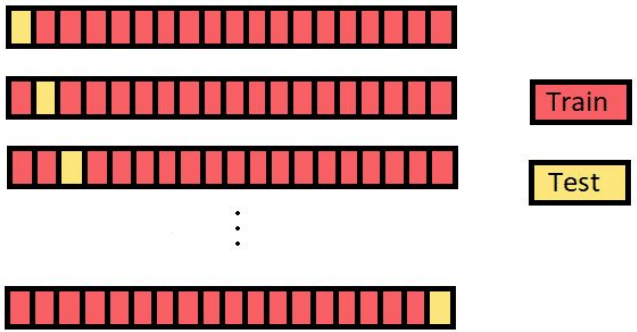
Cross-validation allows us to utilize our data even better



Flowchart of typical cross validation workflow in model training



Supervised Learning: Cross validation



Leave one out (LOO): one observation is taken out of the training set. It's an exhaustive cross validation techniques

Pros:

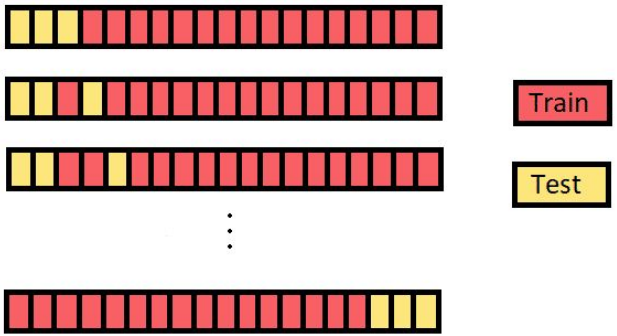
- Simple, easy to understand, and implement

Cons:

- The model may lead to a low bias.
- The computation time required is high.

Supervised Learning: Cross validation

Example: leave 3-out



Leave p-out (LPO): is an exhaustive cross-validation technique, that involves using p-observation as validation data, and remaining data is used to train the model

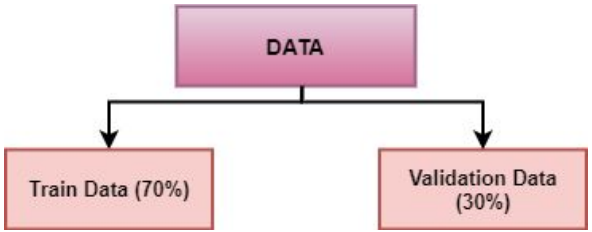
Pros:

- Simple, easy to understand, and implement

Cons:

- The model may lead to a low bias
- The computation time required is high

Supervised Learning: Cross validation



Hold out method: simple kind of cross validation technique. the data is randomly partitioned into train and test set.

Most of the times it is 70/30 or 80/20 split.

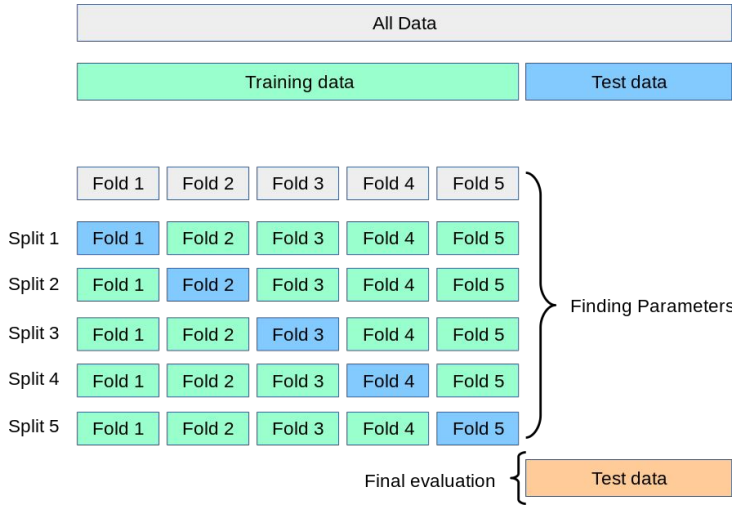
Pros:

- Simple, easy to understand, and implement

Cons:

- Not suitable for an imbalanced dataset.
- A lot of data is isolated from training the model.

Supervised Learning: Cross validation



K-fold: the original dataset is equally partitioned into k subsets or folds. Most of the times it is 3, 5 or 10 folds.

Pros:

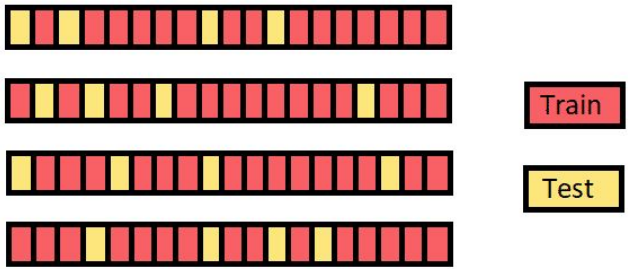
- The model has low bias
- Low time complexity
- The entire dataset is utilized for both training and validation.

Cons:

- Not suitable for an imbalanced dataset.

Notes: https://stats.stackexchange.com/questions/27730/choice-of-k-in-k-fold-cross-validation

Supervised Learning: Cross validation



Repeated random (monte carlo) : splits the dataset randomly into training and validation.

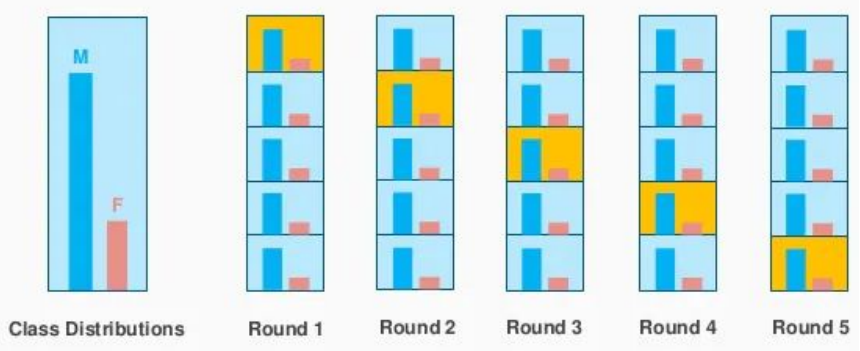
Pros:

- The proportion of train and validation splits is not dependent on the number of iterations or partitions.

Cons:

- Some observations may never be chosen for either training or validation, whereas some might be selected multiple times
- Not suitable for an imbalanced dataset

Supervised Learning: Cross validation



Stratified k-fold: the idea is to try to preserve the distribution of classes in each split while keeping each group within a single split

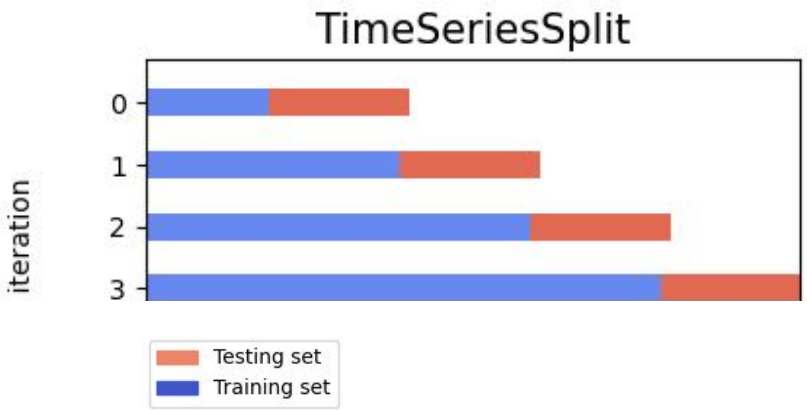
Pros:

- Works well for an imbalanced dataset

Cons:

- Now suitable for time series dataset.

Supervised Learning: Cross validation



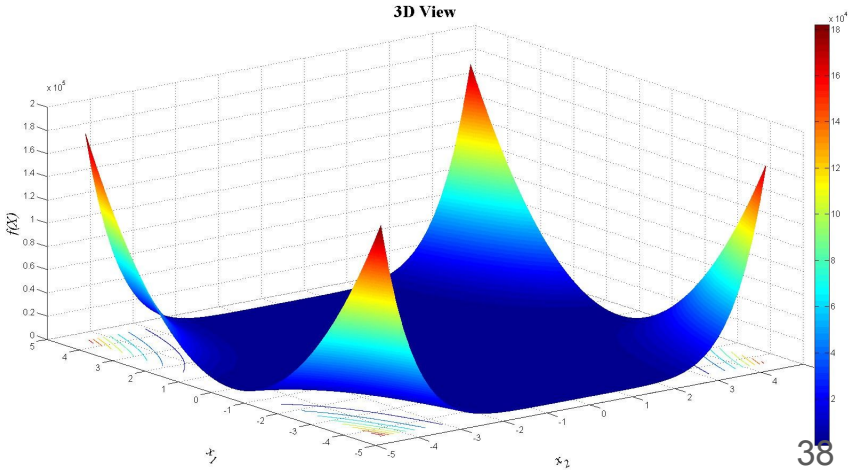
Time series: The order of the data is very important for time-series related problem

Machine Learning: Hyperparameters tuning

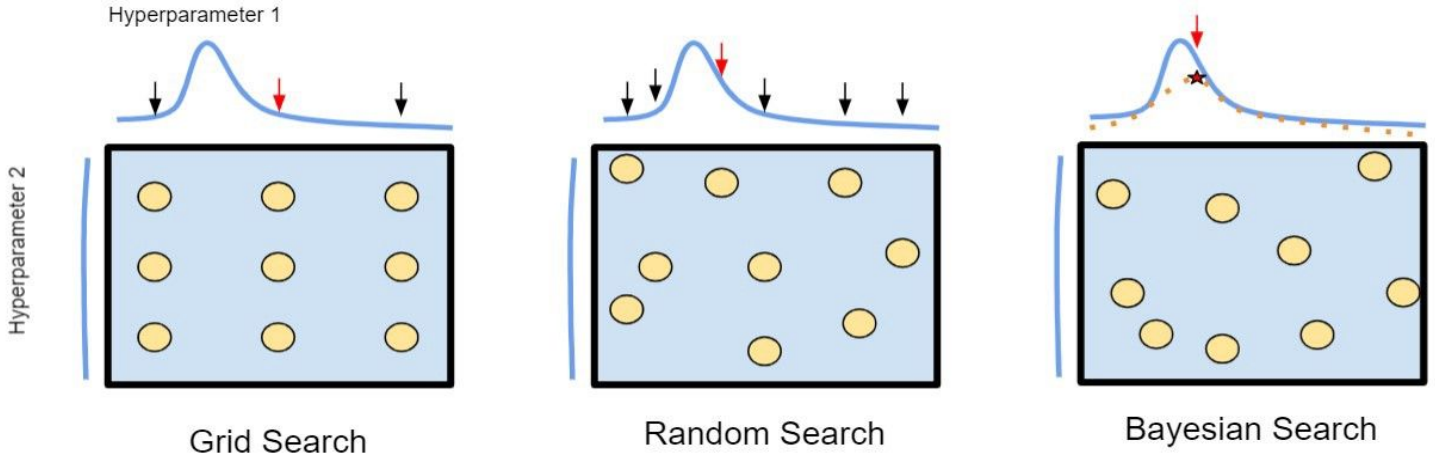
- Parameters which define the model architecture are referred to as **hyperparameters**
- In other words, **hyperparameters** are all the parameters of a model which are not updated during the learning
- This process of searching for the ideal model architecture is referred to as **hyperparameter tuning**
- It is a vital aspect of increasing model performance

In general, this process includes:

1. Define a model
2. Define the range of possible values for all hyperparameters
3. Define a method for sampling hyperparameter values
4. Define an evaluative criteria to judge the model
5. Define a cross-validation method



Machine Learning: Hyperparameters tuning



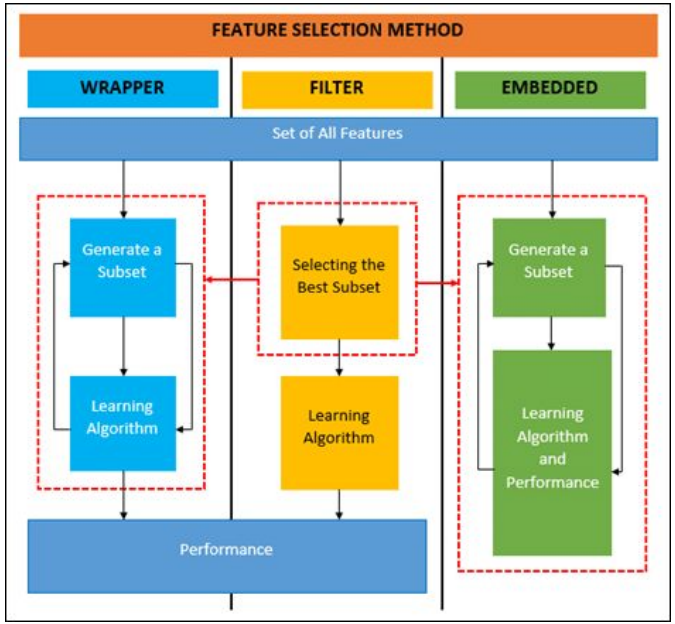
... is simply an exhaustive searching
through a manually specified
subset of the hyperparameter
space of a learning algorithm
computationally expensive
use when hyperparameter search
space is restricted
gridsearch, hyperopt, h2o, tune,
hyperopt-sklearn, hyperas

... replaces the exhaustive
enumeration of all combinations by
selecting them randomly.
is a common alternative to Grid
Search
gridsearch, hyperopt, h2o, tune,
hyperopt-sklearn, hyperas

...is a global optimization method for
noisy black-box functions
Bayesian optimization generates a
function to model the hyperparameter
space, which is then optimized
Bayesian is ideal for rapidly finding
good hyperparameter settings
auto sklearn, hyperopt, optuna,
scikit-optimize, pycaret


Feature selection

Feature selection



Filter

- Relevance of the features is calculated by considering the intrinsic properties of the data
- Use feature relevance score to select the top rank features

Examples

- Information gain
- Correlation coefficient scores
- Chi squared test
- T-test

Advantages

- Can scale to high-dimensional data sets
- Fast and computationally inexpensive in comparison to wrapper methods.

Disadvantages

- No interaction with the classifier
- Univariate feature selection methods do not consider feature dependencies/ redundancy

Wrapper

- Wrapper methods select a subset of relevant features using a learning algorithm
- Conduct search in the space of possible parameters

Examples

- Genetic search
- Exhaustive search
- Sequential forward selection
- Sequential backward elimination

Advantages

- Considers feature dependencies
- Interaction with classifier
- Simple to implement

Disadvantages

- Higher risk of overfitting.
- Selection based on classifier.
- Computationally intensive.

Embedded

- Includes the classifier construction for the optimal feature selection/
- Like wrapper approaches, these methods are specific to a given learning algorithm

Examples

- Decision tree
- Weighted Naive Bayes
- Weighted Vector of SVM

Advantages

- Classifier interaction
- Considers feature dependencies

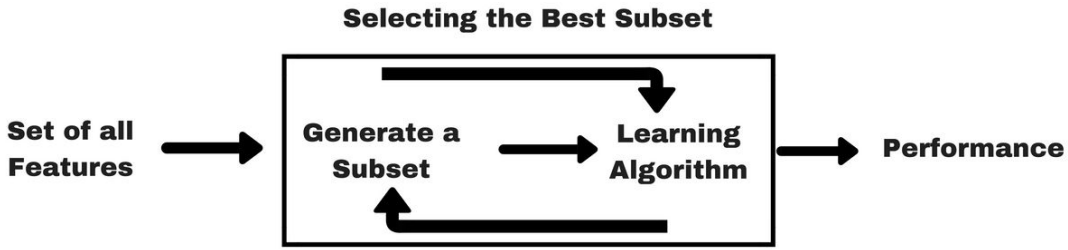
Disadvantages

- Classifier dependencies.

Filter methods

- Filter methods are generally used as a preprocessing step.
- The selection of features is independent of any machine learning algorithms.
- Instead, features are selected on the basis of their scores in various statistical tests for their correlation with the outcome variable.

Wrapper methods



Forward Selection

STEP 1: Select a significance level to enter the model (e.g. SL = 0.05)

STEP 2: Fit all simple regression models $y \sim x_n$ Select the one with the lowest P-value

STEP 3: Keep this variable and fit all possible models with one extra predictor added to the one(s) you already have

STEP 4: Consider the predictor with the lowest P-value. If P < SL, go to STEP 3, otherwise go to FIN

FIN: Keep the previous model

Backward Elimination

STEP 1: Select a significance level to stay in the model (e.g. SL = 0.05)

STEP 2: Fit the full model with all possible predictors

STEP 3: Consider the predictor with the highest P-value. If P > SL, go to STEP 4, otherwise go to FIN

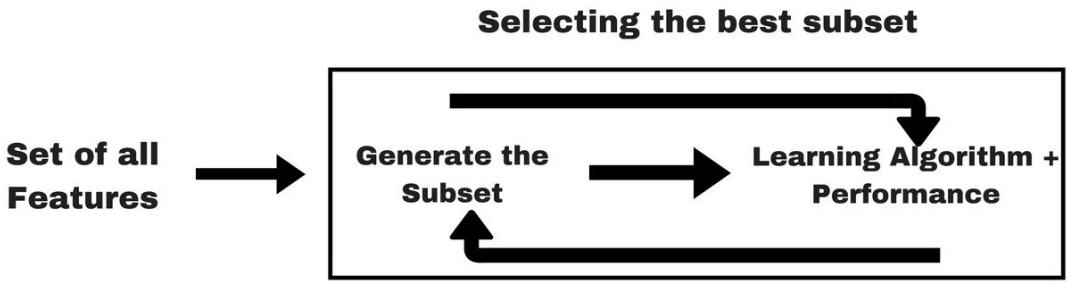
STEP 4: Remove the predictor

STEP 5: Fit model without this variable*

FIN: Your Model Is Ready

Embedded methods

- Embedded methods combine the qualities' of filter and wrapper methods
- It's implemented by algorithms that have their own built-in feature selection methods.
- LASSO and RIDGE regression which have inbuilt penalization functions to reduce overfitting



Machine Learning: regularizations

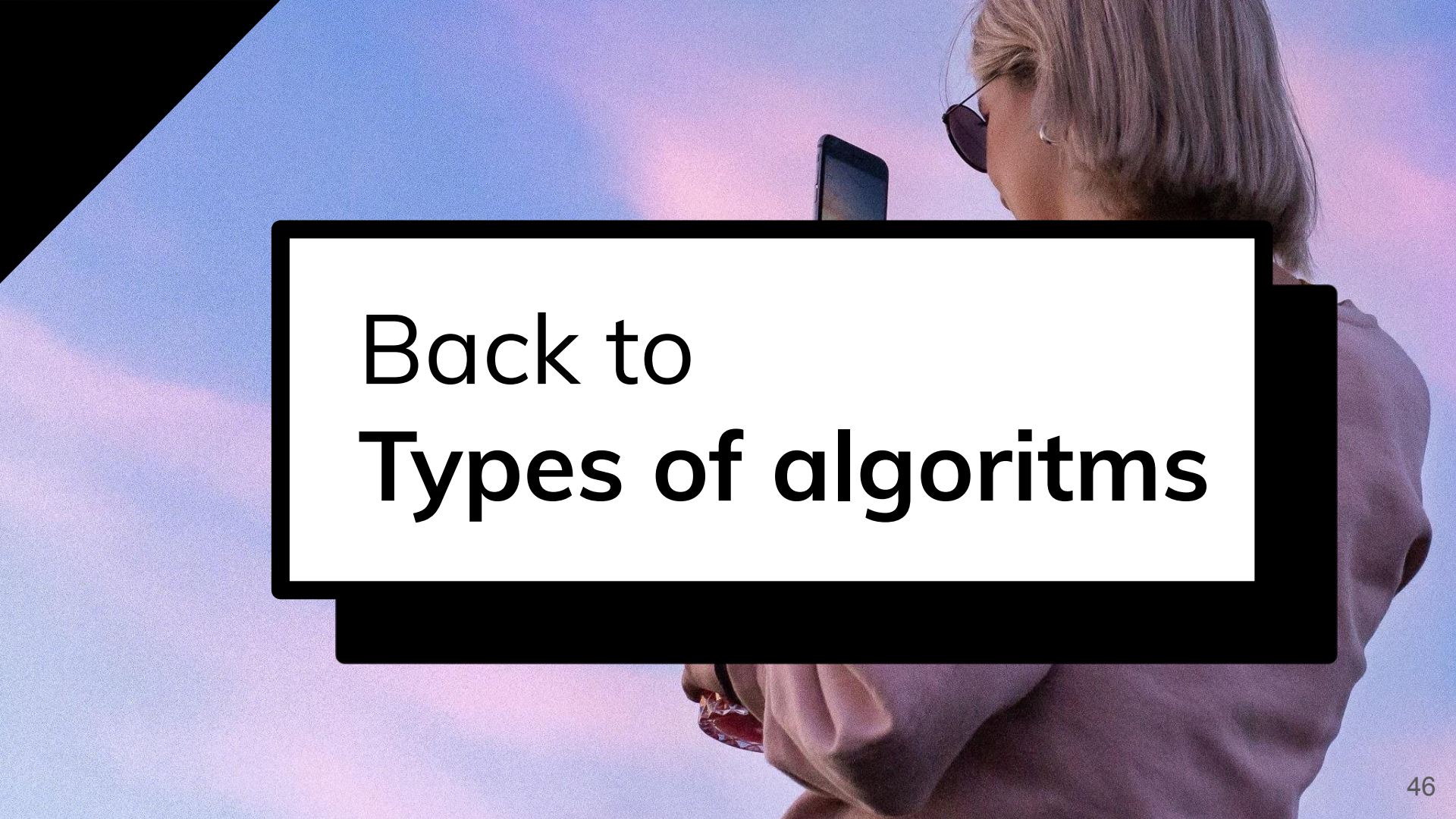
The regularization allows to control simultaneously the model error on the training set and the model complexity

$$\arg \min_{\beta \in \mathbb{R}^{p+1}} (y - X\beta)^{\top}(y - X\beta) + \lambda \quad \text{Regularizer } (\beta)$$

The **regularizer**, which measures the complexity of the model, is a function of the weights β of the model.

λ is the regularization coefficient

The larger λ is, the larger the regularization term. The smaller it is, the larger the error; if it is small enough (and in particular if it is zero), the solution of the unregularized linear regression will be found.



Back to
Types of algoritms

Linear Regression: Ridge or Tikhonov regularization

The regularizer is the l2-norm, does not remove irrelevant features, but minimizes their impact (close to zero)

It is also useful when there are many variables because the probability of having correlated variables is high.

The ridge regression therefore always admits a single explicit solution

$$\underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_2^2}_{\text{Penalty}} \quad \text{where} \quad ||\beta||_2^2 = \sum_{j=0}^{p} \beta_j^2.$$

$$\nabla_\beta \left(||y - X\beta||_2^2 + \lambda ||\beta||_2^2 \right) = 0$$

$$\beta = (\lambda I + X^\top X)^{-1} X^\top y.$$

So, setting λ to 0 is the same as using the classical LS, while the larger its value, the stronger is the coefficients' size penalized.

Ridge regression is highly affected by the **scale of the predictors**. Therefore, it is better to standardize (i.e., scale) the predictors before applying the ridge regression (James et al. 2014)

Linear Regression: Lasso (Least Absolute Shrinkage and Selection Operator)

The only difference in ridge and lasso loss functions is in the penalty terms

The regularizer is the **l1-norm**, the sum of the absolute coefficients.

l1-norm is sometimes known as the Manhattan norm

This is useful when the variables are highly correlated

The penalty has the effect of forcing some of the **coefficient estimates, with a minor contribution** to the model, to be **exactly equal to zero**

$$\arg \min_{\beta \in \mathbb{R}^{p+1}} ||y - X\beta||_2^2 + \lambda ||\beta||_1 \qquad \text{where} \quad ||\beta||_1 = \sum_{j=1}^{p} |\beta_j|$$

The lasso does not have an explicit or necessarily unique solution! Fortunately, we can use a gradient algorithm to solve it

It can perform variable selection in order to reduce the complexity of the model.

Linear Regression: Elastic-net

Elastic Net first emerged as a result of critique on:

- Lasso, whose variable selection (parsimonious model) can be too dependent on data and thus unstable
- Ridge, using the l2 norm allows us to avoid overfitting with a single solution

The solution is to combine the penalties of ridge regression and lasso to get the best of both worlds

It's a solution to the problem of overfitting models

$$\arg\min_{\beta \in \mathbb{R}^{p+1}} ||y - X\beta||_2^2 + \lambda \left((1-\alpha)||\beta||_1 + \alpha||\beta||_2^2 \right)$$

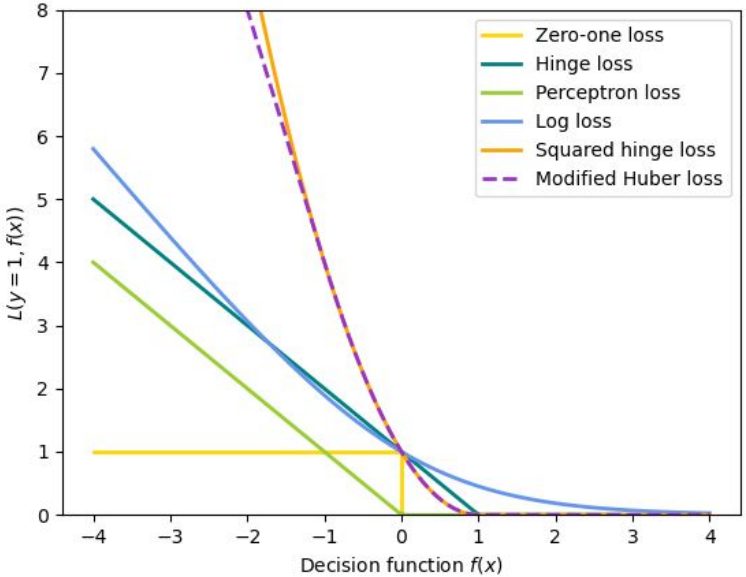
The consequence of this is to effectively **shrink coefficients** (like in ridge regression) and to set some **coefficients to zero** (as in LASSO).

Elastic-net provides a less parsimonious solution than the lasso, but more stable.

Stochastic Gradient Descent (SGD)

The gradient descent algorithm is an approximate and iterative method for mathematical optimization. It can be used it to approach the minimum of any differentiable function.

The cost function, or loss function, is the function to be minimized (or maximized) by varying the decision variables

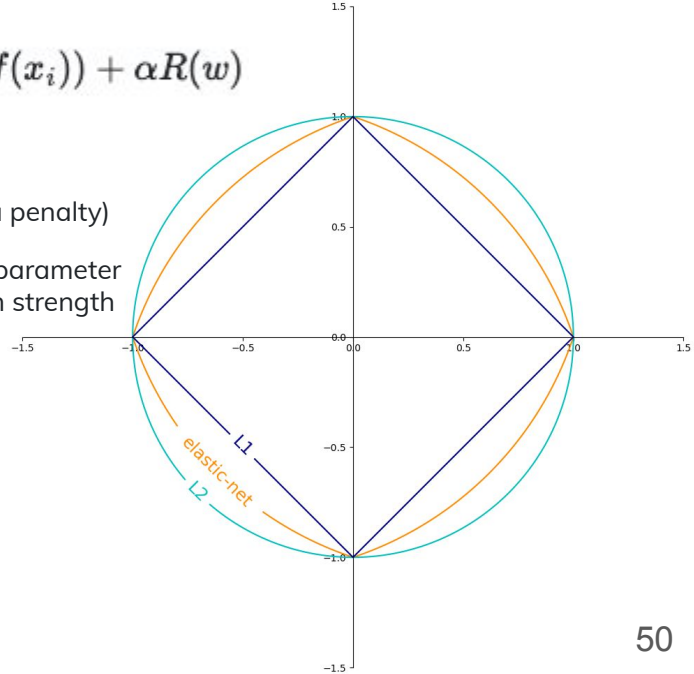


$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

L: loss function

R: is a regularization term (aka penalty)

α > 0: is a non-negative hyperparameter that controls the regularization strength



Stochastic Gradient Descent (SGD)

SGD is a simple yet very efficient approach to fitting linear classifiers and regressors

SGD is merely an optimization technique and does not correspond to a specific family of machine learning models. It is only a way to train a model

The **advantages** of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The **disadvantages** of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

Nearest Neighbors (KNN)

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label/average from these

1. Define the number of samples can be a user-defined constant (k-nearest neighbor learning)
2. Find the k nearest neighbors. The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

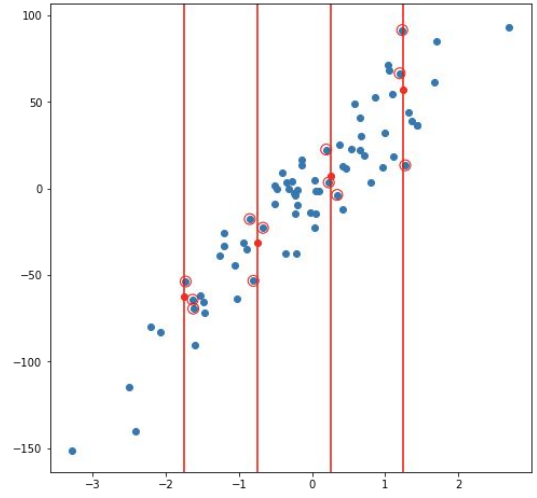
   weights = 'uniform', assigns equal weights to all points

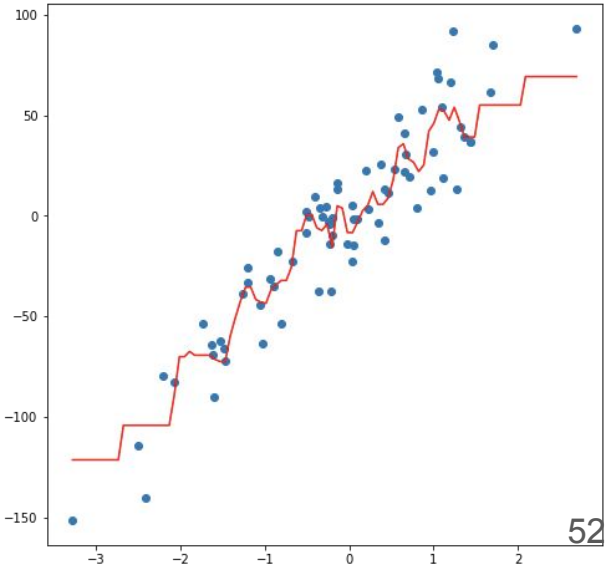
   weights = 'distance' assigns weights proportional to the inverse of the distance from the query point

   Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

   Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

3. Predict the average value





Decision Trees (DTs)

It predicts the value of a target variable by learning **simple decision rules** inferred from the data features.

Decision tree algorithms **decompose the data set** by asking questions until they have r**educed the data sufficiently to make a prediction**.

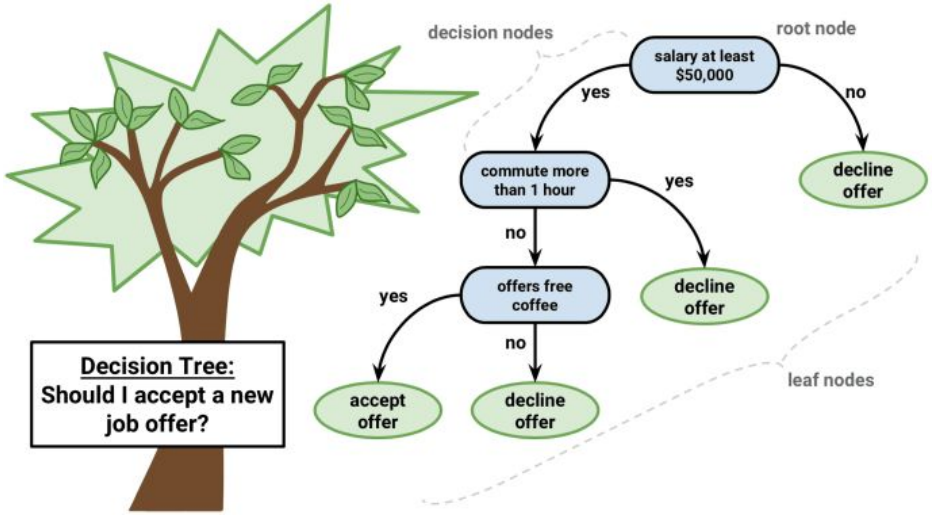
DTs are a non-parametric supervised learning method. A tree can be seen as a piecewise constant approximation

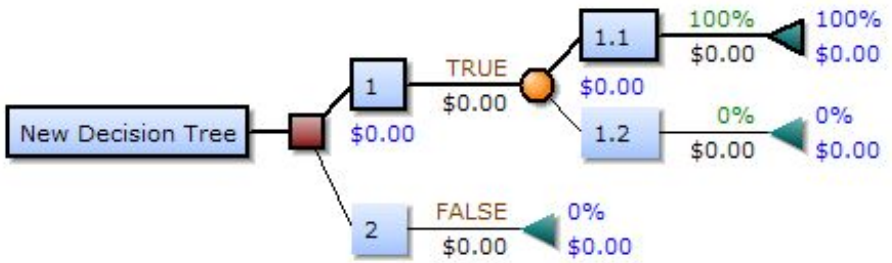
Simple to understand and to interpret. Trees can be visualised.

Requires little data preparation. Uses a white box model.

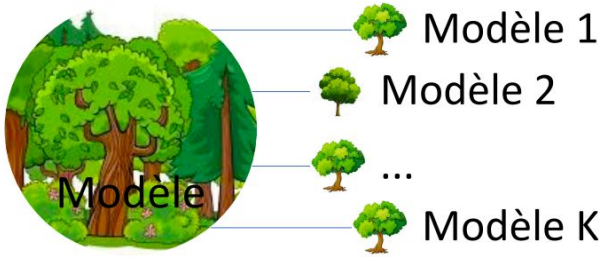
A decision tree consists of three types of nodes:

- Decision nodes – typically represented by squares
- Chance nodes – typically represented by circles
- End nodes – typically represented by triangles





Ensemble methods



The goal of **ensemble methods** is to combine the predictions of several base estimators (weak learners) built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Categories of Ensemble Methods

- **Sequential ensemble techniques** generate base learners in a sequence, e.g., Adaptive Boosting (AdaBoost). The sequential generation of base learners promotes the dependence between the base learners. The performance of the model is then improved by assigning higher weights to previously misrepresented learners.
- In **parallel ensemble techniques**, base learners are generated in a parallel format, e.g., random forest. Parallel methods utilize the parallel generation of base learners to encourage independence between the base learners. The independence of base learners significantly reduces the error due to the application of averages.

Ensemble methods aim at improving predictability in models by combining several models to make one very reliable model.

The most popular ensemble methods are **boosting, bagging, and stacking.**

Ensemble methods are ideal for regression and classification, where they reduce bias and variance to boost the accuracy of models.

Ensemble methods - Focus on Bagging

Bagging consists in fitting several base models on different bootstrap samples and build an ensemble model that “average” the results of these weak learners.

Bagging, the short form for **bootstrap and aggregation** (Breiman, 1996)

- **Bootstrapping** is a sampling technique where samples are derived from the whole set using the replacement procedure
- **Aggregation** in bagging is done to incorporate all possible outcomes of the prediction and randomize the outcome

initial dataset

bootstrap samples + selected features

deep trees fitted on each bootstrap sample and considering only selected features

random forest (kind of average of the trees)

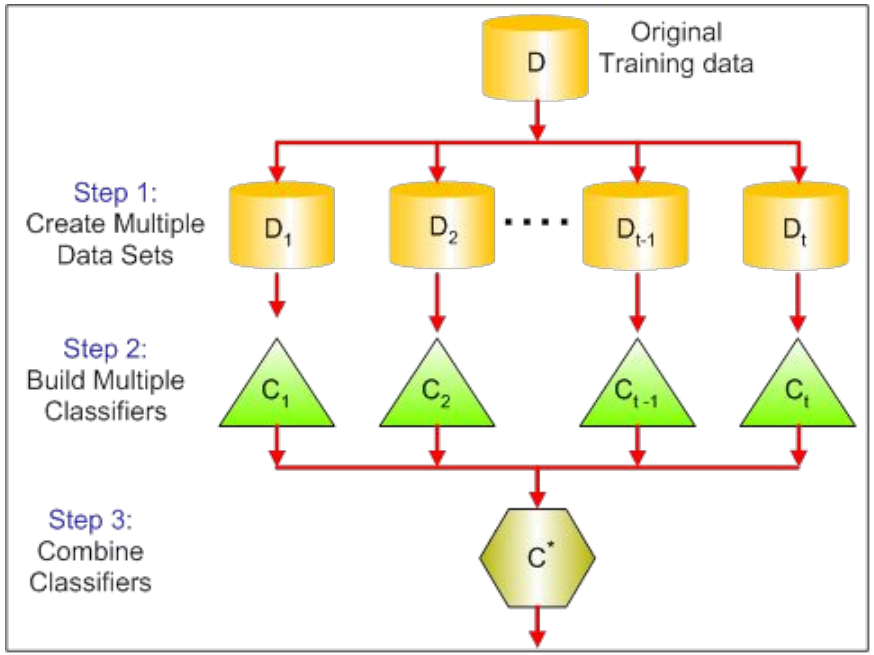
Ensemble methods - Focus on Bagging

Main Steps involved in bagging are :

- **Creating multiple datasets**: Sampling is done with a replacement on the original data set and new datasets are formed from the original dataset.
- **Building multiple classifiers:** On each of these smaller datasets, a classifier is built, usually, the same classifier is built on all the datasets.
- **Combining Classifiers:** The predictions of all the individual classifiers are now combined to give a better classifier, usually with very less variance compared to before.

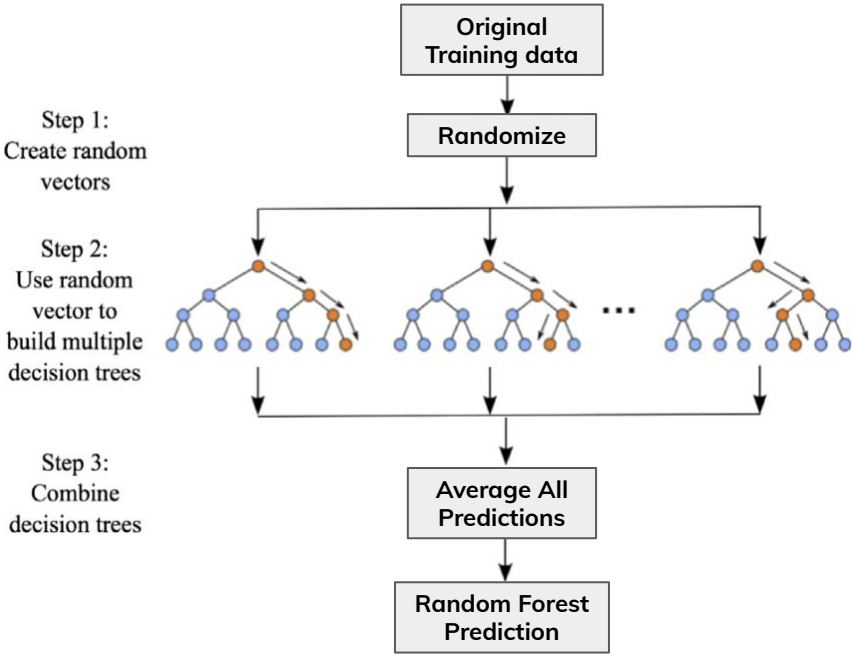
Common Bagging algorithms:

- Bagging methods
- Forests of randomized trees
- Extremely Randomized Trees



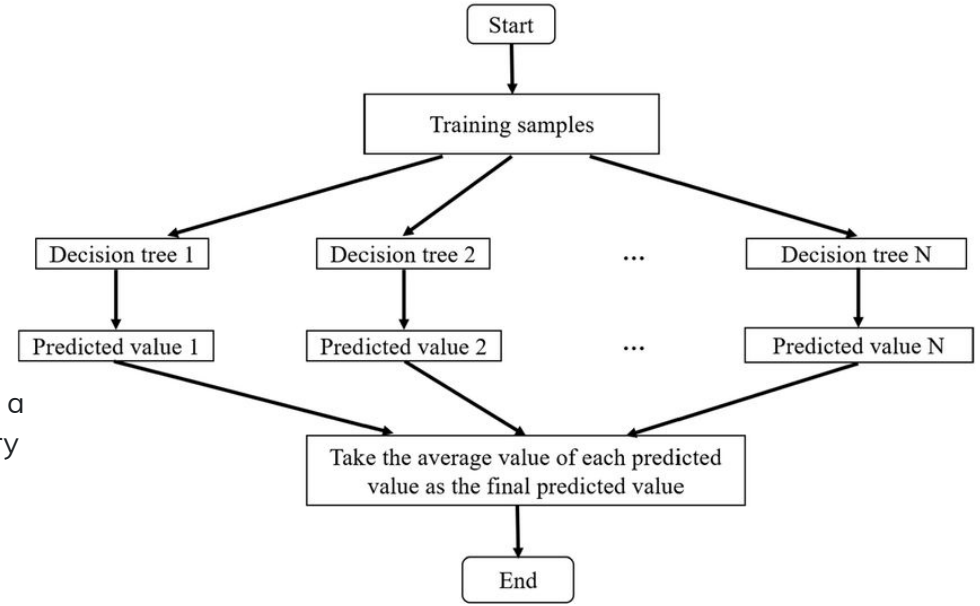
Ensemble methods - Random Forest

- **Random forest** is a **bagging** technique and **not a boosting** technique. The trees in **random forests** are run in parallel. There is no interaction between these trees while building the trees.
- It operates 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)** of the individual trees
- Random forest models reduce the risk of overfitting by introducing randomness by
    - building multiple trees (n_estimators)
    - drawing observations with replacement (i.e., a bootstrapped sample)
    - splitting nodes on the best split among a random subset of the features selected at every node. Split is process to convert non-homogeneous parent node into 2 homogeneous child node (best possible).



Ensemble methods - Extremely Randomized Trees (ExtraTrees)

- Extra Trees is like a Random Forest, in that it builds multiple randomized decision trees and splits nodes using random subsets of features
- This class uses averaging to improve the predictive accuracy and control over-fitting.
- So in summary, ExtraTrees:
  - builds multiple trees without bootstrap, which means it samples without replacement
  - nodes are split based on random splits among a random subset of the features selected at every node



Ensemble methods - Random Forest vs ExtraTrees

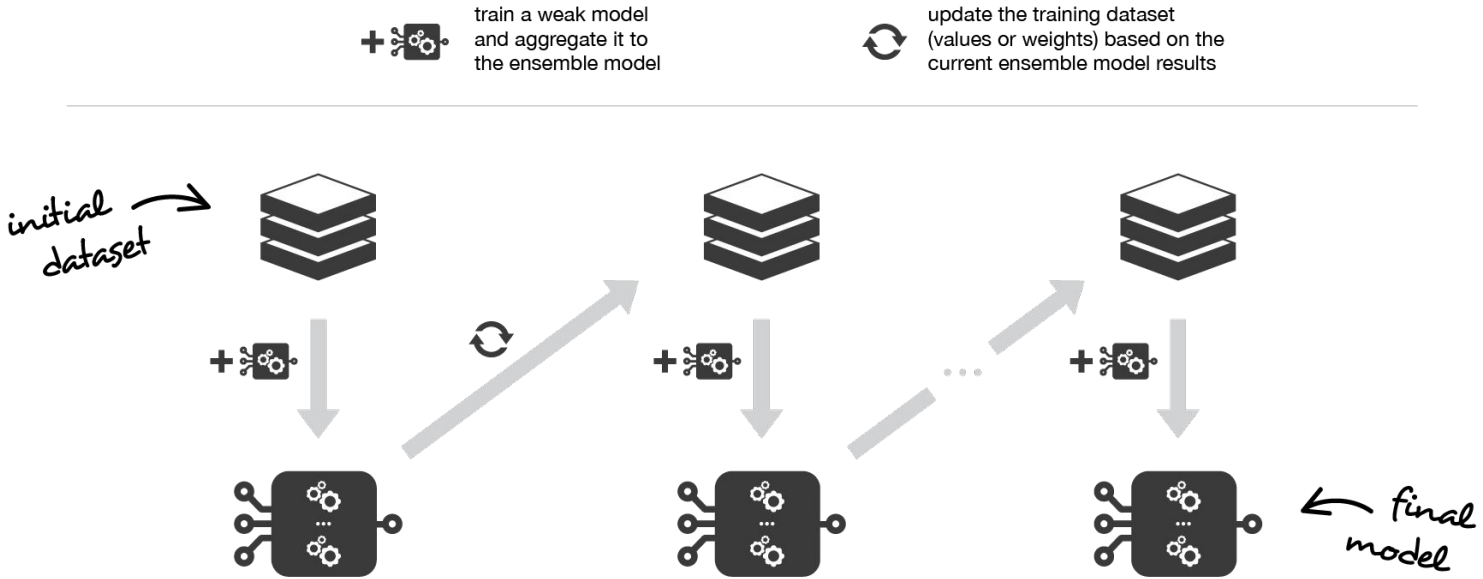
	Decision Tree	Random Forest	Extra Trees
Number of trees	1	Many	Many
No of features considered for split at each decision node	All Features	Random subset of Features	Random subset of Features
Boostrapping(Drawing Sampling without replacement)	Not applied	Yes	No
How split is made	Best Split	Best Split	Random Split

Ensemble methods - Focus on Boosting

In **sequential methods** the different combined weak models are no longer fitted independently from each others.

The idea is to fit models iteratively such that the training of model at a given step depends on the models fitted at the previous steps

Boosting consists in, iteratively, **fitting a weak learner,** aggregate it to the ensemble model and "update" the training dataset to better take into account the strengths and weakness of the current ensemble model when fitting the next base model. In other words, it learns from previous predictor mistakes to make better predictions in the future



Ensemble methods - Focus on Boosting

Boosting algorithm iteratively learns weak classifiers and add them to a final strong classifier

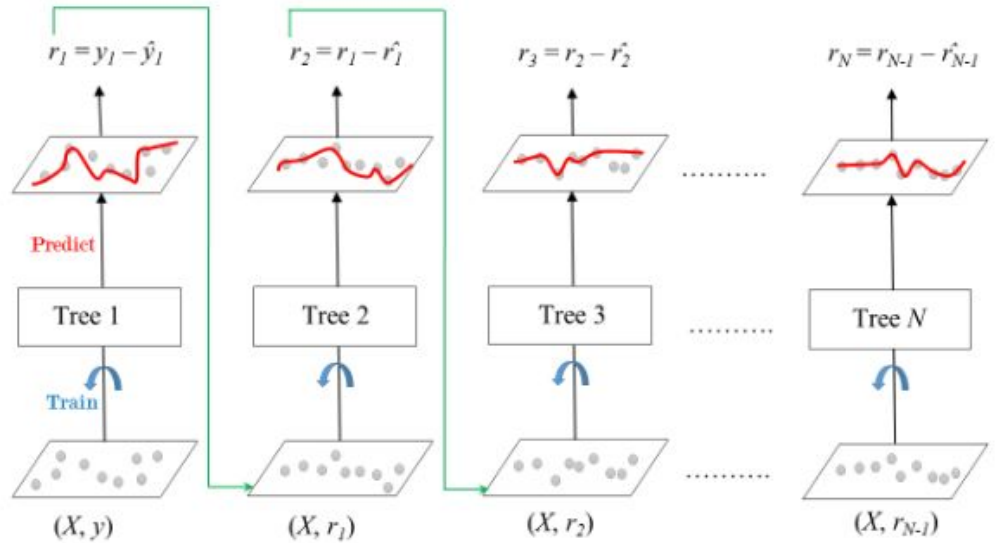
Main Steps involved in boosting are :

- Train model A on the whole set
- Train the model B with exaggerated data on the regions in which A performs poorly
- ...

Instead of training the models in *parallel*, we can train them *sequentially*. This is the main idea of Boosting!

Common Boosting algorithms:

- AdaBoost (Adaptative boosting)
- Gradient Boosting
- XGBoost (eXtreme Gradient Boosting)
- LightGBM
- CatBoost

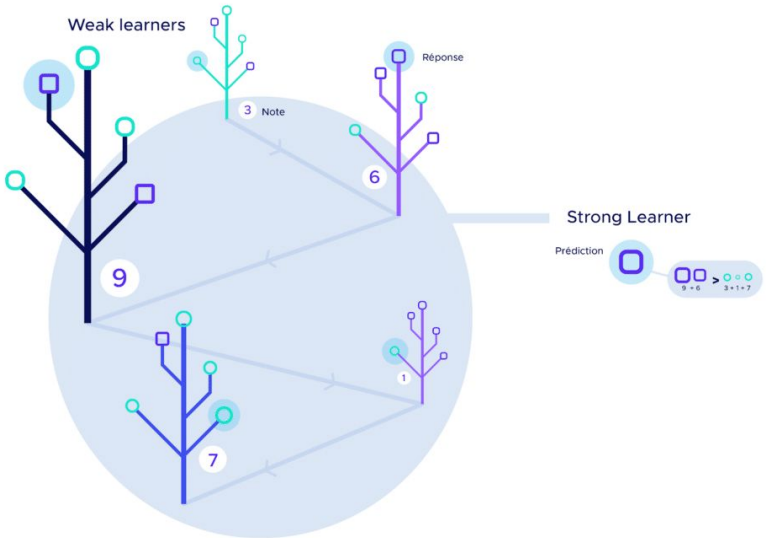


Ensemble methods - AdaBoost

Adaboost was first used by Yoav Freund and Robert Schapire, and won the Gödel Prize in 2003

Pseudocode: (Zhu, H. Zou, S. Rosset, T. Hastie, "Multi-class AdaBoost", 2009)

$$\text{for a weak classifier } C$$
$$X: n \times d, Y: n \times k \text{ with sample weights } W: n \times 1$$
$$Error = \frac{\sum_{j=1}^{n} W_j I(C(X_j) \neq Y_j)}{\sum_{j=1}^{n} W_j}, I(x) = \begin{cases} 1, & \text{if } x \text{ is True} \\ 0, & \text{if } x \text{ is False} \end{cases}$$



$$\text{Initial weights } W: W_1, W_2, \ldots, W_n = \frac{1}{n}$$
$$\text{for } i \text{ in } [1, M] \text{// M weak classifiers}$$
$$\text{fit weak classifier } C^i \text{ with sample weights } W$$
$$Error^i = \frac{\sum_{j=1}^{n} W_j I(C^i(X_j) \neq Y_j)}{\sum_{j=1}^{n} W_j}$$
$$\alpha^i = \log\left(\frac{(1 - Error^i)}{Error^i}\right) + \log(K - 1) \text{//coefficient for } C^i$$
$$W_j = W_j * e^{\alpha^i * I(C^i(X_j) \neq Y_j)} \text{ for } W_j \in W \text{// if wrong, increase weights}$$
$$W = W - mean(W) \text{// normalize weights}$$
$$\text{//output of the model is done by weighted voting, find the class with highest vote}$$
$$\text{Prediction: } \hat{Y}_j = \max_k \left(\sum_{i=1}^{i} \alpha_i I(C^i(X_j) = k)\right)$$

Ensemble methods - Gradient Boosting

Similar to Adaboost, but the "weak learners" all have equal weight in the voting system, regardless of their performance.

The creation of the weak learners always follows the same pattern:

- The first "weak learner" (w1) is very basic, it is simply the average of the observations.
- From the last predictions, we calculate the new residuals (difference between reality and the prediction)
- We train the new "weak learner" to predict these residues
- We multiply the predictions of this "weak learner" by a factor less than 1
- We obtain new predictions, often slightly better than the previous ones

$$\text{To improve a model } F\text{: We want to minimize } Loss(Y, F(X))$$
$$\text{The loss function } L = func(F(X_1), F(X_2), \dots, F(X_n), Y)$$
$$\text{Minimization is performed by fitting an estimator } H \text{ on } \left(X_i, \frac{\partial L}{\partial X_i}\right) \forall i$$
$$F(X) + H(X) \text{ is an approximation of gradient descent } \hat{F}(X_i) = F(X_i) - \frac{\partial L}{\partial F(X_i)}$$

$$\text{fit estimator } F^1$$
$$\text{for } i \text{ in } [1, M] \text{//M weak estimators}$$
$$Loss^i = \sum_{j=1}^{n} \left(Y_j - F^i(X_j)\right)^2 \text{ // loss in } i^{th} \text{ iteration}$$
$$\text{calculate neg gradient: } -\frac{\partial L^i}{\partial X_j} = -\frac{2}{n} * \left(Y_j - F^i(X_j)\right) \forall i$$
$$\text{Fit a weak estimator } H^i \text{ on } \left(X, \frac{\partial L}{\partial X}\right)$$
$$\text{//}\rho \text{ changes the step size}$$
$$\text{Prediction: } F^m(X) = F^i(X) + \rho * H^i(X) = F^1 + \rho * \sum_{i=1}^{m} H^i(X)$$

Gradient Boosting with Least Square

Ensemble methods - XGBoost

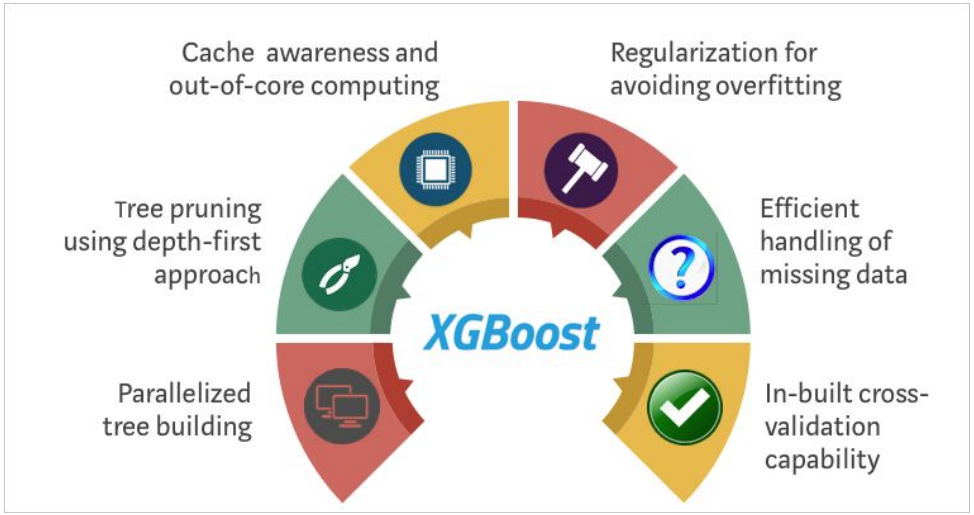
XGBoost is in fact a particular version of the Gradient Boost algorithm

Trees that are not good enough are "pruned", i.e. branches are cut off, until they are good enough. Otherwise they are completely removed (**pruning**)

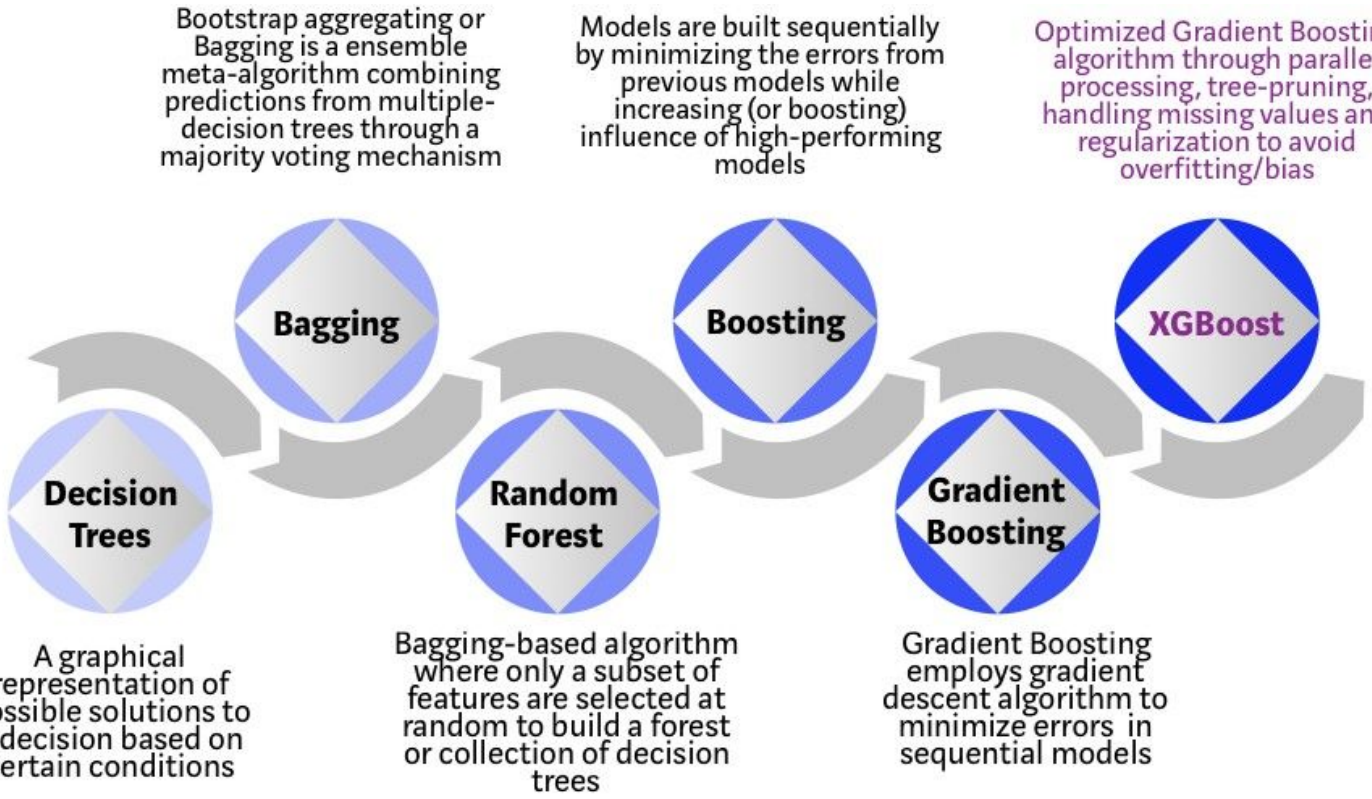
This way, XGBoost makes sure to keep only good weak learners.

XGBoost is often the **winning algorithm in Kaggle** competitions, it is fast, **accurate** and **efficient**, allowing a **flexibility** of maneuvering unseen on the Gradient Boosting.

Nevertheless, **each problem has a different solution.**



Ensemble methods - Summary

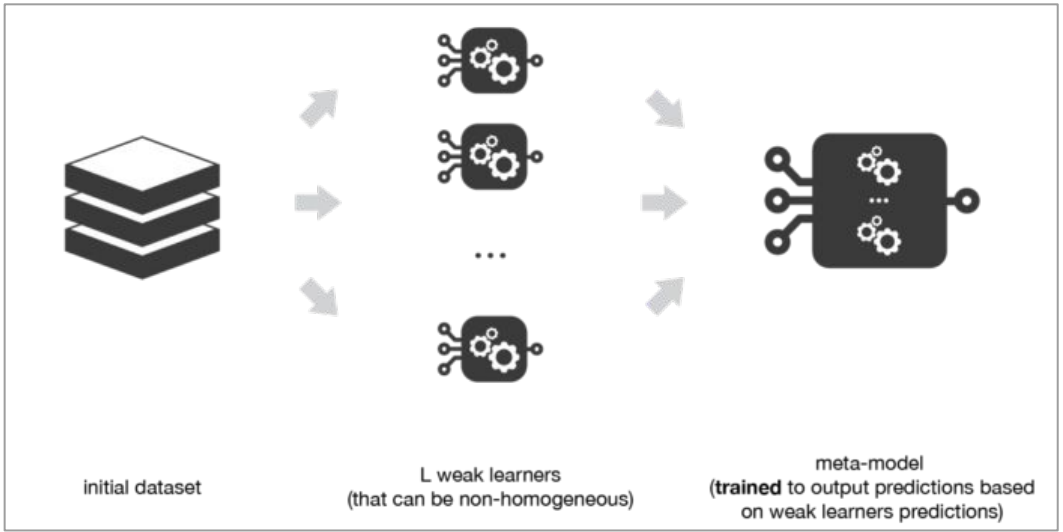


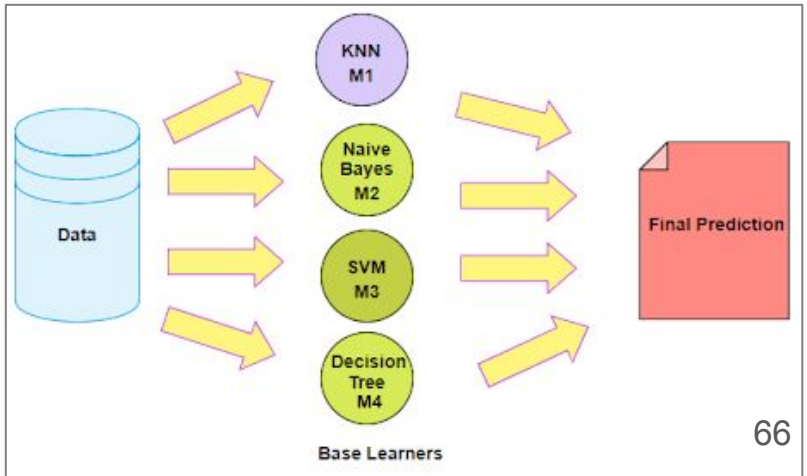
Ensemble methods - Focus on Stacking

Stacking mainly differ from bagging and boosting on two points

- First stacking often considers **heterogeneous weak learners** (different learning algorithms are combined) whereas bagging and boosting consider mainly homogeneous weak learners.
- Second, stacking learns to combine the base models using a meta-model whereas bagging and boosting combine weak learners following deterministic algorithms.

The idea of stacking is to learn several different weak learners and combine them by training a meta-model

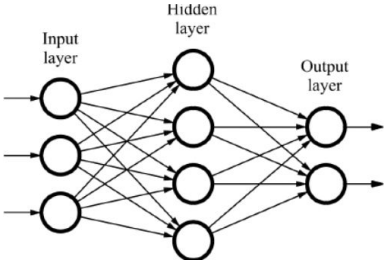




Neural network - Architectures

Neural networks, also known as Artificial Neural network use different deep learning algorithms.

There are 3 most common neural network architectures every Deep Learning practitioner must be aware of



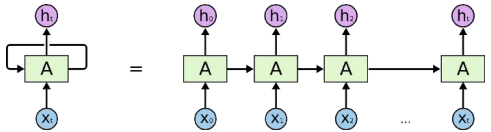
Feed Forward Neural Network

Most common type of neural network.

It consists of an input layer; an output layer and in between, we have some hidden layers.

If there are multiple hidden layer, its a “deep” neural networks

They perform a series of transformations on the input (non-linear function & full connectivity)



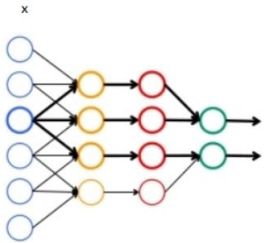
Recurrent Neural Network (RNN)

Can be used to model sequences, speech recognition and natural language processing (NLP: text generation, machine translation, ...)

They have directed cycles in the connection graph

They can remember information in the hidden states (but its very hard to train them to use this potential.)

The output being depended on the previous computations



Convolutional Neural Network (CNN)

A CNN has several layers through which data is filtered into categories

CNNs are regularized versions of multilayer perceptrons

CNNs have proven to be very effective in areas such as image recognition, text language processing, and classification

Hidden layers include multiple convolutional layers, pooling layers, fully connected layers, and normalization layers

Neural network - Multi-layer Perceptron (MLP)

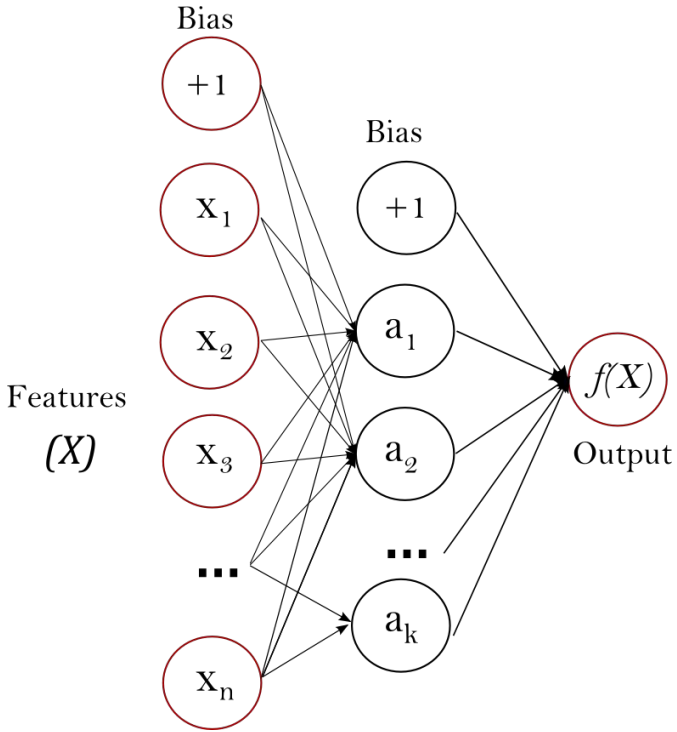
A **multilayer perceptron** (**MLP**) is a class of feedforward artificial neural network (ANN)

Artificial neurons seek to mimic the functioning of brain neurons

An MLP consists of at least three layers of nodes: an input layer, a hidden layers and an output layer.

There can be one or more non-linear layers, called hidden layers

1. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation
2. followed by a non-linear activation function (f)
3. The output layer receives the values from the last hidden layer and transforms them into output values.



One hidden layer (MLP)

Neural network - Multi-layer Perceptron (MLP)

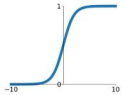
Except for the input nodes, each node is a neuron that uses a nonlinear activation function.

MLP utilizes a supervised learning technique called backpropagation for training

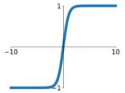
It can distinguish data that is not linearly separable

Activation Functions

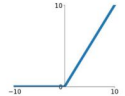
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



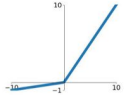
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

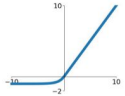


Leaky ReLU
$\max(0.1x, x)$

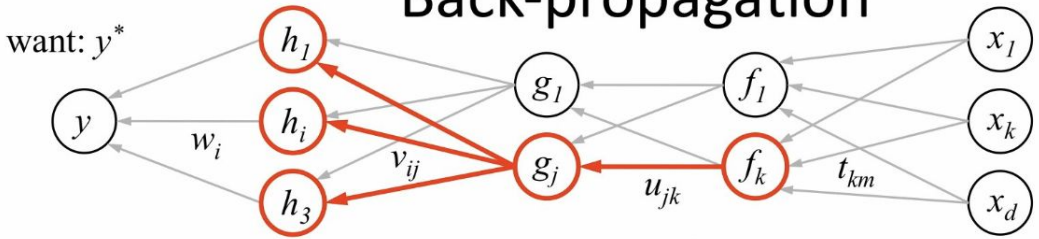


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Back-propagation



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y\text{-}y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j

$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Neural network - Multi-layer Perceptron (MLP)

Input Layer

The bottom layer that takes input from your dataset is called the visible layer

Hidden Layers

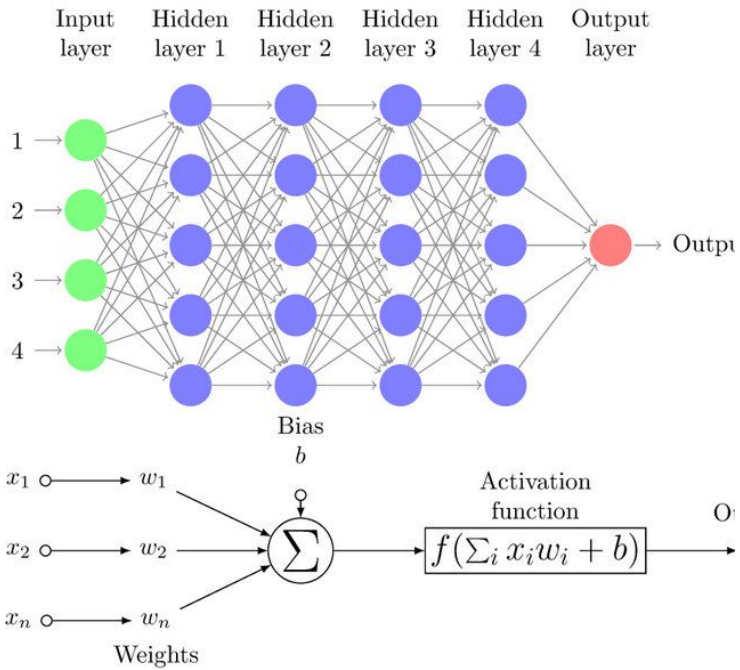
Layers after the input layer are called hidden layers because that are not directly exposed to the input.

Output Layer

The final hidden layer is called the output layer and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

The advantages of Multi-layer Perceptron are:

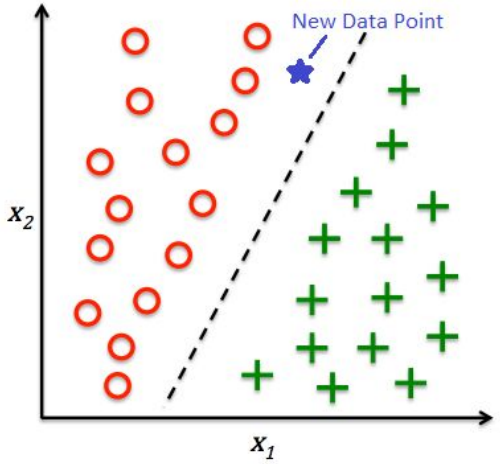
- Capability to learn non-linear models.
- Capability to learn models in real-time

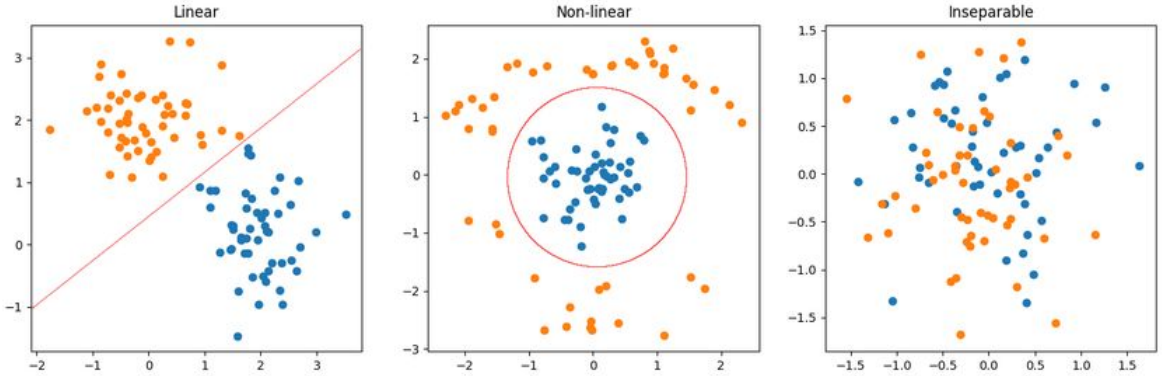


II. Classification

What is Classification problem ?

- It classifies data into different parts/classes/groups
- Target variable is categorical
- Classification is the process of assigning new input variables (X) to the class they most likely belong to





Type of classification tasks

Binary classification

Two class labels (0 or 1)

Examples:

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not)
- Medical Diagnosis (healthy or Disased)

- Spam
- Not spam

Multiclass classification

> 2 class labels

One-vs-Rest: Fit one binary classification model for each class vs. all other classes.

One-vs-One: Fit one binary classification model for each pair of classes.

Examples:

- Face classification.
- Plant species classification.
- Optical character recognition

- Dog
- Cat
- Horse
- Fish
- Bird
- ...

Multilabel classification

two or more class labels, where one or **more class labels may be predicted** for each example

Examples:

- Face detection
- Object detection
- Text categorisation

- Dog
- Cat
- Horse
- Fish
- Bird
- ...

Imbalanced classification

It refers to classification tasks where the number of examples in each class is **unequally distributed**

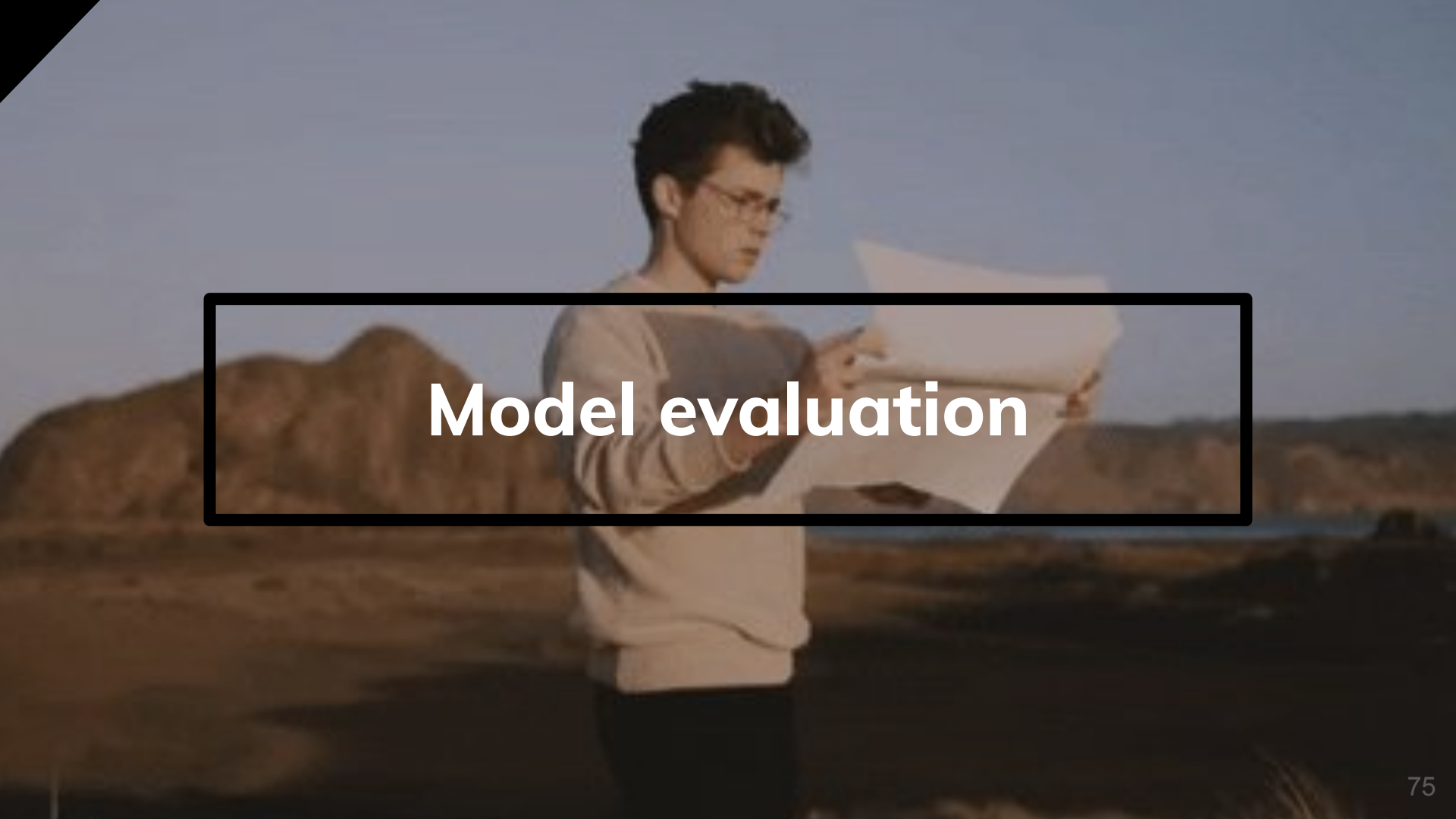
> *"I am working on a classification model. In my dataset I have **three different labels** to be classified, let them be **A, B and C.** But in the training dataset I have **A dataset with 70% volume, B with 25% and C with 5%.** Most of time my results are overfit to A. Can you please suggest how can I solve this problem?"*

Examples:

- Fraud detection.
- Outlier detection.
- Medical diagnostic tests.

Specialized techniques may be used to change the composition of samples in the training dataset by undersampling the majority class or oversampling the minority class:

- Resampling: oversampling, undersampling
- Can You Collect More Data
- Changing Your Performance Metrics

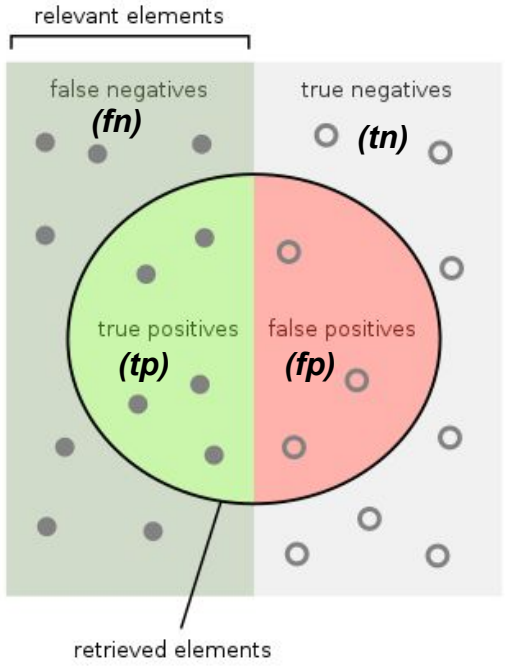

Model evaluation

Model evaluation

For classification tasks, the terms ***true positives***, ***true negatives***, ***false positives***, **and** ***false negatives*** (see Type I and type II errors for definitions) compare the results of the classifier under test with trusted external judgments. The terms ***positive and negative refer*** to the classifier's prediction (sometimes known as the *expectation*), and the terms ***true and false*** refer to whether that prediction **corresponds to the external judgment** (sometimes known as the *observation*)

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

Confusion matrix



Model evaluation: Accuracy

The fraction (default) or the count (normalize=False) of correct predictions.

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

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

		True Class	
		Positive	**Negative**
Predicted Class	**Positive (P)**	True Positive (TP)	False Positive (FP)
	Negative (N)	False Positive (FP)	True Negative (TN)

Accuracy can be a misleading metric for imbalanced data sets. Consider a sample with **95 negative and 5 positive values.** Classifying all values as negative in this case gives **0.95 accuracy score**.

Model evaluation: Balanced Accuracy

In the binary case, balanced accuracy is equal to the arithmetic mean of sensitivity (True Positive Rate, recall) and specificity (selectivity, True Negative Rate)

$$\texttt{balanced-accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

$$= \frac{TPR+TNR}{2}$$

		True Class	
		Positive	**Negative**
Predicted Class	**Positive**	True Positive (tp)	False Positive (fp)
	Negative	False Positive (fp)	True Negative (tn)

Accuracy can be a misleading metric for imbalanced data sets. Consider a sample with **95 negative and 5 positive values.** Classifying all values as negative in this case gives **0.95 accuracy score**.

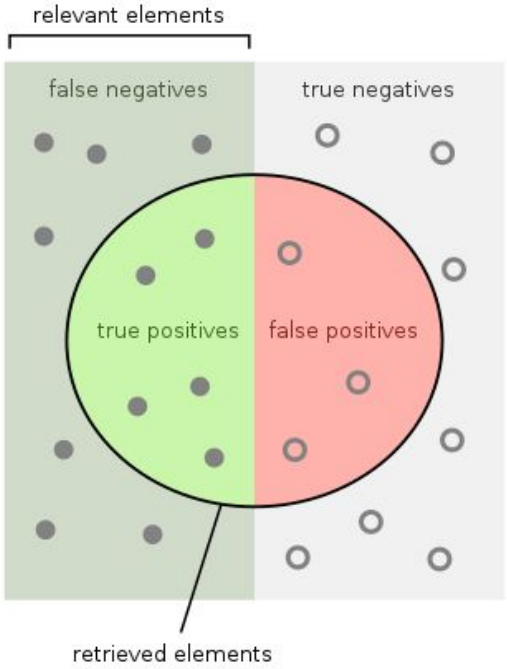
Model evaluation: Top-k Accuracy

This metric computes the **number of times where the correct label is among the top k labels predicted** (ranked by predicted scores)

$$\texttt{top-k accuracy}(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \sum_{j=1}^{k} 1(\hat{f}_{i,j} = y_i)$$

If $\hat{f}_{i,j}$ is the predicted class for the i-th sample corresponding to the j-th largest predicted score and y_i is the corresponding true value, then the fraction of correct predictions over n_{samples} is defined as

Model evaluation: Recall, precision, F_β score



Precision is the estimated **probability that a document randomly selected** from the pool of **retrieved documents is relevant**.

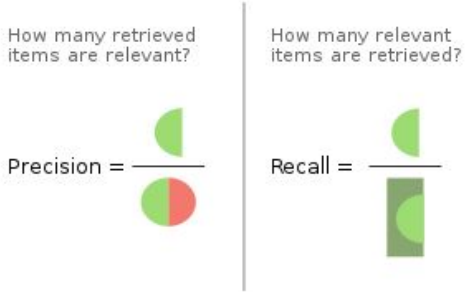
Recall is the estimated **probability that a document randomly selected** from the pool of **relevant documents is retrieved**.

$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}$$

The traditional F-score or balanced **F-score** (F$_1$ score) is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

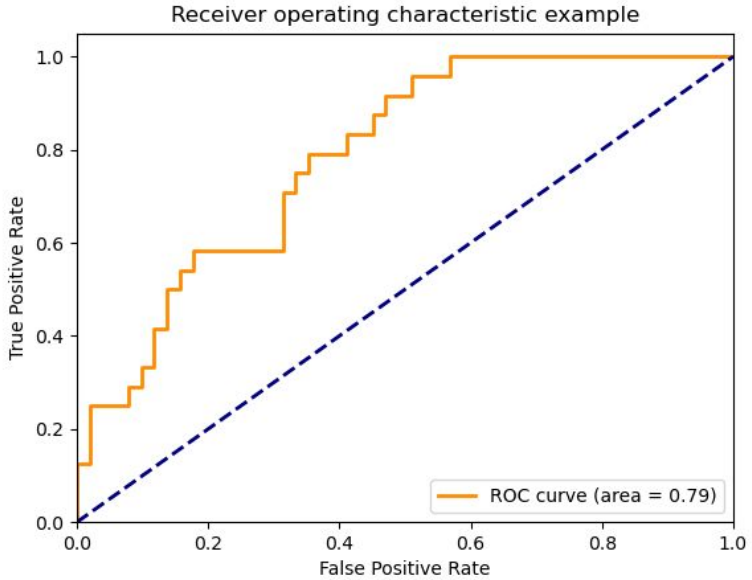


Model evaluation: Area Under the Receiver Operating Characteristic Curve

computes the area under the receiver operating characteristic (ROC) curve

It is created by plotting the fraction of TPR vs. FPR, at various threshold settings

TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.”



Gini coefficient = **2 * AUC - 1**

Model evaluation - Multiclass Aggregate metrics

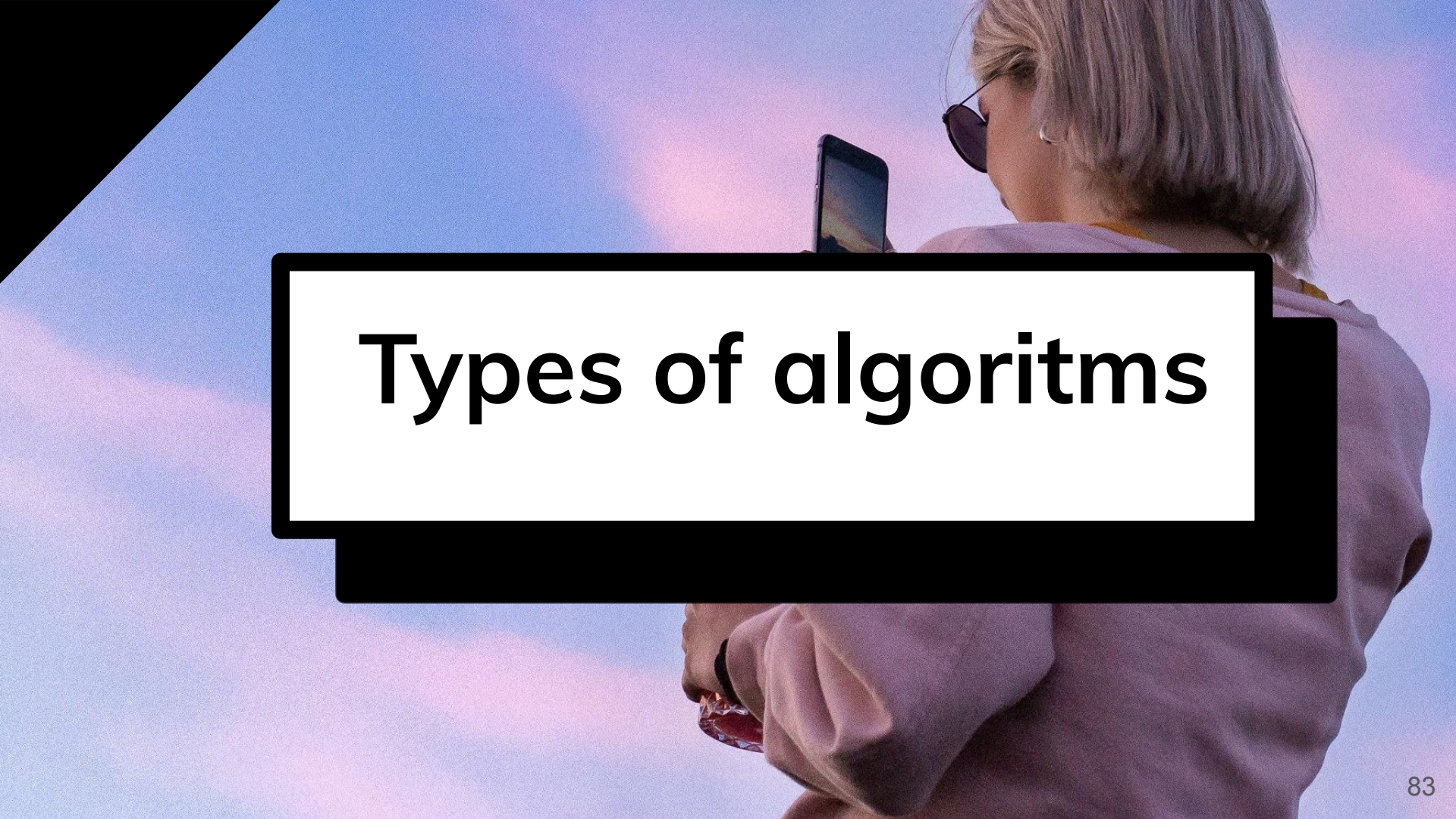
- **micro**: Calculate metrics globally by counting the total number of times each class was correctly predicted and incorrectly predicted.
- **samples:** Calculate metrics for each instance, and find their average (only meaningful for multilabel classification).
- **macro**: Calculate metrics for each "class" independently, and find their unweighted mean. This does not take label imbalance into account.
- **weighted**: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label).

average	Precision	Recall	F_beta
"micro"	$P(y, \hat{y})$	$R(y, \hat{y})$	$F_\beta(y, \hat{y})$
"samples"	$\frac{1}{\lvert S\rvert}\sum_{s\in S} P(y_s, \hat{y}_s)$	$\frac{1}{\lvert S\rvert}\sum_{s\in S} R(y_s, \hat{y}_s)$	$\frac{1}{\lvert S\rvert}\sum_{s\in S} F_\beta(y_s, \hat{y}_s)$
"macro"	$\frac{1}{\lvert L\rvert}\sum_{l\in L} P(y_l, \hat{y}_l)$	$\frac{1}{\lvert L\rvert}\sum_{l\in L} R(y_l, \hat{y}_l)$	$\frac{1}{\lvert L\rvert}\sum_{l\in L} F_\beta(y_l, \hat{y}_l)$
"weighted"	$\frac{1}{\sum_{l\in L}\lvert\hat{y}_l\rvert}\sum_{l\in L} \lvert\hat{y}_l\rvert P(y_l, \hat{y}_l)$	$\frac{1}{\sum_{l\in L}\lvert\hat{y}_l\rvert}\sum_{l\in L} \lvert\hat{y}_l\rvert R(y_l, \hat{y}_l)$	$\frac{1}{\sum_{l\in L}\lvert\hat{y}_l\rvert}\sum_{l\in L} \lvert\hat{y}_l\rvert F_\beta(y_l, \hat{y}_l)$

$$RecallMicroAvg = \frac{(TP_1+TP_2+\ldots+TP_n)}{(TP_1+TP_2+\ldots+TP_n+FN_1+FN_2+\ldots+FN_n)}$$

$$PrecisionMicroAvg = \frac{(TP_1+TP_2+\ldots+TP_n)}{(TP_1+TP_2+\ldots+TP_n+FP_1+FP_2+\ldots+FP_n)}$$

- y the set of *predicted* $(sample, label)$ pairs
- $\hat{y}$ the set of *true* $(sample, label)$ pairs
- L the set of labels
- S the set of samples
- y_s the subset of y with sample s, i.e. $y_s := \{(s', l) \in y | s' = s\}$
- y_l the subset of y with label l
- similarly, $\hat{y}_s$ and $\hat{y}_l$ are subsets of $\hat{y}$



Types of algoritms

Logistic Regression (LR)

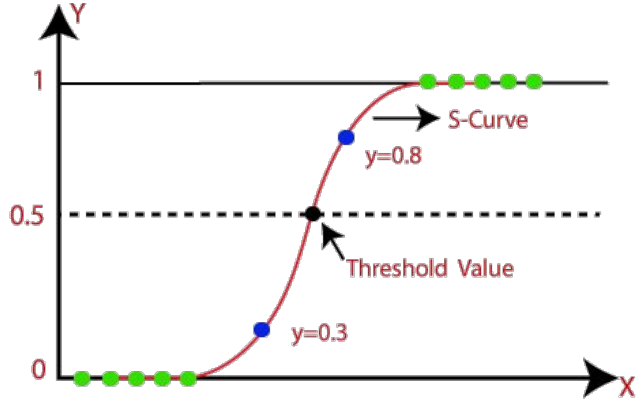
LR is a linear model for classification rather than regression

Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier

[logistic function](#) is used to calculate the probability

$$\mathbb{P}\left(Y = 1 \,/ X_1 = x_1, \ldots, X_p = x_p\right) = \frac{\exp\left(\beta_1 x_1 + \ldots + \beta_p x_p\right)}{1 + \exp\left(\beta_1 x_1 + \ldots + \beta_p x_p\right)}$$

Example: binary classification



l2 regularization

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

l1 regularization

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

Elastic-net regularization

$$\min_{w,c} \frac{1-\rho}{2} w^T w + \rho \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

Logistic Regression (LR) - Results analysis

➢ **Odds**

$$\text{odds}(i) = \frac{p(\mathbf{x}_i)}{1 - p(\mathbf{x}_i)}$$

➢ **Odds ratio (OR)**

$$\text{OR}(i, i') = \frac{\text{odds}(i)}{\text{odds}(i')} = \frac{\frac{p(\mathbf{x}_i)}{1-p(\mathbf{x}_i)}}{\frac{p(\mathbf{x}_{i'})}{1-p(\mathbf{x}_{i'})}}$$

$$\text{OR}(i, i') \simeq \frac{p(\mathbf{x}_i)}{p(\mathbf{x}_{i'})}$$

Using **backward**, **forward** and **stepwise** algorithms based on criteria such as the **AIC** (Akaike criterion) or the **BIC** (Schwarz criterion)

Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem

Naive Bayes is a simple technique for constructing classifiers

independence assumptions between the features

Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables

Bayes' theorem states the following relationship, given class variable y and dependent feature vector
x1 through xn

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y)$$

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Other versions:

- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Complement Naive Bayes

K-Nearest Neighbors (KNN)

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label/average from these

The number of samples can be a user-defined constant (k-nearest neighbor learning)

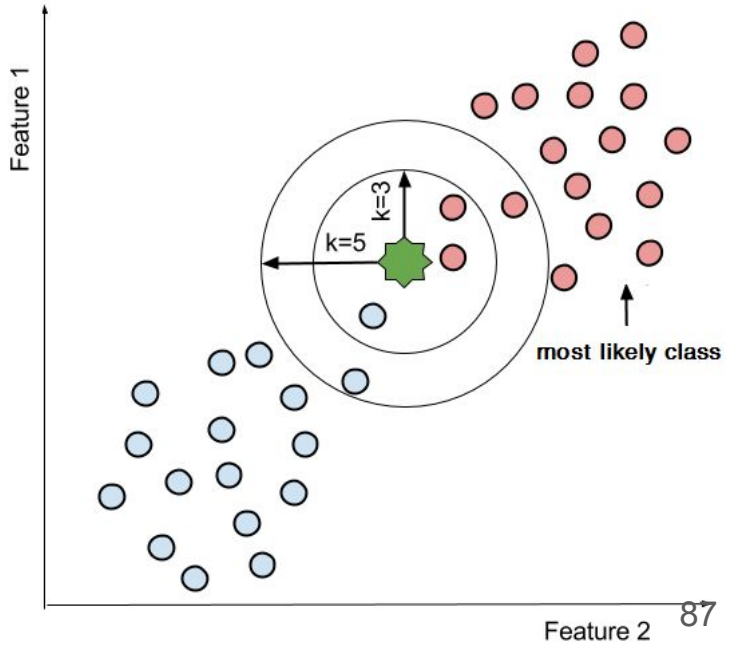
The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice

`weights = 'uniform'`, assigns equal weights to all points

`weights = 'distance'` assigns weights proportional to the inverse of the distance from the query point

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$



Perceptron

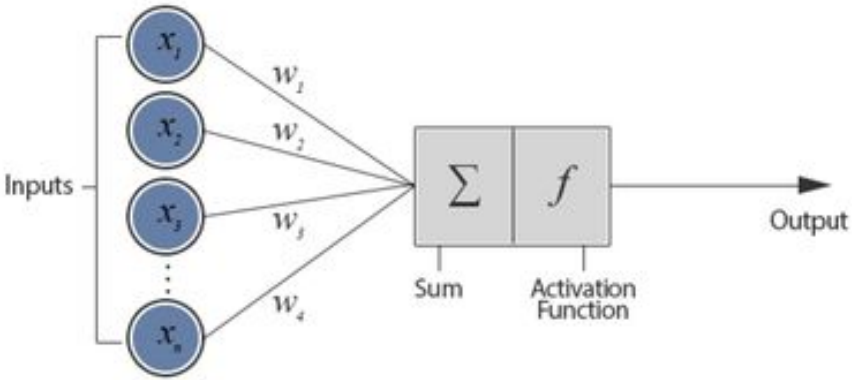
In 1957, the Perceptron was invented by Frank Rosenblatt

This is an algorithm for supervised learning of binary classification.

It is a type of linear classifier

The predicted result is compared with the known result. In case of difference, the error is backpropagated in order to adjust the weights

A neural network is a set of interconnected Perceptrons



$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

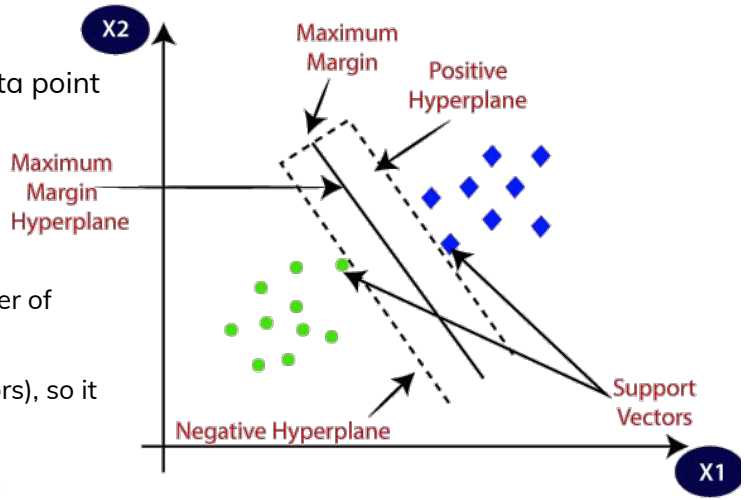
Support Vector Machines

SVM constructs a hyperplane or set of hyperplanes

SVM can be used for both classification or regression challenges

Support Vectors are simply the coordinates of individual observation.

Identify the right hyper-plane, maximizing the distances between nearest data point (either class)



Pros:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function.

Cons:

- Very slow with large data set
- It also doesn't perform very well, when the data set has more noise
- SVM doesn't directly provide probability estimates

Decision Trees - Loss function

1. Gini Impurity

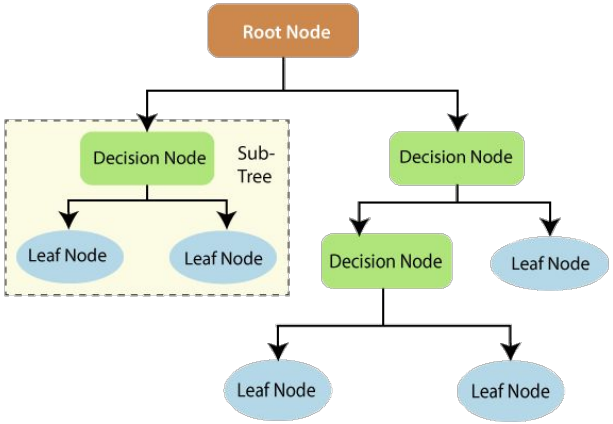
Gini Impurity is measure of variance across the different classes[1].

$$\mathrm{G}(\mathrm{node}) = \sum_{k=1}^{c} p_k(1 - p_k)$$

Probability of *not* picking a data point from class *k*

Probability of picking a data point from class *k*

$p_k = \frac{\text{number of observations with class k}}{\text{all observations in node}}$



2. Entropy (log loss)

Similarly to Gini Impurity, Entropy is a *measure of chaos within the node*. And chaos, in the context of decision trees, is having a node where all classes are equally present in the data

$$\mathrm{Entropy}(\mathrm{node}) = -\sum_{i=1}^{c} p_k \log(p_k)$$

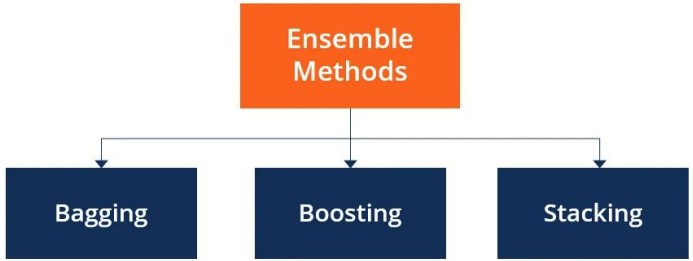
3. Hinge loss

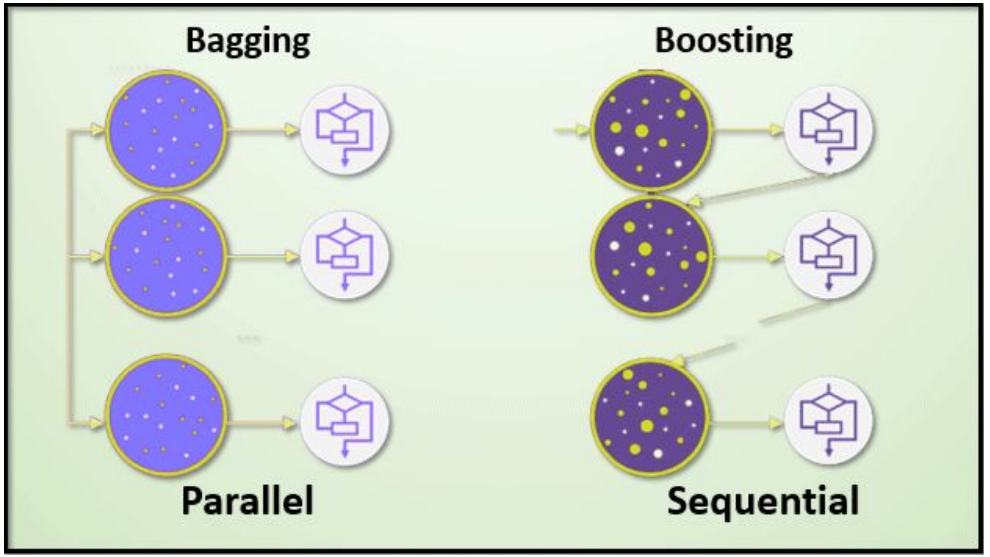
$$\ell(y) = \max(0, 1 - t \cdot y)$$

Ensemble methods

Examples:

- Random Forest
- AdaBoost
- ExtraTrees
- LightGBM
- XGBoost
- CatBoost





References

https://scikit-learn.org/stable/supervised_learning.html
https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net
https://towardsdatascience.com/ridge-lasso-and-elasticnet-regression-b1f9c00ea3a3
https://www.padhodatascience.com/2021/04/regularization-techniques.html
http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/
https://openclassrooms.com/fr/courses/4444646-entrainez-un-modele-predictif-lineaire/4507801-reduisez-l-amplitude-des-poids-affectes-a-vos-variables
https://openclassrooms.com/fr/courses/4444646-entrainez-un-modele-predictif-lineaire/4507806-reduisez-le-nombre-de-variables-utilisees-par-votre-modele
https://scikit-learn.org/stable/modules/sgd.html#sgd-mathematical-formulation
https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/
https://machinelearningmastery.com/types-of-classification-in-machine-learning/
https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff
https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
https://www.quora.com/In-classification-how-do-you-handle-an-unbalanced-training-set
https://en.wikipedia.org/wiki/Precision_and_recall
https://en.wikipedia.org/wiki/Receiver_operating_characteristic
https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205
https://datascientest.com/algorithmes-de-boosting-adaboost-gradient-boosting-xgboost
https://towardsdatascience.com/overview-ensemble-learning-made-simple-d4ac0d13cb96
https://vitalflux.com/micro-average-macro-average-scoring-metrics-multi-class-classification-python/
https://ml-compiled.readthedocs.io/en/latest/loss_functions.html

Questions ???