School of
Economics & Business
Universidad de Navarra

Introduction to Modelling & Machine Learning

Joan Arnau Romeu, PhD

Master in
Banking and Financial
Regulation

Universidad de Navarra
School of Economics & Business

Objective

Learn **the challenges involved in creating a model in the real world.** From obtaining the data to the best model to be used and the validation mechanisms depending on the type of problem. Good practices and description of the usual problems in a large modelling project.

In this course you **won't** find:

- Theorems and proofs
- Absolute truths
- In-depth and detailed model explanations
- Programming challenges

But you **will**:

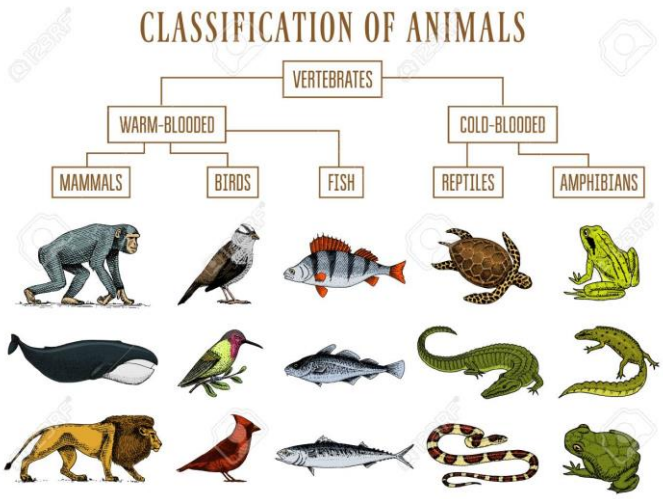
- Get a generalist approach to Machine Learning
- The intuition behind the mathematics of the key concepts
- Gain understanding on big corporates idiosyncrasies and how a Machine Learning project is developed
- Improve your ability to code and learn new models using package documentation and the internet (stack overflow)
- Develop a complex Machine Learning model from scratch

We have data... What do we do with it?

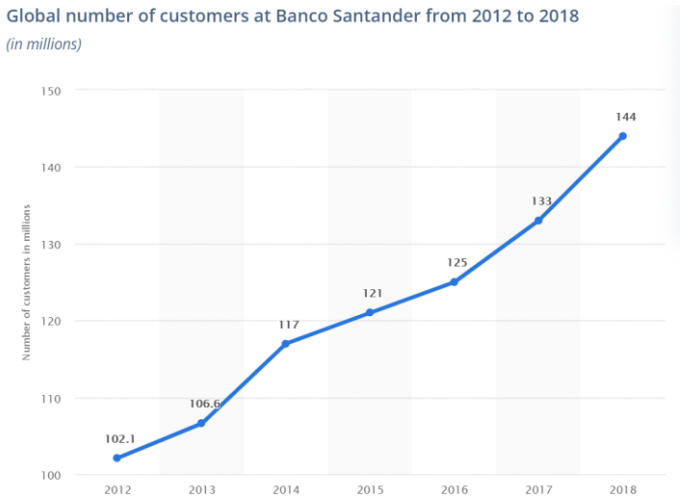
Data analysis allows you to extract useful and valuable information from the data you have. The basic tool used to understand data is the (often ignored) statistic.

Basic level ▶ *A bit more useful* ▶ *Level in which you start enjoying statistics*

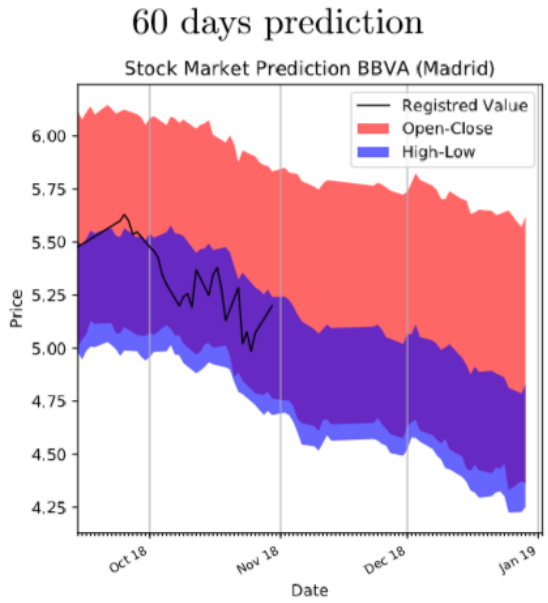
Manipulation and visualization



Extraction from past information



Prediction of future events



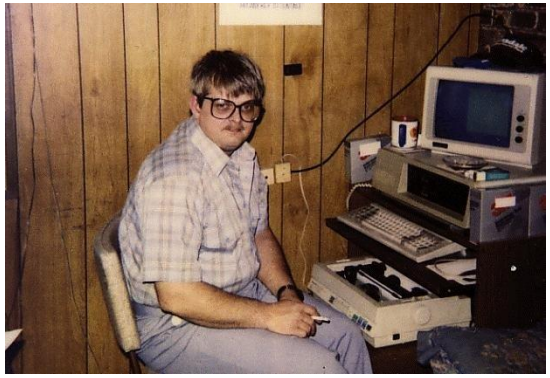
Why do Banks need predictive models?

Predictive models allow financial institutions to reduce de uncertainty on the likelihood of future events

Skilled management

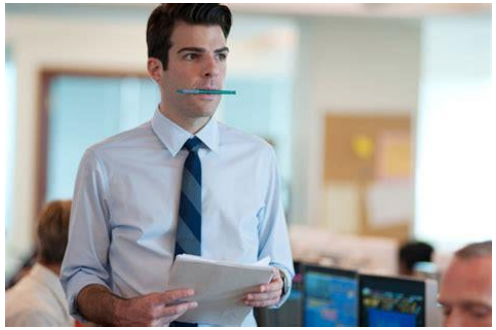
+

Unicorns



=

Successful & Efficient Business Strategy



+ extremely **deep understanding of business**

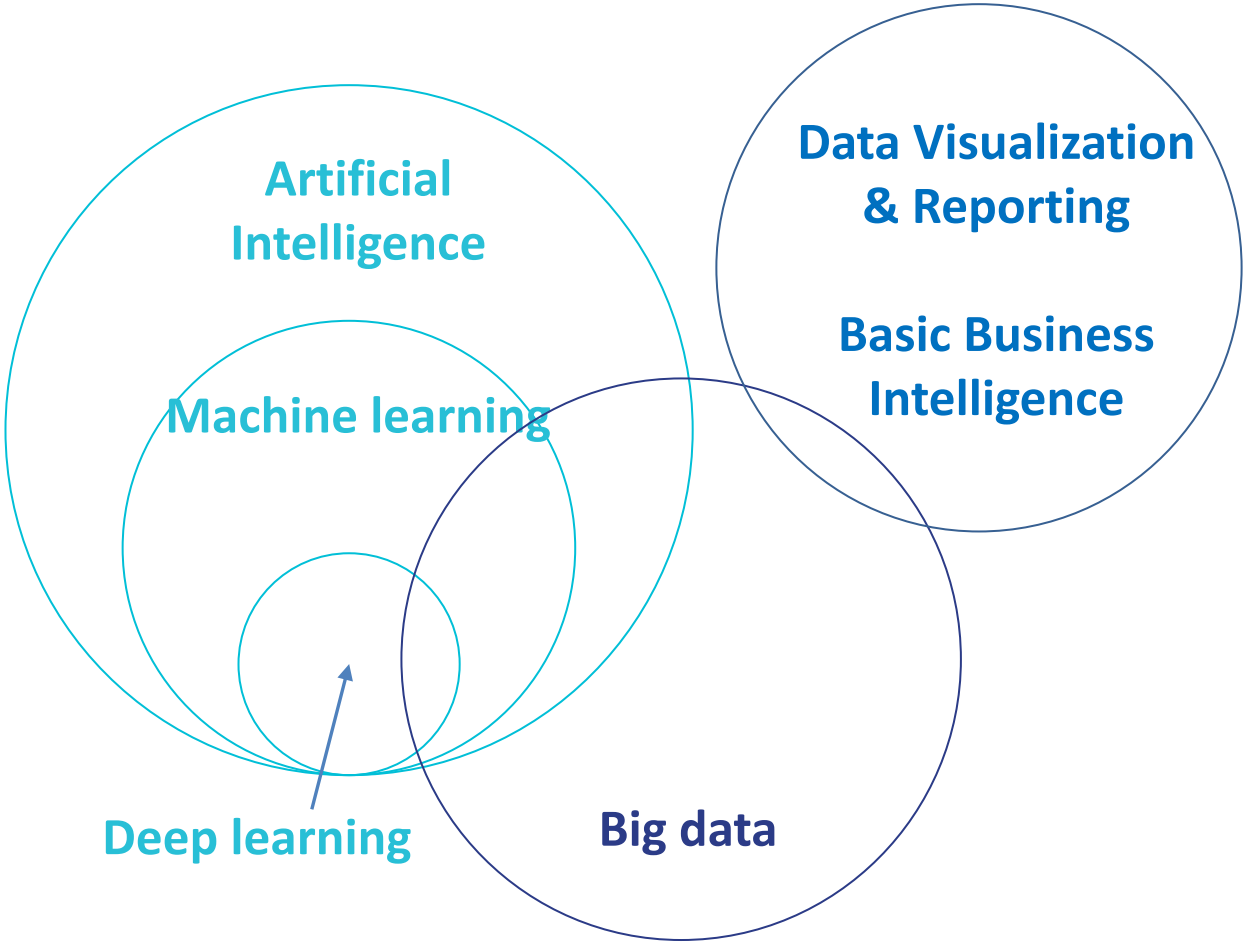
+ leadership qualities

+ communication skills

+ political skills

…

Data Science, Big Data, Machine Learning, Business Intelligence... What are we talking about?



Machine Learning algorithms build a mathematical model based on an observed data (training data) instead of using explicit instructions to make predictions or decisions without the need to explicitly program an algorithm to perform these tasks

Big Data refers to techniques for managing Data Sets that are too extensive or complex to be managed in a traditional way. Any Data Set that cannot be managed efficiently with a laptop can be considered Big Data, and requires a totally different data storage system (distributed and redundant)

Categorization of problems in Machine Learning

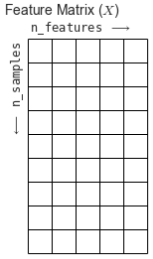
Unsupervised Learning

The result of the observation is not known a priori.

Objective: to find relationships in existing data (simplify data or group them into similar categories)

Main applications:

- Reduce the number of explanatory variables
- Segment: for example, types of customers



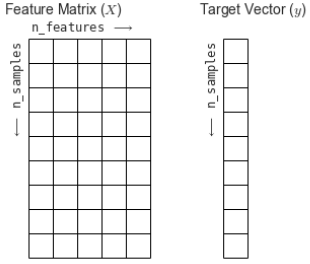
Supervised Learning

The result (target, for example default / no default) is known a priori

Objective: to train a model that is capable of making accurate predictions with new observations for which the result is unknown.

Main applications:

- Predict the default entry of healthy customers
- Predict the sale price of a real estate asset



Just if we have time

Reinforcement Learning

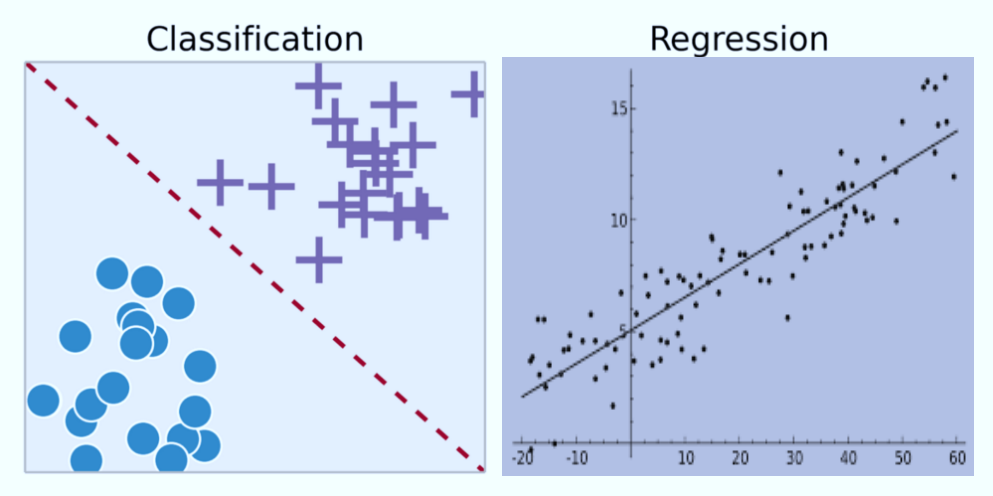
The result (target) is known only after the model makes a decision and the consequences are observed

Objective: to train a model that is capable of improving decision making to achieve a fixed objective

Main applications:

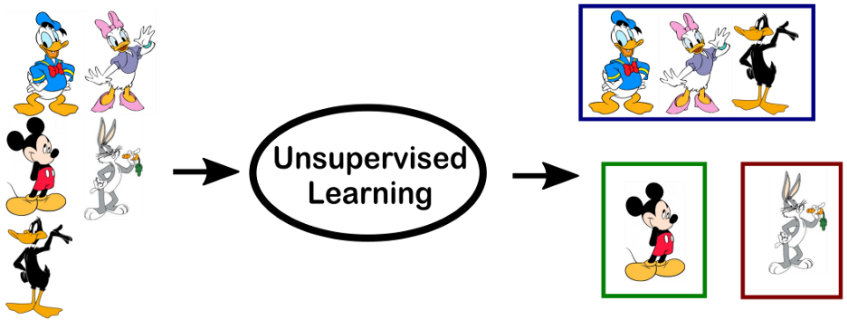
- Useful in problems where you want to "imitate" human intelligence: autonomous driving, games (chess, Go), etc.

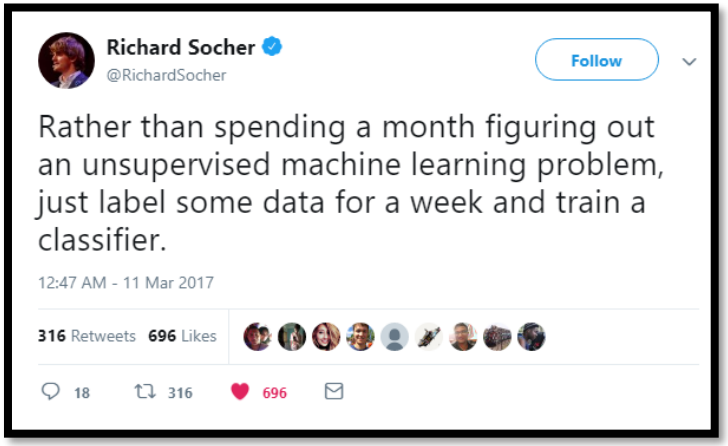
Supervised -vs- Unsupervised learning



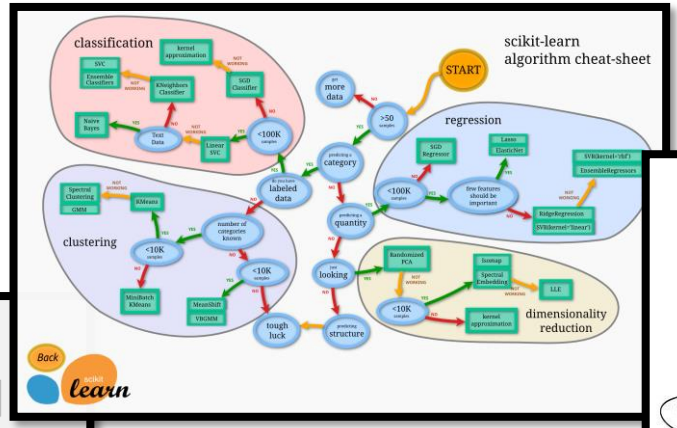
Supervised Learning: Learn a function that maps an input to an output based on known input-output pairs (labeled data). Classification & Regression.

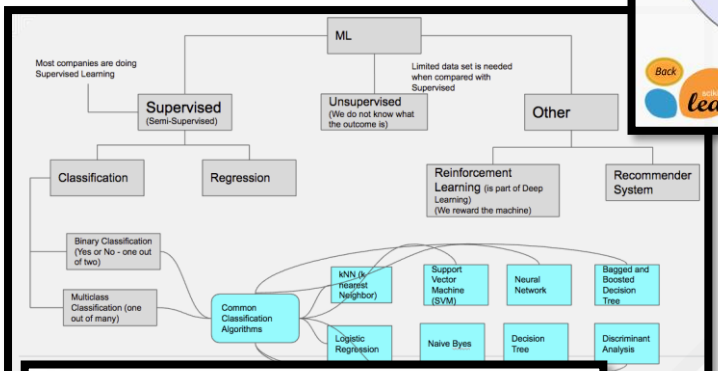
Unsupervised Learning: Find previously unknown patterns in data without pre-existing labels. Some notorious examples are clustering, dimensionality reduction and association (people that buy X tend to buy Y).

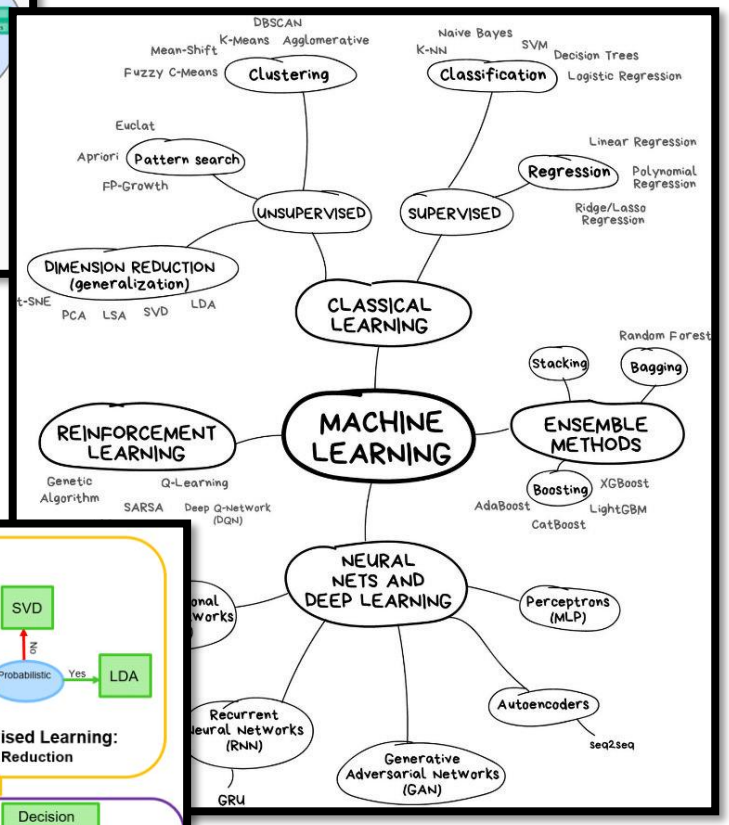


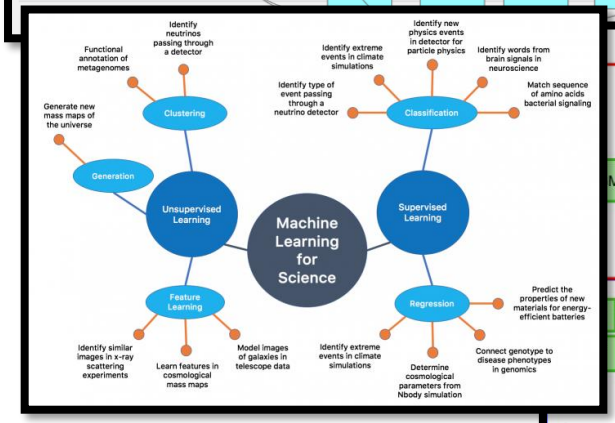


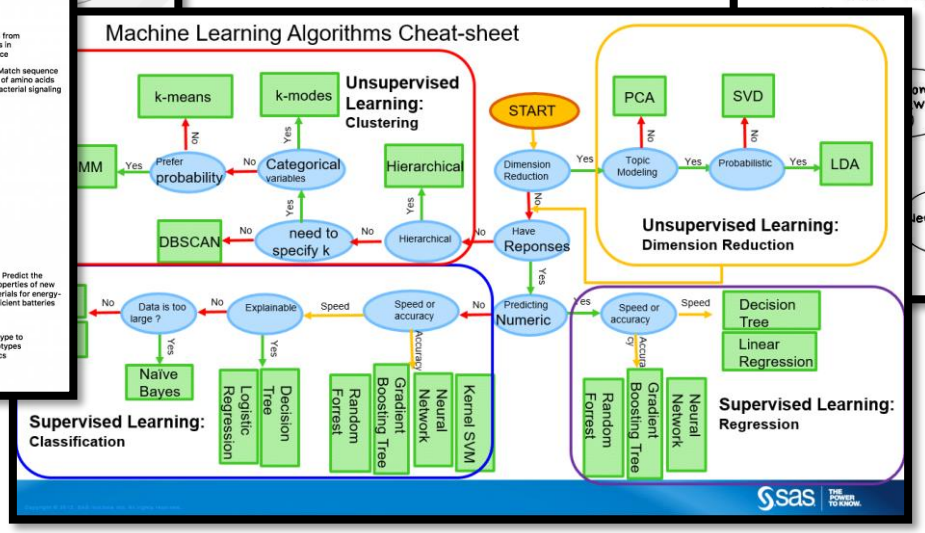
Machine Learning maps



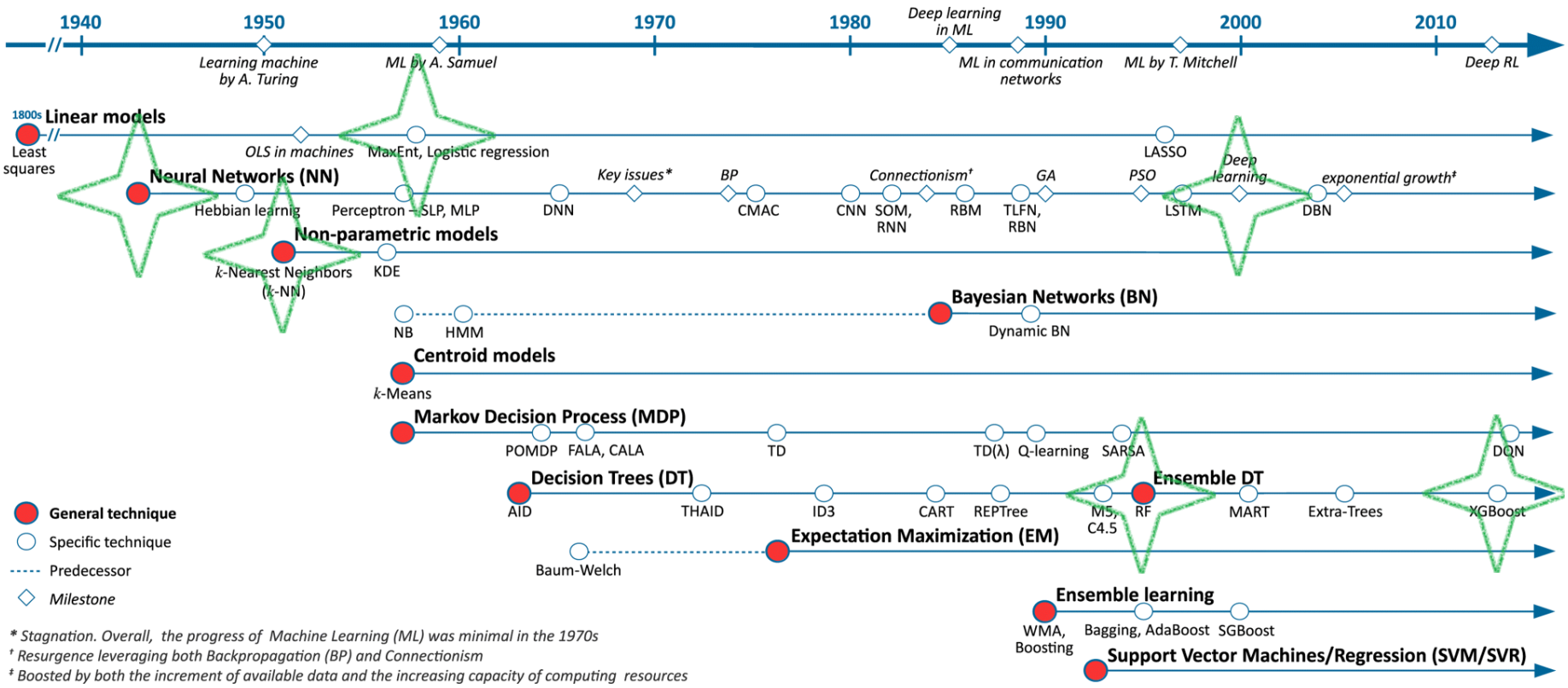






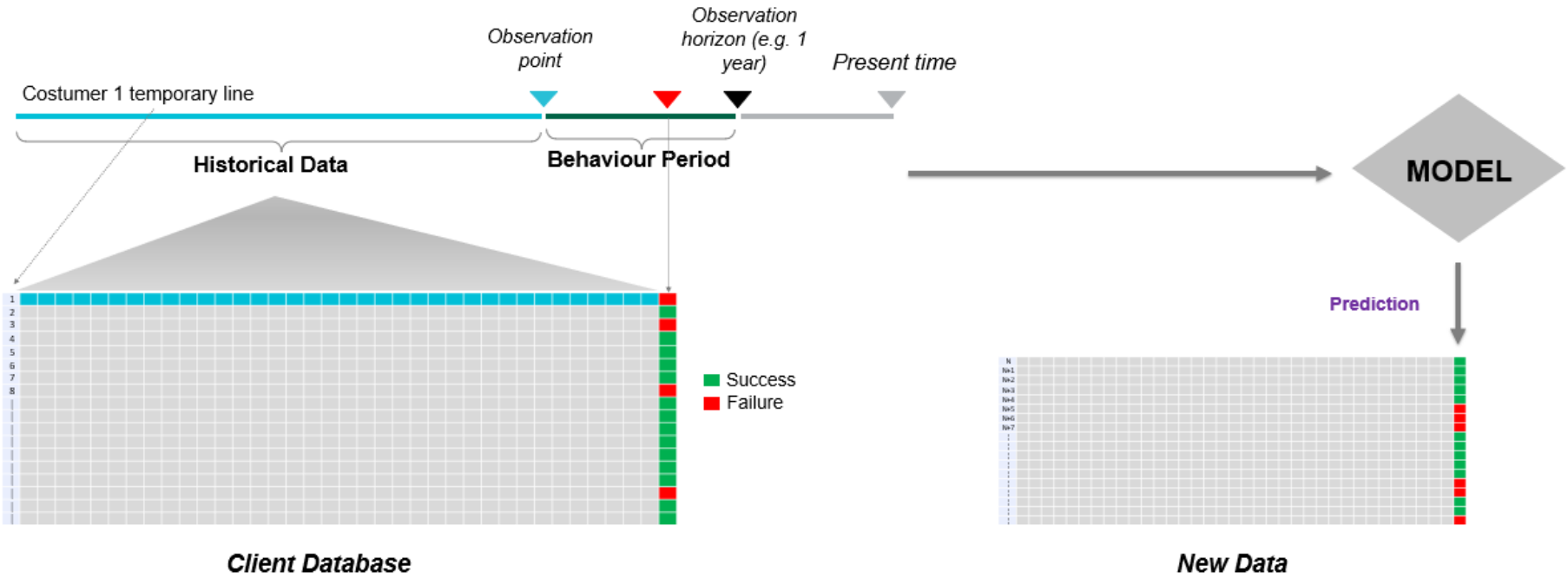


Historical perspective of Machine Learning



Example: Classical problems

Given an **OBJECTIVE**, we obtain a universe of susceptible "clients" and a Database with all the information relevant to the prediction of the **FLAG** that determines the success / failure in that **OBJECTIVE**, both for classification 0/1 (Default, Recovery, Activation, Loss of client, etc.) as for regression (Gross margin, Total benefit, Recovery amount, etc.)



WARNING: No future (post observation point or FLAG constructor) information!

Examples of behavioral problems

PROFITABILTY

OBJETIVE: Δ MARGIN (Regression)

CLIENT DATABASE	DATA	FLAG
All clients	Payment Behavior Products Sociodemographic Credit Bureau Liability accounts	Gross margin of interest and commissions 1 year after the observation point (€)

LINKAGE

OBJETIVE : Δ LINKAGE / ACTIVATION (Classification)

CLIENT DATABASE	DATA	FLAG
GOOD / INACTIVE GOOD	Products Sociodemographic Marketing Actions Liability Accounts Insurances	New credit in x weeks

OBJETIVE : (NON) CLIENT LOSS (Classification)

CLIENT DATABASE	DATA	FLAG
Liability Accounts	Liability Accounts Campaigns Sociodemographic History of Communications	Account cancellation in 1 year

REPAYMENT BEHAVIOR

OBJETIVE : (NON) DEFAULT (Classification)

CLIENT DATABASE	DATA	FLAG
GOOD (past due <= 4 weeks) New / Old Clients	Products Sociodemographic Credit Bureau Payment Behavior	Default (90 days in 1 year)

OBJETIVE : PREAPPROVED (Classification)

CLIENT DATABASE	DATA	FLAGS
Liability Clients without Credit	Sociodemographic Credit Bureau Liability accounts	Default or Past due

OBJETIVE : REPAYMENT (Regression / Classification)

CLIENT DATABASE	DATA	FLAGS
Credit clients with at least 1 ast due Payment	Products Sociodemographic Credit Bureau Payment Behavior	Total Recovery Partial Recovery Recovered Amount (€)

(Large) Modeling project

1 - Meetings with data owners and "acceptance of reality" (meaning that data that a priori was available, in fact, it is not)

1-4 Weeks

2 - Construction of Model Database

3 months

3 - Pure Modeling

3 days

python™

sas

R

4 - Analysis and System Implementation

- Back testing, Validation, Functionals, Documentation, Technology requirements…
- Robust system implementation vs "external" implementation

3 months to year(s)

Main modeling Approaches & (pre-programmed) Algorithms

Regression and Classification (tabular data)

Tabular data → rows (observations) and columns (features)

- **Linear/Logistic Regression** (Ordinary Least Squares / Maximum Likelihood Estimation)
- **Regularized Linear Models** (LASSO, Ridge, ElasticNet)
- **Random Forest**
- **Gradient Boosting Machines** (XGBoost, LightGBM, Catboost)
- **Standard Neural Network** (Multi-Layer Perceptron)

Unsupervised

Tabular data → rows (observations) and columns (features)

- **K-means** (Clustering)
- **Principal Components Analysis (PCA)** (Dimensionality Reduction)

Image Recognition (pixel data)

Each observation is a sequence of numbers between 0 and 255 for each color (RGB):

- **Standard Neural Network** (Multi-Layer Perceptron)
- **2D Convolutional NN**

Text Sentiment Analysis (words)

Each observation is a text in some human language

- **Standard Classification with a previous word sparse embedding** (Natural Language Processing: Stop Words, Stemming and Vectorizer)
- **1D Convolutional NN**
- **Recurrent NN** (LSTM: Long Short-Term Memory)

RULE #1

Nothing is what it seems

All counting values have statistical fluctuations

If you have N observations you actually have probably between N-$\sqrt{N}$ and N+ $\sqrt{N}$ observations.

RULE #2

Size matters

The more data, the les relevant is the statistical fluctuation.

RULE #3

But the quality of the raw material also matters

The model will be more reliable the more the observations are similar to the reality in which the model will be applied.

Mantra of modelling	***UNDERSTAND YOUR INPUTS!!***

To practice and learn more!

Dedicated webpages

- Towards Data Science: blog with many articles and almost daily periodicity on data science and modeling
- PyData: quarterly convention, the speakers' talks can be found on YouTube

Main uses of Kaggle:

- **Competitions**: make a model and compare results
- **Kernels and discussions**: users hang projects and discuss results, can serve as a "source of ideas"
- **Learning**: courses and competitions “playground”

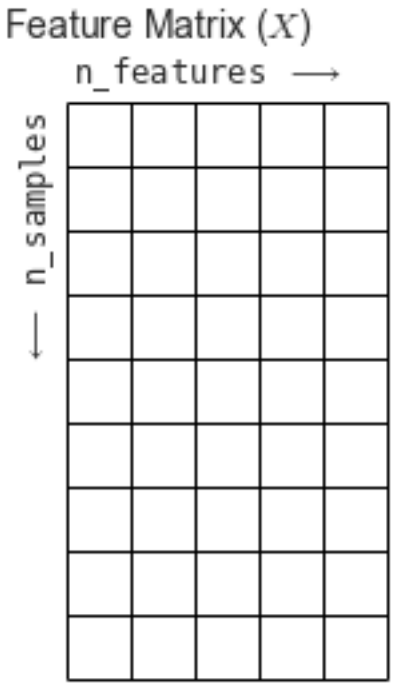
In addition, there are many technical help online resources to go deeper into modeling and data-science issues. Some examples:

Stack overflow and Cross validated: questions and answers forum on programming (Stack Overflow) and data science (Cross Validated)

A brief introduction to Unsupervised Learning

A better understanding of the data

In unsupervised learning problems we do not have the “real” (target) variable, so the objective is not to predict, but to better understand the data we have



+

Predefined Algorithm,
a priori it is a black box

=

A Better understanding of the populations of the dataset

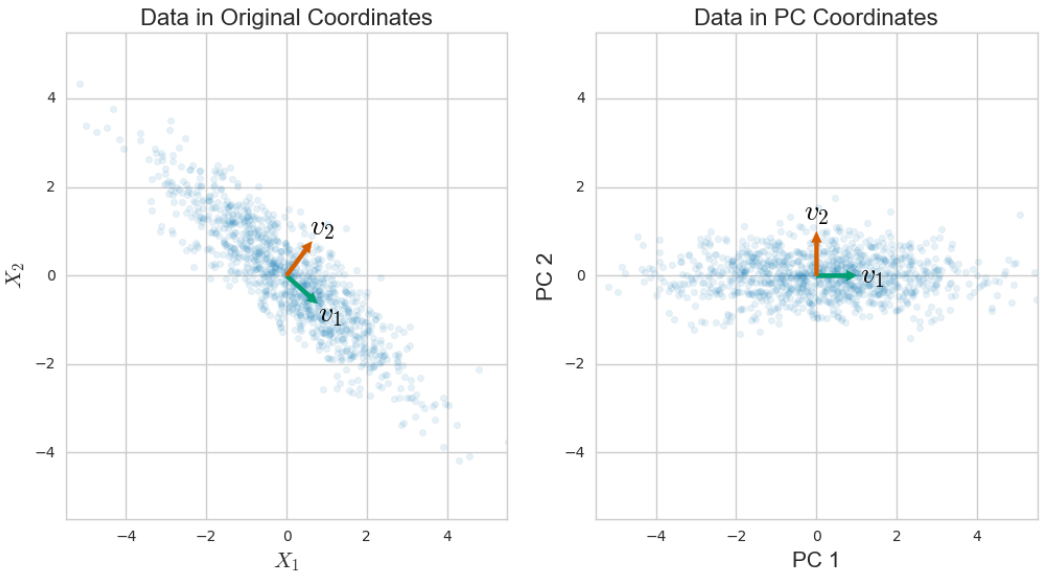
Dimensionality reduction

Objective: simplify the existing data, so that most of the variance of the data can be captured with the least number of features possible

Algorithms: PCA y t-SNE

PCA = PCA(n_components = 2)
prin_components = pca.fit_transform(data)

t-SNE = TSNE(n_components = 2)
tsne_results = tsne.fit_transform(data)



Main applications:

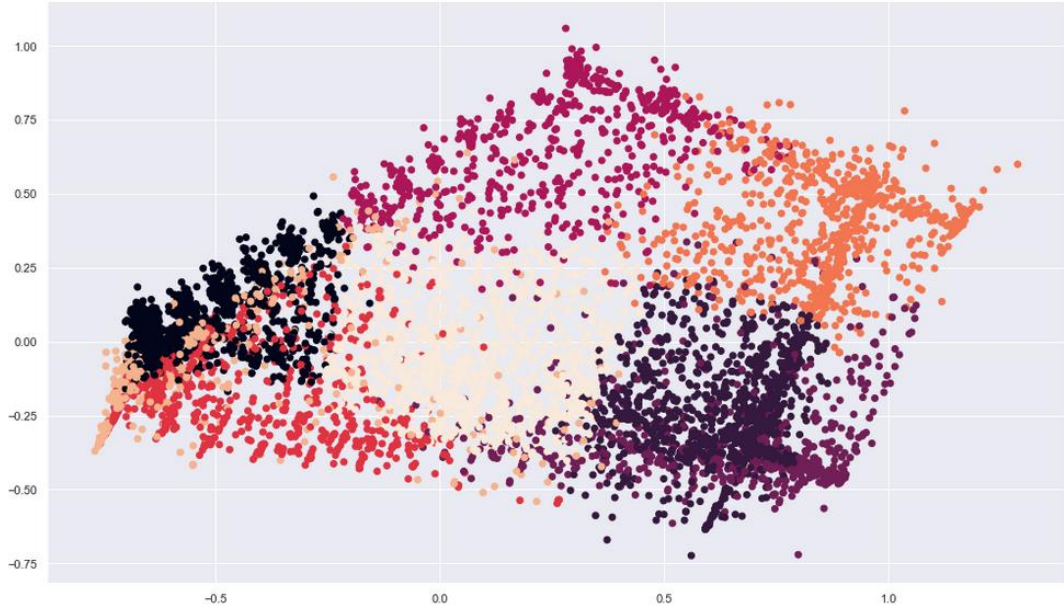
- Simplify models: less features and more representative => less computational cost
- Data visualization with many features (many dimensions)

Clustering

Objective: group observations into more or less homogeneous groups

Algorithms: K-means

```
from   sklearn.cluster   import KMeans
themodel = KMeans(n_clusters=5)
themodel.fit(data)
pred_y = themodel.predict(data)
```



Main applications:

- Identify subpopulations: for example, to apply different models per group
- Real application example: Segmenting Credit Card Customers with Machine) Learning

https://towardsdatascience.com/segmenting-credit-card-customers-with-machine-learning-ed4dbcea009c

Optimizing the number of clusters

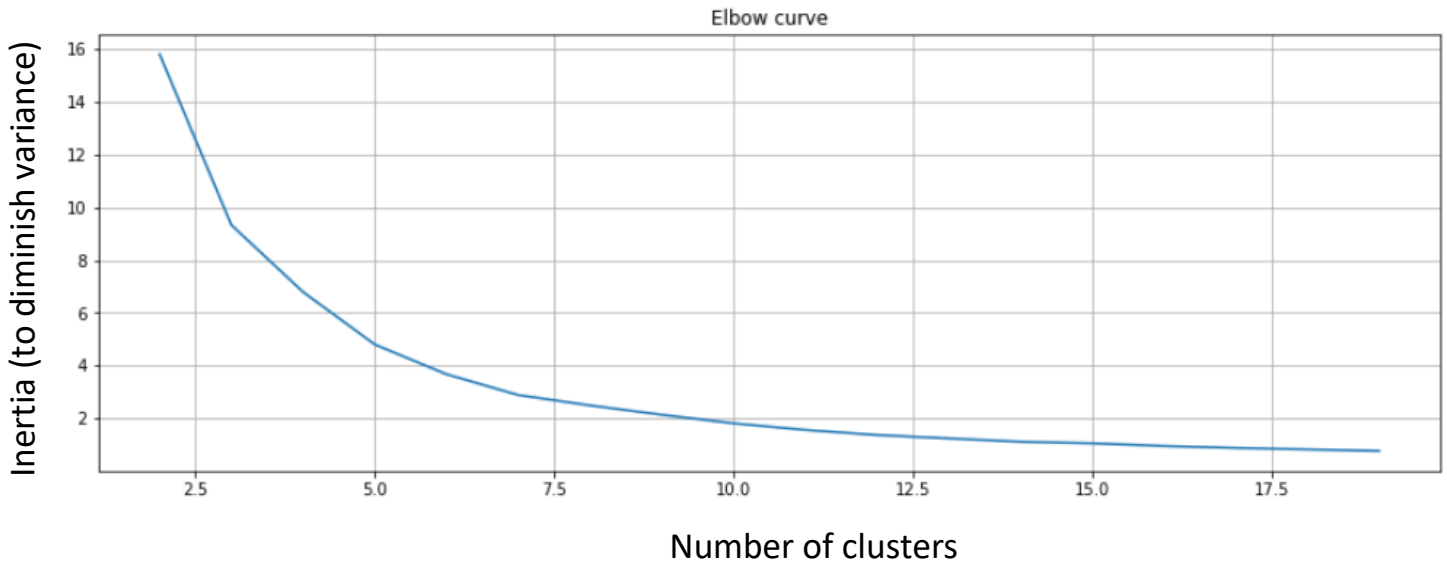
Elbow method (clustering)

From Wikipedia, the free encyclopedia

In cluster analysis, the **elbow method** is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. The same method can be used to choose the number of parameters in other data-driven models, such as the number of principal components to describe a data set.

The method can be traced to speculation by Robert L. Thorndike in 1953.[1]

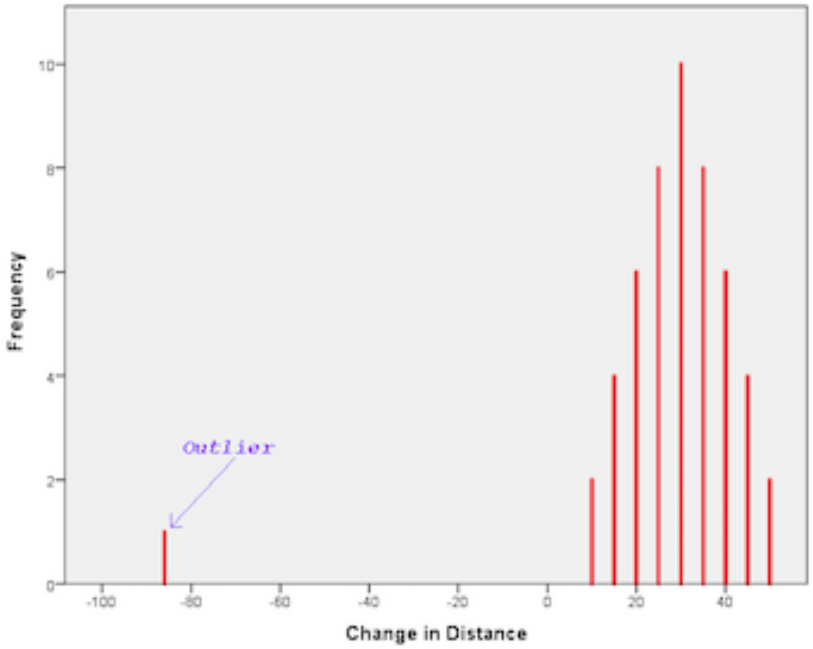
```
inertia = []
for k in range(2, 20):
    k_means = KMeans(n_clusters=k)
    k_means.fit(data)
    inertia.append(k_means.inertia_)

fig = plt.figure(figsize=(15, 5))
plt.plot(range(2, 20), inertia)
plt.grid(True)
plt.title('Elbow curve')
```



Considerations

Be careful about outliers

Sometimes you will have to delete rows containing outliers

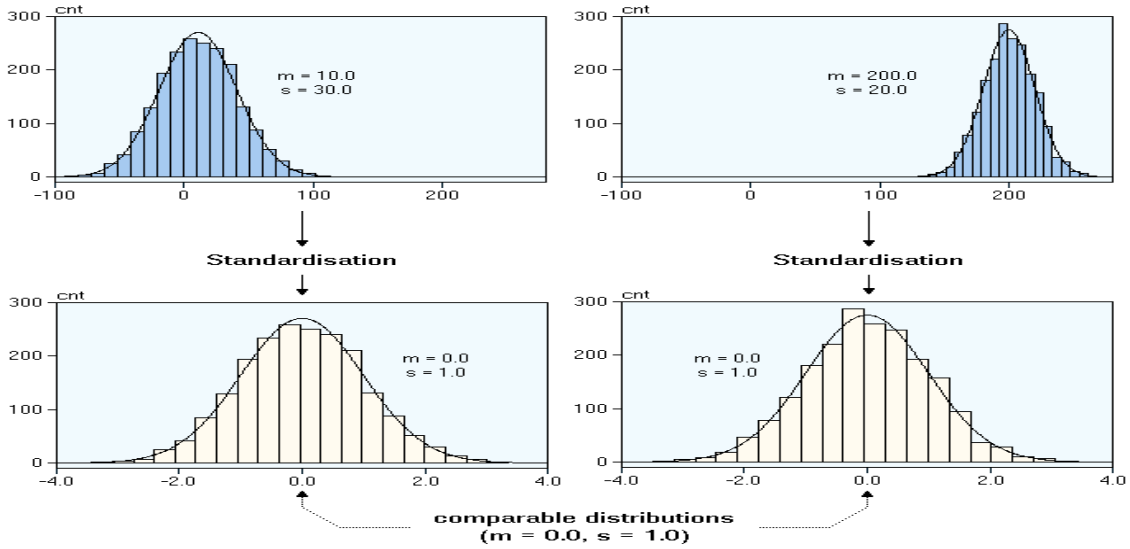


Variables must be in the same units

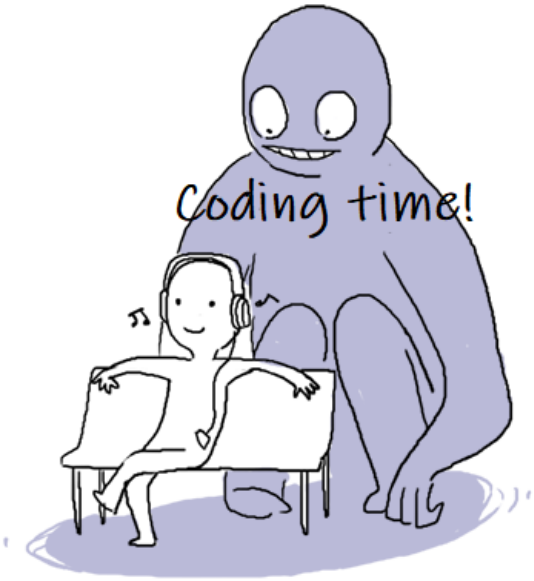
Kmeans needs that the variables are in the same units (e.g. in €). If the input variables are in different units (=> different order of magnitudes) it is necessary to standardize them first.

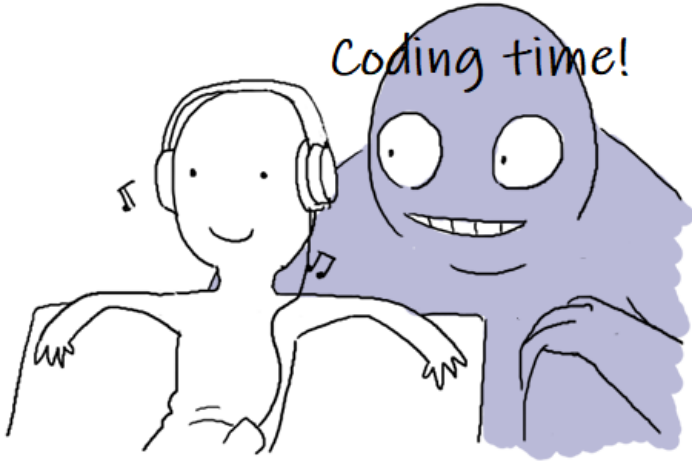
Example in python of how to standardize one column

```
df['colA_standard'] = (df['colA'] - df['colA'].mean()) / df['colA'].std()
```



Exercise:





Unsupervised Learning algorithm example

Building a customized portfolio with clustering techniques

You have available a DB with a wide list S&P 500 stocks.

GOAL: Group the stocks by similarities in their historical return and similarities to create a customized risk/returns porfolio offers for your clients

Steps

1) Read the data "scrapped_database.csv" and set the 'Date' column as index

2) Compute the stock historical returns
- Compute the intraday percentage change (you can use the dataframe method .pct_change())
- Find the mean of the daily percentage change (you can use the dataframe method .mean())

You should find a dataframe as the following now

	Mean
Ticket1	-
Ticket2	-
...	...

- Use the following HYPOTHESIS: The intraday mean percentage day is constant through the year.

Therefore, to get the yearly returns, multiply the "Mean" column by the number of working days in year (e.g. 252)

3) Compute the Volatility using the standard deviation of the percentage change. You can use .pct_change().std() * sqrt(252)

4) Merge returns and Volatility in a single dataframe (df) with the 'ticket' as index, and create the input data for the k-means algorithm as

```
data = np.array([np.array(df['Returns']),np.array(df['Volatility'])]).T
```

5) Proceed to use the K-Means clustering algorithm to divide the stocks into distinct groups based upon their returns and volatilities. Try to optimize the nur

6) Visualize the data. You can use the followong lines:

```
plt.scatter(data[:,0], data[:,1], c=themodel.labels_, cmap=plt.cm.autumn)
plt.scatter(themodel.cluster_centers_[:, 0], themodel.cluster_centers_[:, 1], s=300, c='blue')
plt.xlabel('Returns')
plt.ylabel('Volatility')
plt.show()
```

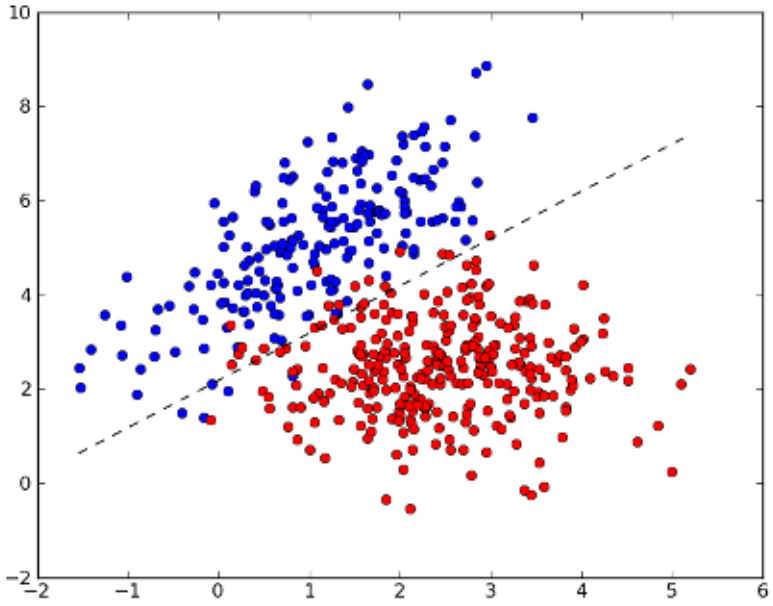
7) Is everything ok? If you feel that some datapoint is not correct, eliminate it an re-do point 3) and 4)

Supervised Learning

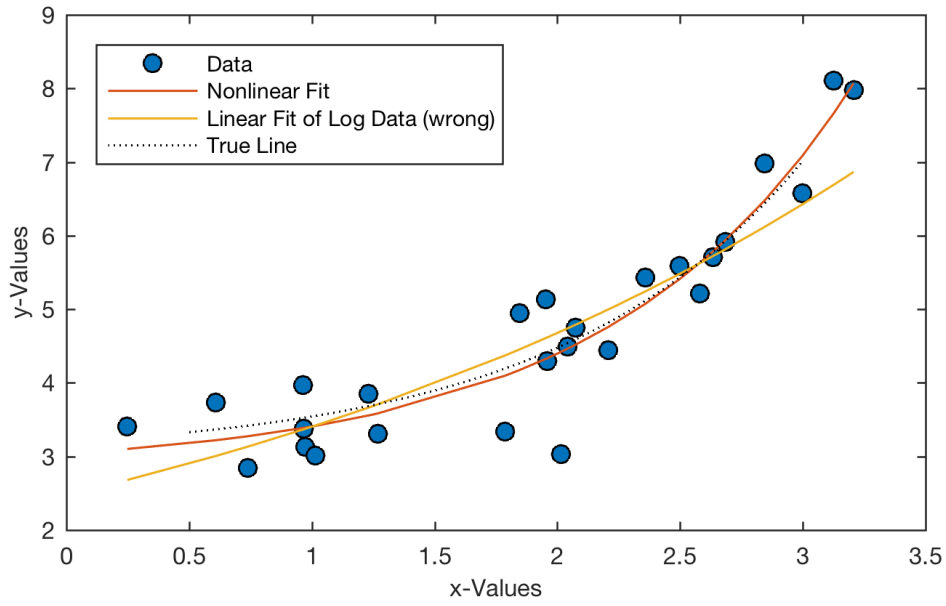
Building predictive models

In supervised learning problems we do have the “real” (target) variable for training the models.

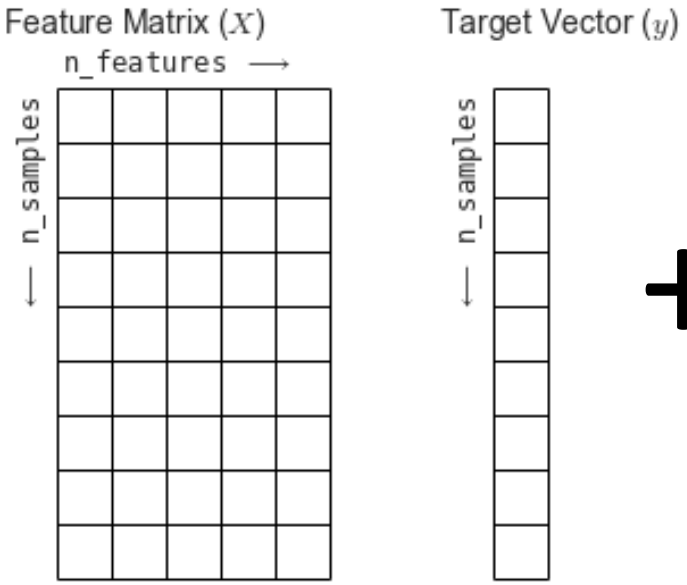
Classification



Regresion



In supervised learning problems we do have the “real” (target) variable for historical data.



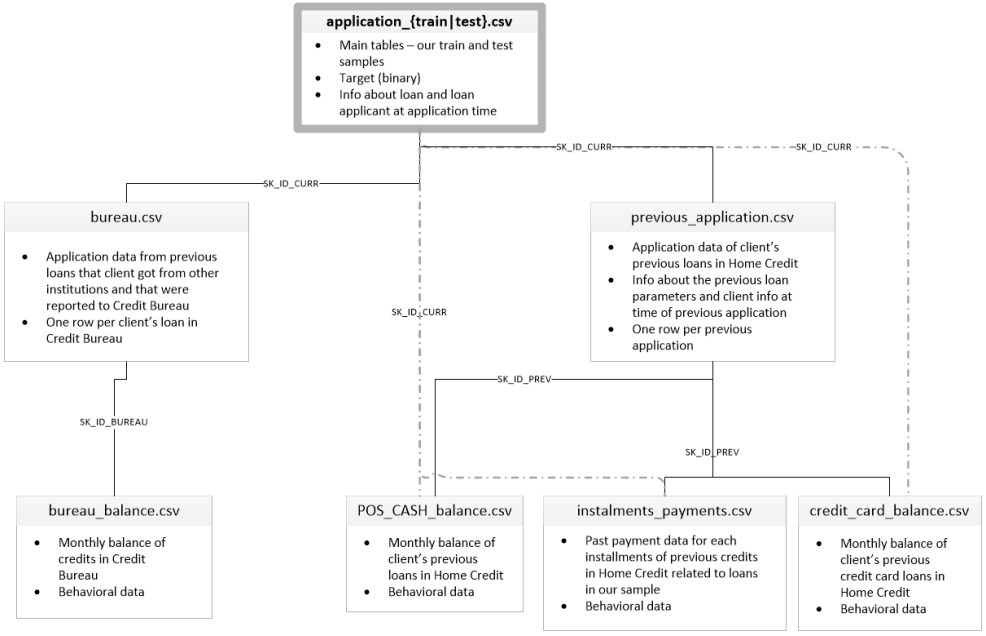
+

= A predictive model

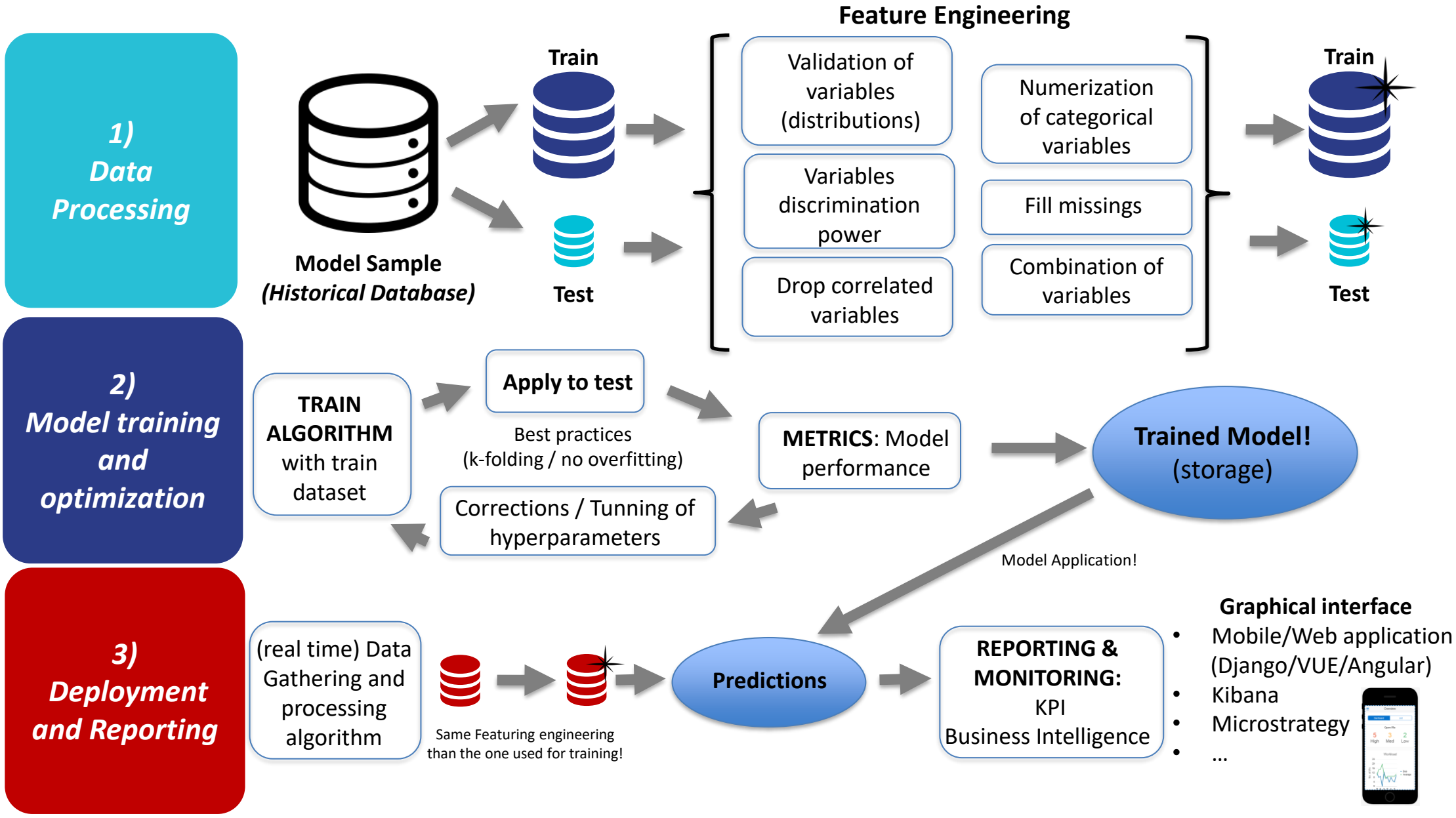
Predefined Algorithm,
a priori it is a black box

But indeed it is not that easy…

20's century technology



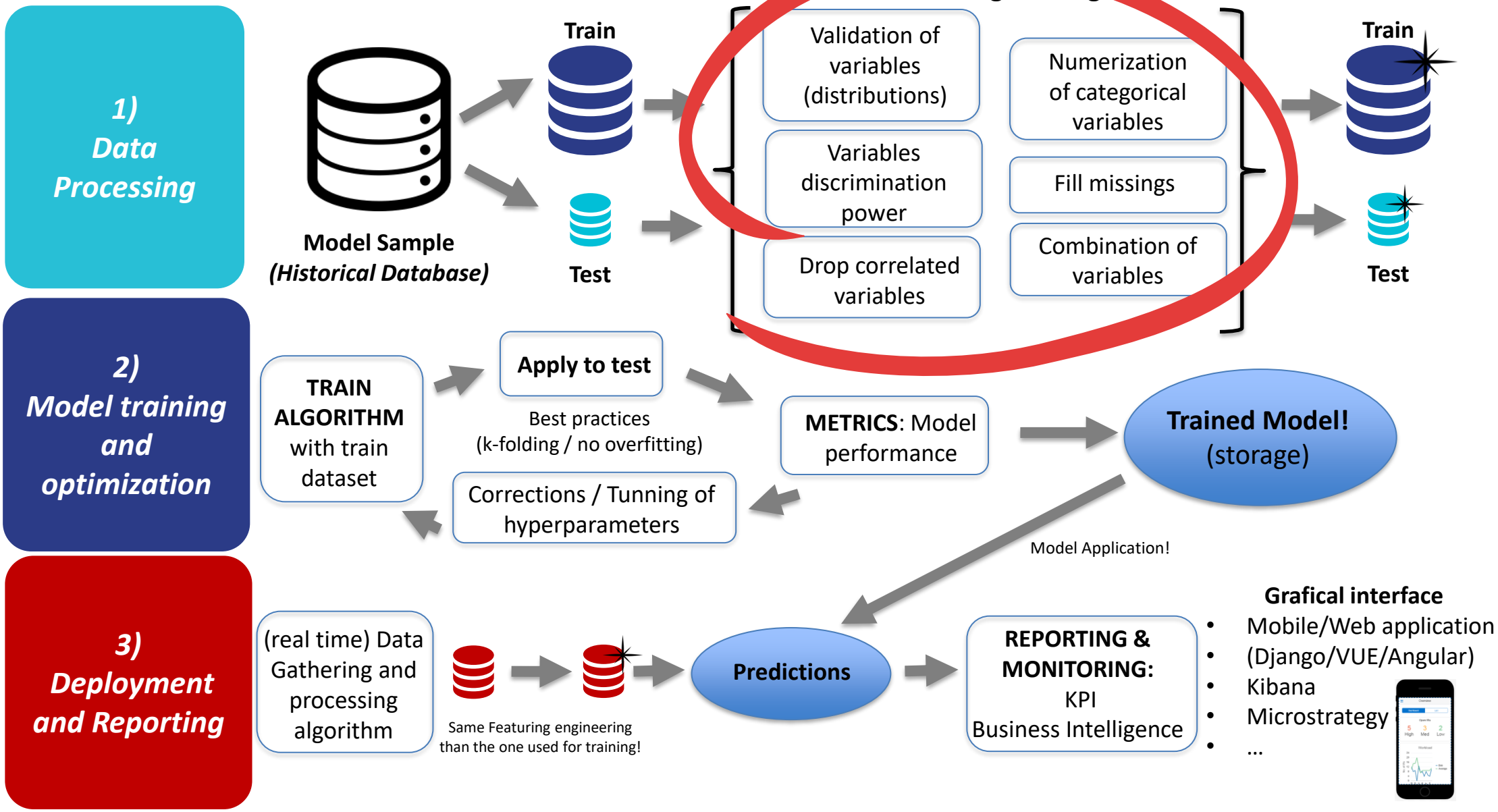
General flow of a (large) modeling project of supervised learning



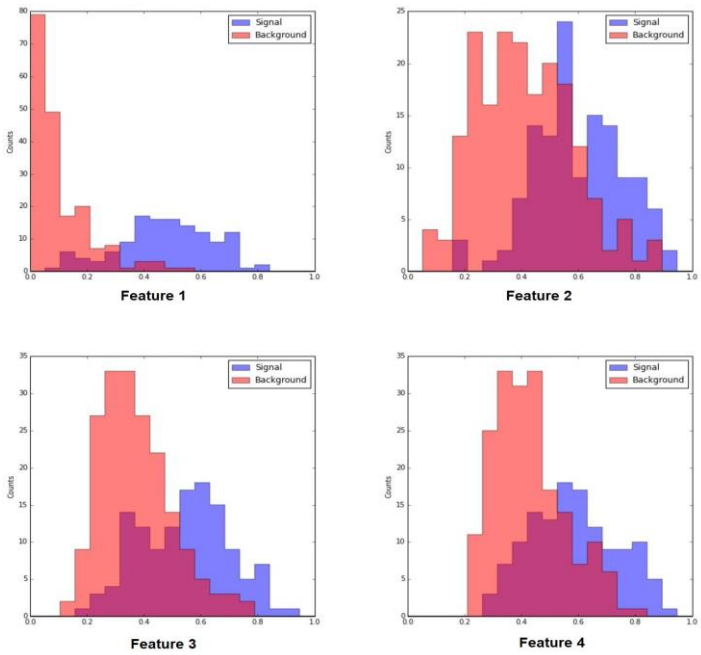
Preparing Data for modelling:

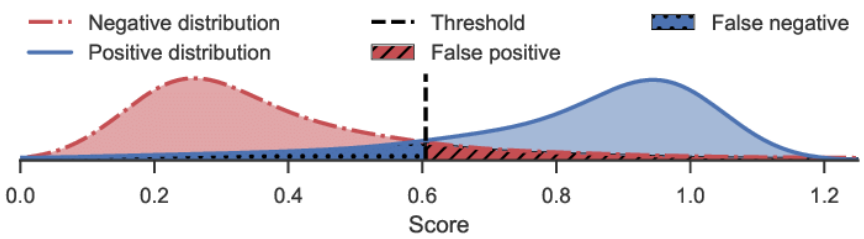
Variables Validation and Feature Engineering

General flow of a (large) modeling project of supervised learning

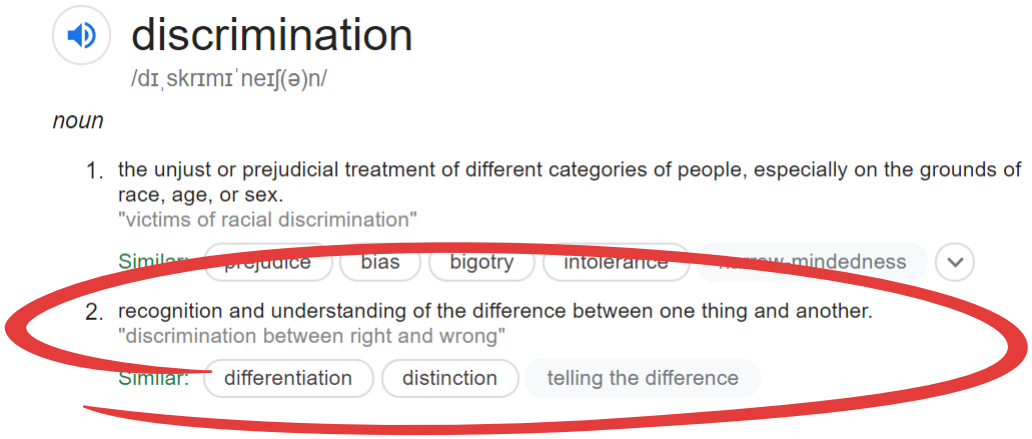


Variables distributions: A monographic on histograms





* Look at histograms for features of regression problems is also useful to check for inconsistencies present in the data



discrimination

/dɪˌskrɪmɪˈneɪʃ(ə)n/

noun

1. the unjust or prejudicial treatment of different categories of people, especially on the grounds of race, age, or sex.
   "victims of racial discrimination"
   Similar: prejudice | bias | bigotry | intolerance | narrow-mindedness
2. recognition and understanding of the difference between one thing and another.
   "discrimination between right and wrong"
   Similar: differentiation | distinction | telling the difference

- An histogram is the “statistical distribution” by excellence.
- The frequentist histogram provides the *probability density* of any variable.

For our purposes on classification problems:

- The separation is intimately related to the discrimination power of a variable.
- To build a model we want the variables with greater discrimination power.
- The distribution of a given variable MUST have (financial) sense

Variables discrimination power: Univariant analysis for feature selection

Check which variables useful for the model by ranking them in terms of their discrimination power.

Classification

Sommer's D or GINI (very well explained by our friend Wikipedia)

$$D_{YX} = \frac{N_C - N_D}{N_C + N_D + N_T},$$

Frequencies of Y, X pairs

Y \ X	0.25	0.5	0.75
0	3	5	2
1	1	7	6

The number of concordant pairs equals

$$N_C = 3 \times 7 + 3 \times 6 + 5 \times 6 = 69.$$

The number of discordant pairs equals

$$N_D = 1 \times 5 + 1 \times 2 + 7 \times 2 = 21.$$

The number of pairs tied on Y but not on X equals

$$N_T = 3(5+2) + 5 \times 2 + 1(7+6) + 7 \times 6 = 86.$$

Thus, Somers' *D* equals

$$D_{YX} = \frac{69 - 21}{69 + 21 + 86} \approx 0.273.$$

Later we will see the AUC...

$$\text{AUC} = \frac{D_{YX} + 1}{2}$$

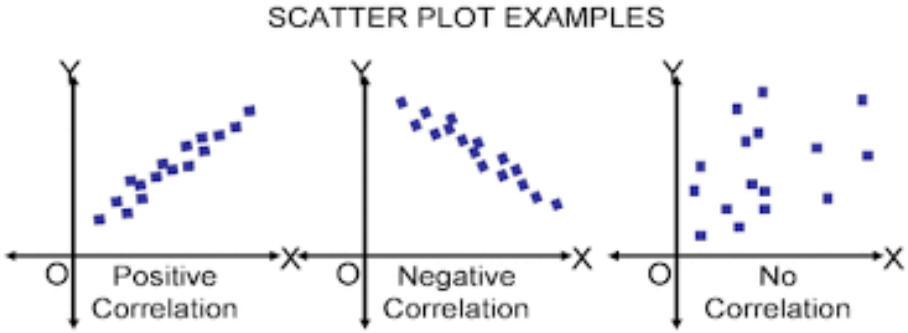
WARNING

Always be suspicious about features with too good GINI or too good correlations!

They may carry FUTURE information!!

Regresion

Correlations with the target variable



$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sqrt{(\sum x_i^2 - n\bar{x}^2)}\sqrt{(\sum y_i^2 - n\bar{y}^2)}}$$

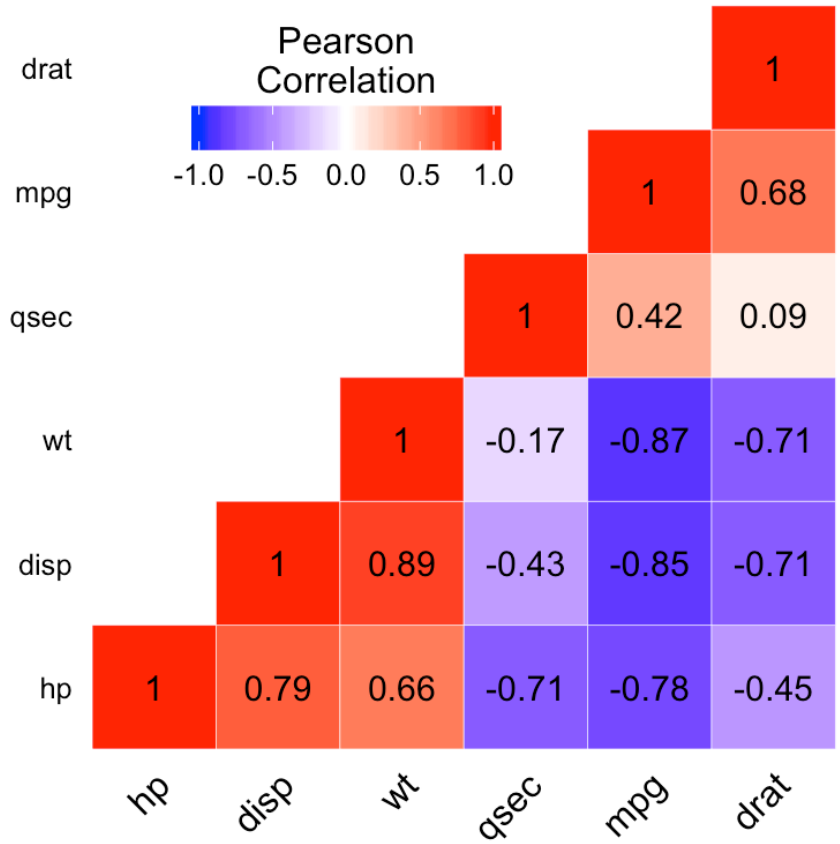
where:

- n is sample size
- x_i, y_i are the individual sample points indexed with *i*
- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$

Variables correlations

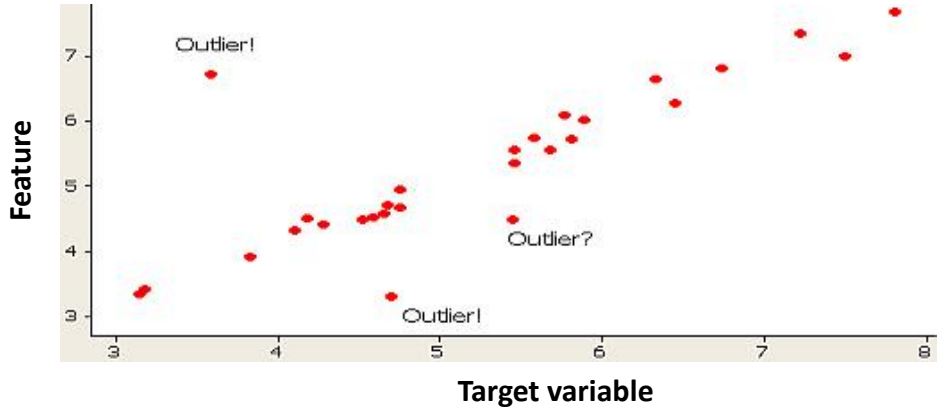
It is advisable to drop the correlated features to not over-complicate the model. The highly correlated features carry the same information.

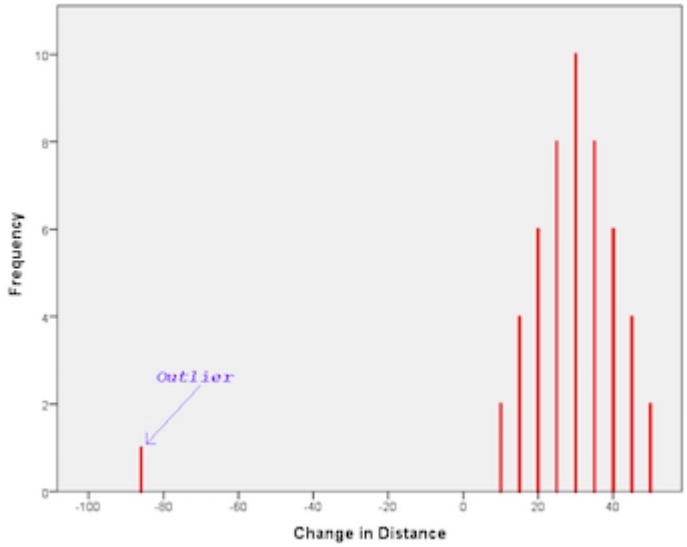
The confusion matrix is made up with the Pearson's Correlation coefficients between all variables combinations.



Outliers treatment

It is also advisable to drop the outliers to avoid biases in the model






```python
import numpy

def is_outlier(points, thresh=3.5):
    points=numpy.array(points)
    if len(points.shape) == 1:
        points = points[:,None]
    median = numpy.median(points, axis=0)
    diff = numpy.sum((points - median)**2, axis=-1)
    diff = numpy.sqrt(diff)
    med_abs_deviation = numpy.median(diff)
    modified_z_score = 0.6745 * diff / med_abs_deviation
    return modified_z_score > thresh

```

For well distributed variables:

$$outlier >< mean \pm X \sqrt{(variance)}$$

For badly distributed variables:

$$outlier >< median \pm X\ dispersion$$

Numerization: Treatment of categorical variables

There are 2 main reasons to make an specific treatment of the categorical variables:

- Not all model algorithms accept string variables as input.
- Minimize losing (even adding) information in the transformation process.

The best way of numerizing the categorical variable is by **correlating them to the TARGET variable.**

The **Feature Engineering procedure is a determining factor** for the quality of the model!

Be careful with introducing unnecessary biases!!

All categories tranformed to number via correlations need to have a significant population. If low population, merge some of the categories.

All numerized values must have economical/financial sense!!

For classification and regression problems:

Substitute each category by the mean (or mode) of the TARGET variable for such category.

For classification problems:

- Substitute each category (aI by its Weight of evidence (WoE):
- Use the one-hot-encoding technique (I am not a big fan of this one…)

$$\mathrm{WoE_a} = \ln\left(\frac{\frac{N_a^{good}}{N_{total}^{good}}}{\frac{N_a^{bad}}{N_{total}^{bad}}}\right) = \ln\left(\frac{\%good_a}{\%bad_a}\right)$$

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

WoE is a classic tool in Credit Score that consists in transforming all the features (numeric and categorical) to the natural logarithm of the odds ratio which is the dependent variable in logistic regression. Only valid for 0/1 Classification Problems.

Range	Bins	Non events	Events	% of Non-Events	% of Events	WOE
0-50	1	197	20	5.4%	5.9%	-0.0952
51-100	2	450	34	12.3%	10.1%	0.2002
101-150	3	492	39	13.4%	11.5%	0.1522
151-200	4	597	51	16.3%	15.1%	0.0774
201-250	5	609	54	16.6%	16.0%	0.0401
251-300	6	582	55	15.9%	16.3%	-0.0236
301-350	7	386	41	10.5%	12.1%	-0.1405
351-400	8	165	23	4.5%	6.8%	-0.4123
>401	9	184	21	5.0%	6.2%	-0.2123
	Total	3662	338			

Key Facts:

- WoE is also a way to deal with "U-shaped" features and other non-linear correlated exotic variables
- For Regression, the analogous is to take mean values of the TARGET
- Risk of overfitting

Steps for Calculating WoE:

- For a continuous variable, split data into 10 parts (or lesser depending on the distribution).
- Calculate the number of events and non-events in each group (bin)
- Calculate the % of events and % of non-events in each group.
- Calculate WOE by taking natural log of division of % of non-events and % of events
- Combine bins with similar WoE. Reduce number of bins or combine bins if there is no "economic sense"

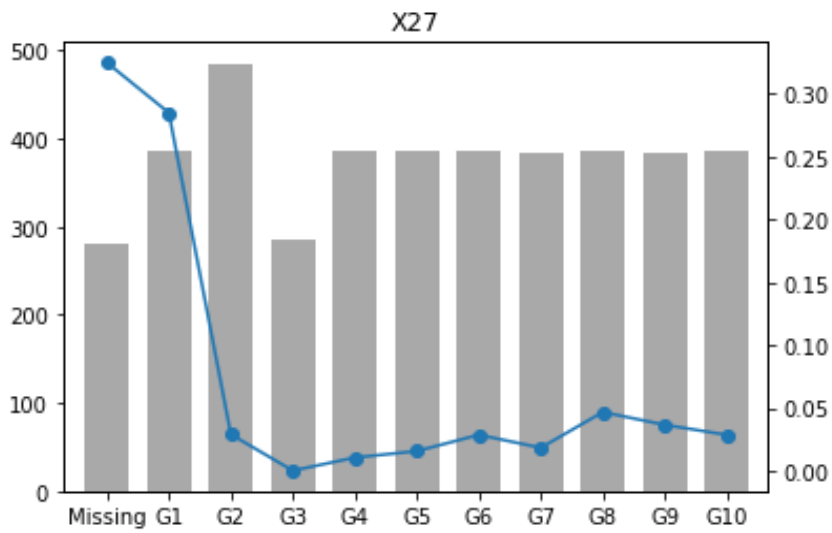
Note : For a categorical variable, you do not need to split the data (Ignore fist step and follow the remaining steps)

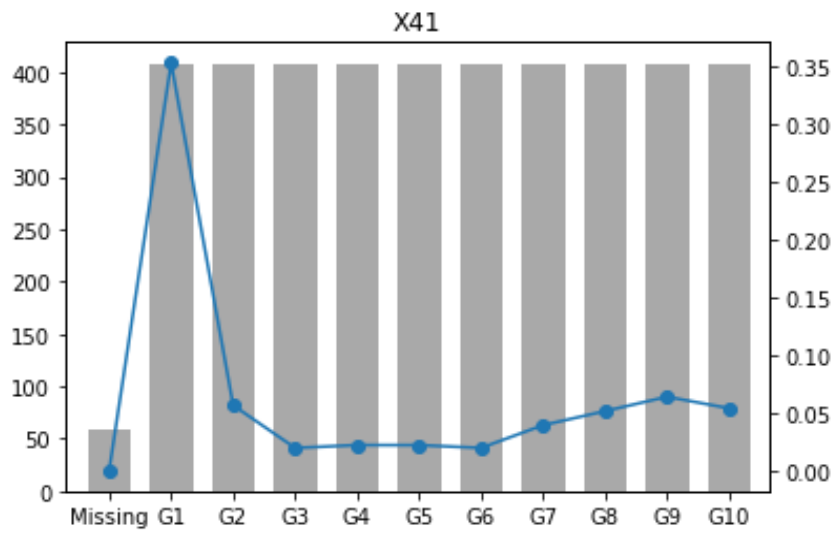
Missings imputation

Best option: Know the origin of the missing and impute 0 or the attribute average or any value with sense. Also for categorical variables, impute new category "NA" or the most frequent value for instance.

Ignore row (only for those observations that have ***MANY*** missing values)

Plot with respect to target and **assign a quantile** (or max or min → works well when using tree-based learners):



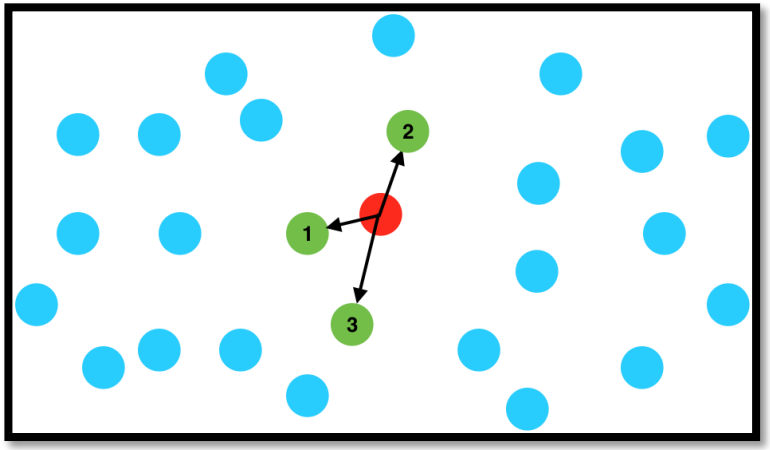


Default rate for each category

Missings imputation

- **K-Nearest Neighbors**

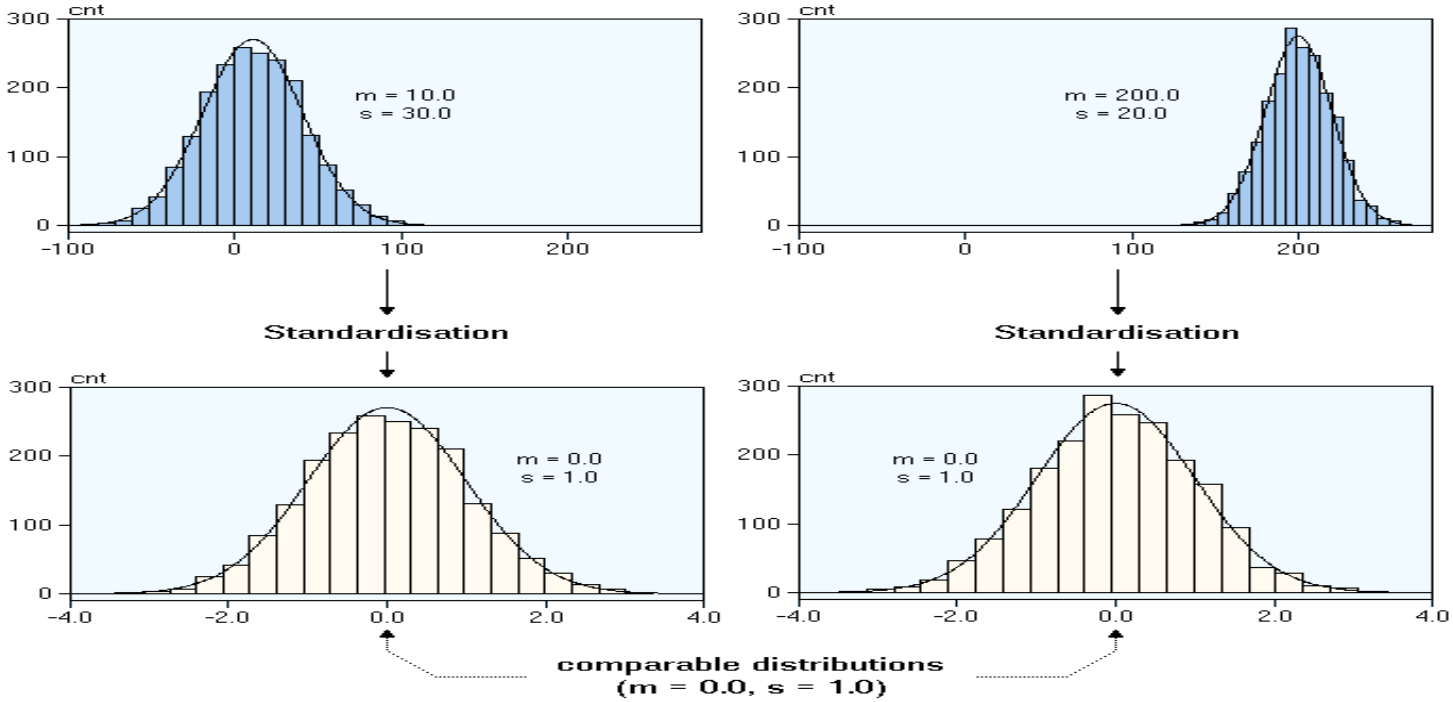
  1. Standardize the data
  2. Based on a distance metric (e.g. Euclidean) take the K nearest observations (rows) for each observation with a missing attribute
  3. For the missing attribute, take the mean (or weighted mean, or mode, or median) of the neighbors and impute this value



- **Internal Models:** Use the other features as predictors and column with missing values as target
- Other techniques to investigate: **SVD, MICE**, etc.

Normalization and standarization

- **Standardization:** move from a distribution (μ, σ) to one (0, 1) → useful for comparing variables with different orders of magnitude.



- **Normalization:** modify a non-normal distribution to approximate a normal one → for example, log (variable) for skewed variables

Adding discrimination power: Combination of variables

In general, the data we have to make the models will not be in the “correct” format for modeling, so a series of previous steps are necessary to achieve optimal results.

Raw Input

Ratios

- Financial ratios: Debt to equity ratio, price/m^2…

Agrupations

- Average sale price by province / population, average income by population…

Categorical –vs- numerical

- It may be the case that categorical variables (which do not have sorting) are best described with numerical variables (with sorting) or vice versa: e.g. "good-normal-bad" passes to categories 1, 2 and 3

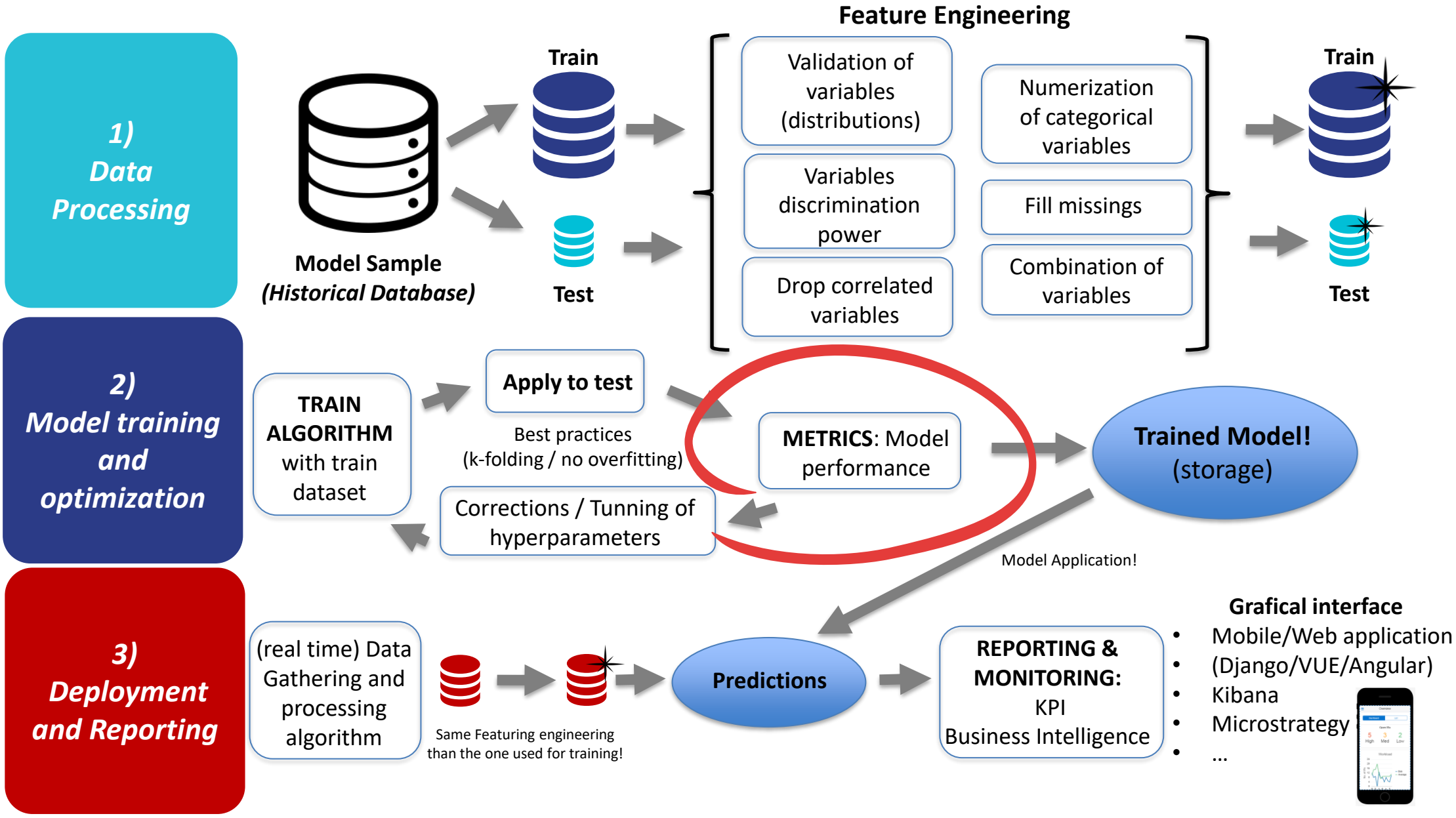
Weight of Evidence

- Very useful in modeling the probability of default, it consists of separating the variable into N groups (by quantiles) and grouping those groups with a similar behavior

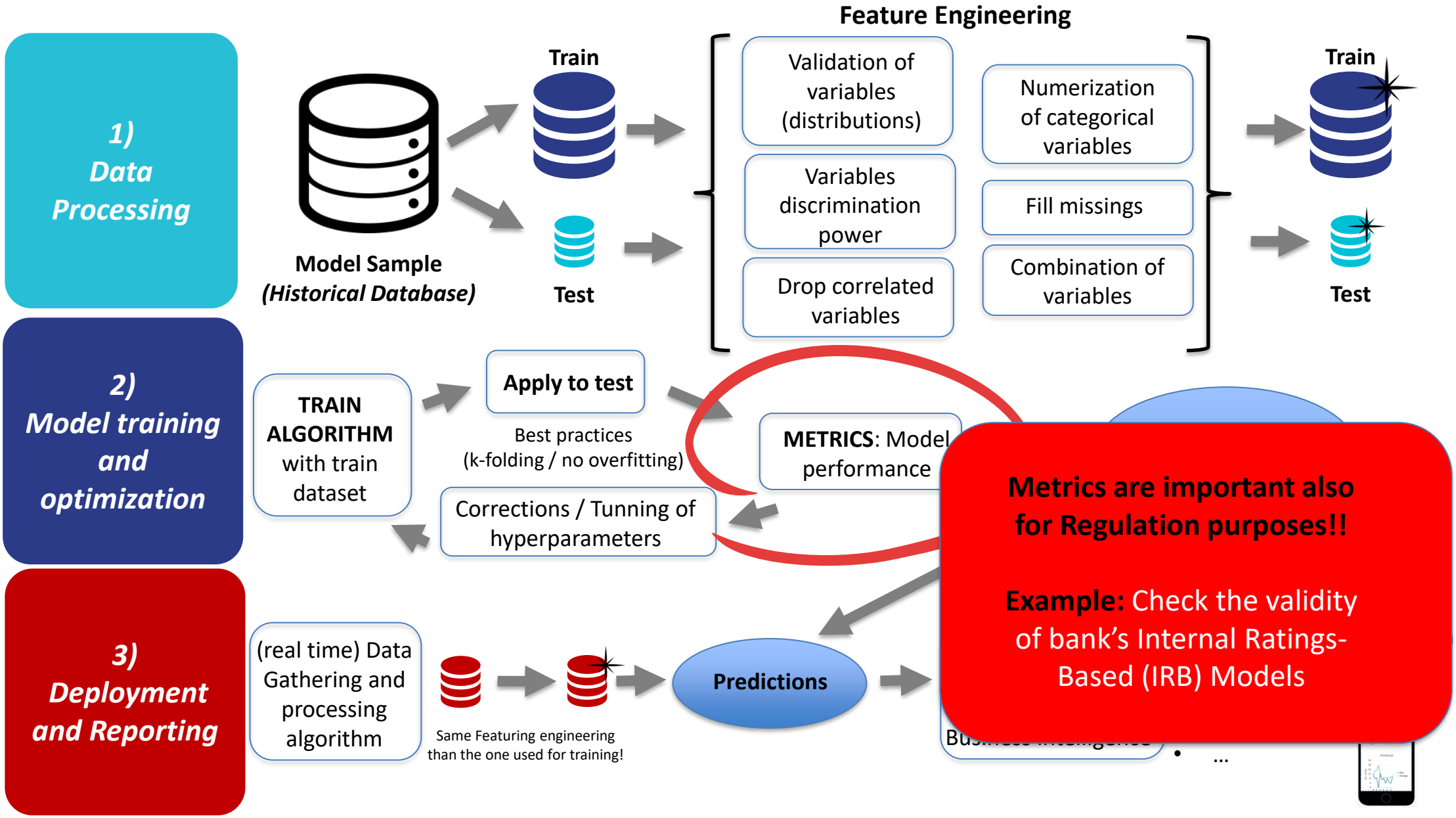
Alerts

- Separate a continuous or discrete variable based on the value of the variable that best discriminates the target, and mark as 0/1 to which group the observation belongs

General flow of a (large) modeling project of supervised learning



General flow of a (large) modeling project of supervised learning



How good is your model?

Metrics

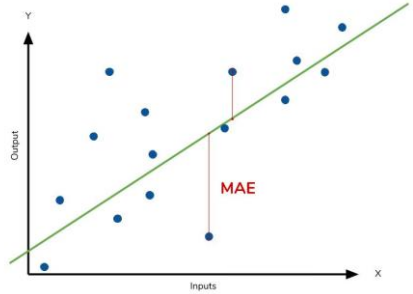
Regression Metrics

- MAE - Mean Absolute Error

Average of the absolute difference between y and ŷ (0 best, unit dependant, robust to outliers)

$$MAE = \frac{1}{n} \sum \left| y - \hat{y} \right|$$

Divide by the total number of data points; Actual output value; Predicted output value; Sum of; The absolute value of the residual



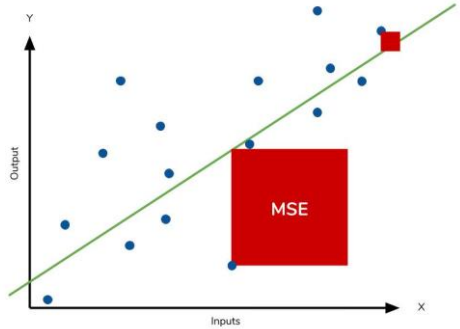
- RMSE - Root Mean Square Error

Sample Standard Deviation of the differences between y and ŷ (0 best, unit dependant)

Most used: is computationally simple, easily differentiable and present as default metric for most of the models

$$MSE = \frac{1}{n} \Sigma \left(y - \hat{y} \right)^2$$

The square of the difference between actual and predicted

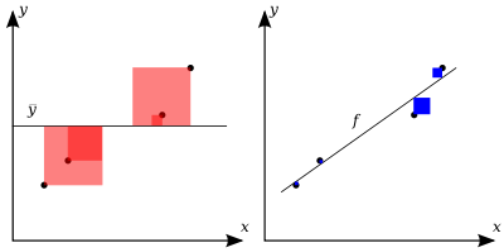


- R^2 (adj-R^2)

The numerator is MSE (average of the squares of the residuals) and the denominator is the variance in Y values. Higher the MSE, smaller the R^2 and poorer is the model (1 best)

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

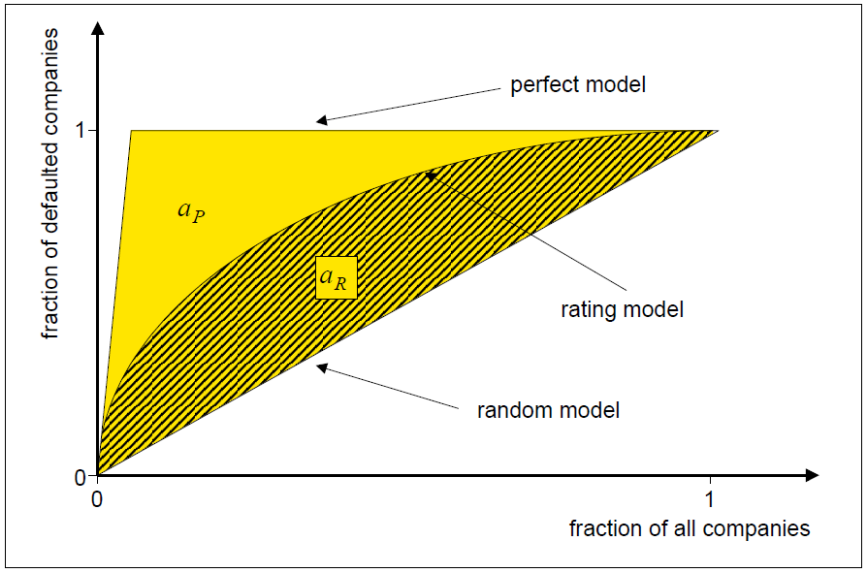
$$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$



Classification (Rank) Metrics – Accuracy ratio (GINI)

The AR metric is the most extended and accepted metric in a ranking (binary classification) model validation. The area under the CAP Curve definition is the most intuitive among many equivalent ones

Cumulative Accuracy Profile (CAP) Curve



- Range: -1 to 1 (often expressed as %)

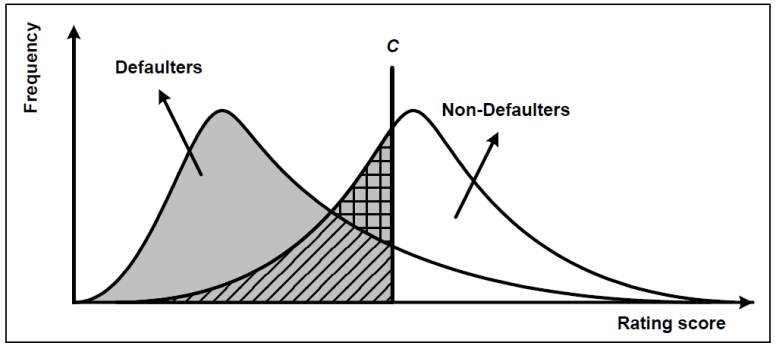
To obtain **the CAP curve of a model**, all debtors are first ordered by their respective scores, from the debtor with the lowest score (riskiest) to the debtor with the highest score. For a given fraction x of the total number of debtors the CAP curve is constructed by calculating the percentage d(x) of the defaulters whose rating scores are equal to or lower than the maximum score of fraction x. This is done for x ranging from 0% to 100%.

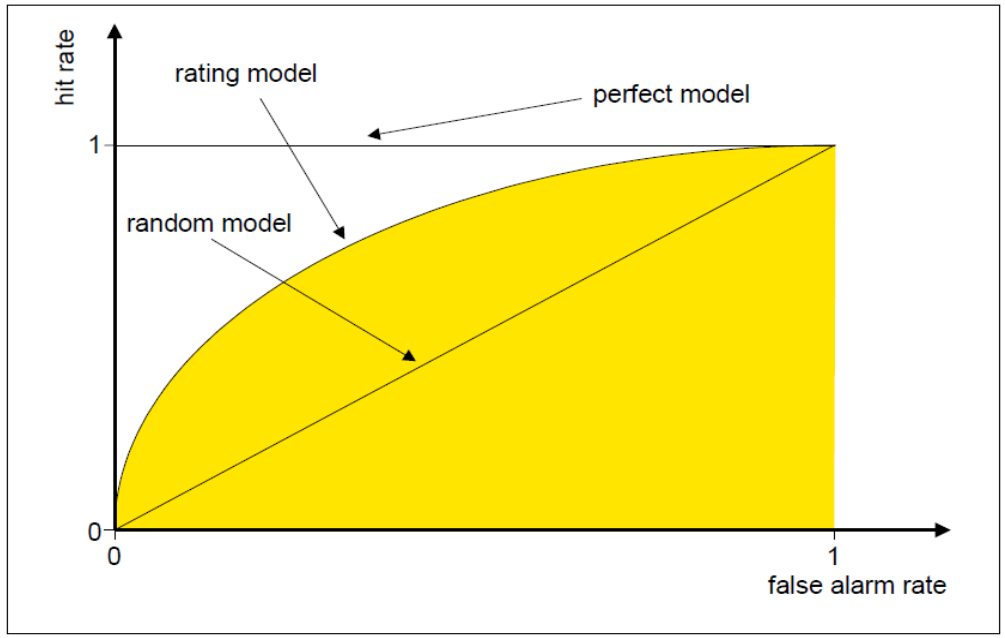
- **A perfect rating mode**l will assign the lowest scores to the defaulters. In this case the CAP is increasing linearly and then staying at one.
- **For a random mode**l without any discriminative power, the fraction x of all debtors with the lowest rating scores will contain x percent of all defaulters.
- **Real rating systems** will be somewhere in between these two extremes.

The quality of a rating system is measured by the Accuracy Ratio AR which is defined as **the ratio of the area a_R** between the CAP of the rating model and the CAP of the random model, **and the area a_P** between the CAP of the perfect rating model and the CAP of the random model:

$$AR = \frac{a_R}{a_P}$$

Classification (Rank) Metrics – AUC





- AR = 2 * AUC -1
- Range: 0 to 1 (often expressed as %)

- Given a cut-off value C, **classify each debtor with a rating score lower than C as a potential defaulter and each debtor with a rating score higher than C as a non-defaulter.**

- 4 possible results:
  1. Score < C and the debtor defaults ✓
  2. Score < C and the debtor doesn't default ✗
  3. Score > C and the debtor doesn't default ✓
  4. Score > C and the debtor defaults ✗

HR(C) = # Defaulters predicted correctly / # Total Defaulters
FAR(C) = # Non-Defaulters predicted incorrectly / # Total Non-Defaulters

$$\text{AUC} = \int_0^1 HR(FAR)\,d(FAR)$$

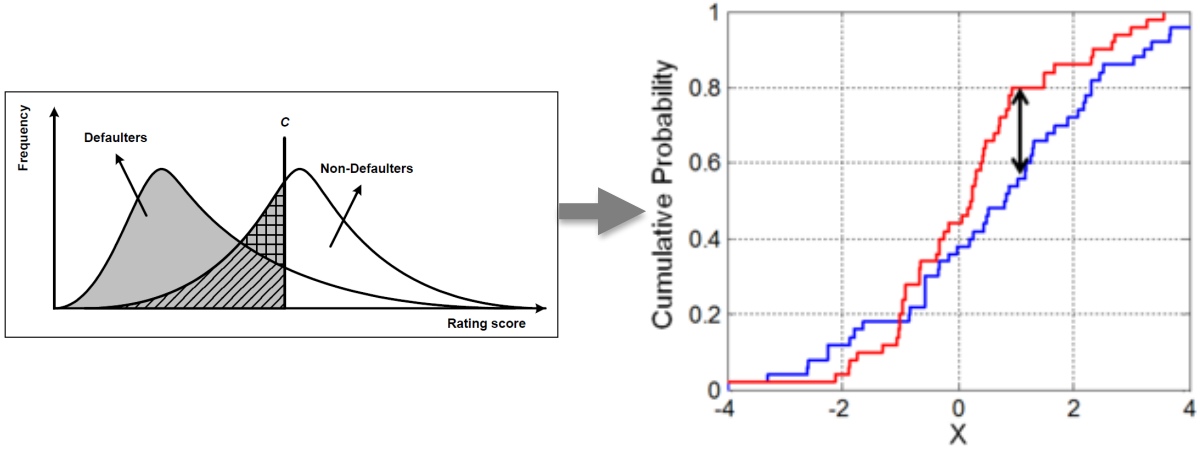
The **ROC curve is a plot of HR(C) (Hit Rate) versus FAR(C) (False Alarm Rate)** for all cut-off values C that are contained in the range of the rating scores.

Classification (Rank) Metrics

- Kolmogorov - Smirnov (Two Sample)

K-S <= AR <= AUC

[K-S statistic corresponds to the area of the largest triangle inscribed between CAP curve and diagonal]



- Somers' D (Kendall's Tau ignoring ties)

	Prof. A	Prof. B
Student 1	3	1
Student 2	7	5
Student 3	8	10

Somers'D = 3/3 (100%)

	Prof. A	Prof. B
Student 1	3	1
Student 2	8	5
Student 3	7	10

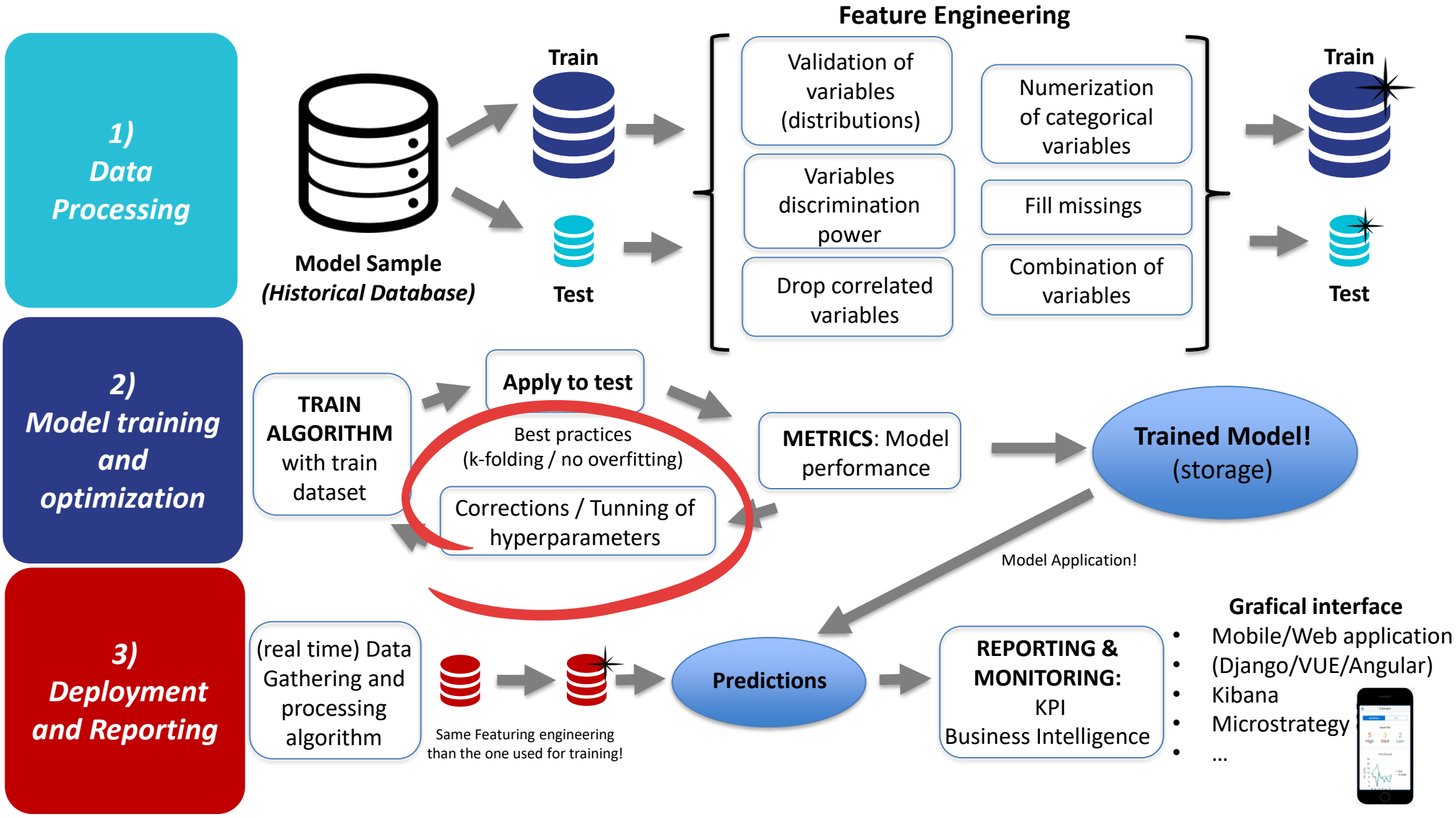
Somers'D = (+1+1-1)/3 (33%)

	Prof. A	Prof. B
Student 1	3	0
Student 2	7	1
Student 3	8	1

Somers'D = (+1+1)/2 (100%)

Somers' D when the second "professor" is 0/1 is equivalent to Accuracy Ratio

General flow of a (large) modeling project of supervised learning

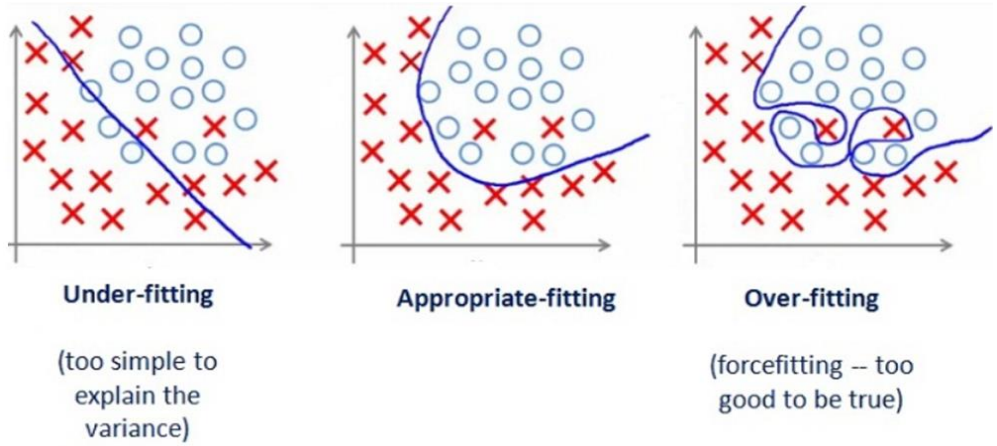


Providing a robust model

Best modelling practices

Overfitting / Underfitting

- **Overfitting occurs when a particular set of data is adjusted too much,** and the adjustment of additional data or the prediction of future observations reliably may fail.
- An **"overfitted model"** is a statistical model that contains more parameters than can be justified with data. The essence of overfitting is to have extracted, without knowing it, part of the residual variation (that is, the noise) as if that variation represented the underlying structure of the model.



- **Underfitting occurs when a statistical model cannot adequately capture the underlying structure of the data.**
- An underfitted model is a model where some parameters or terms that would appear in a correctly specified model are missing. Underfitting would occur, for example, when adjusting a linear model to nonlinear data. Such a model will tend to have poor predictive performance.

Bias / Variance tradeoff

The bias-variance tradeoff is a central problem in supervised learning. Ideally, one wants to choose a model that accurately captures the regularities in their training data, but also generalizes the unobserved data well. Unfortunately, it is usually impossible to do both simultaneously. High variance learning methods may be able to represent your train set well, but you run the risk of over-adapting to noisy or non-representative train data. In contrast, algorithms with low variance generally produce simpler models that do not tend to "overfit", but can "underfit" their training data, since they fail to capture important regularities.

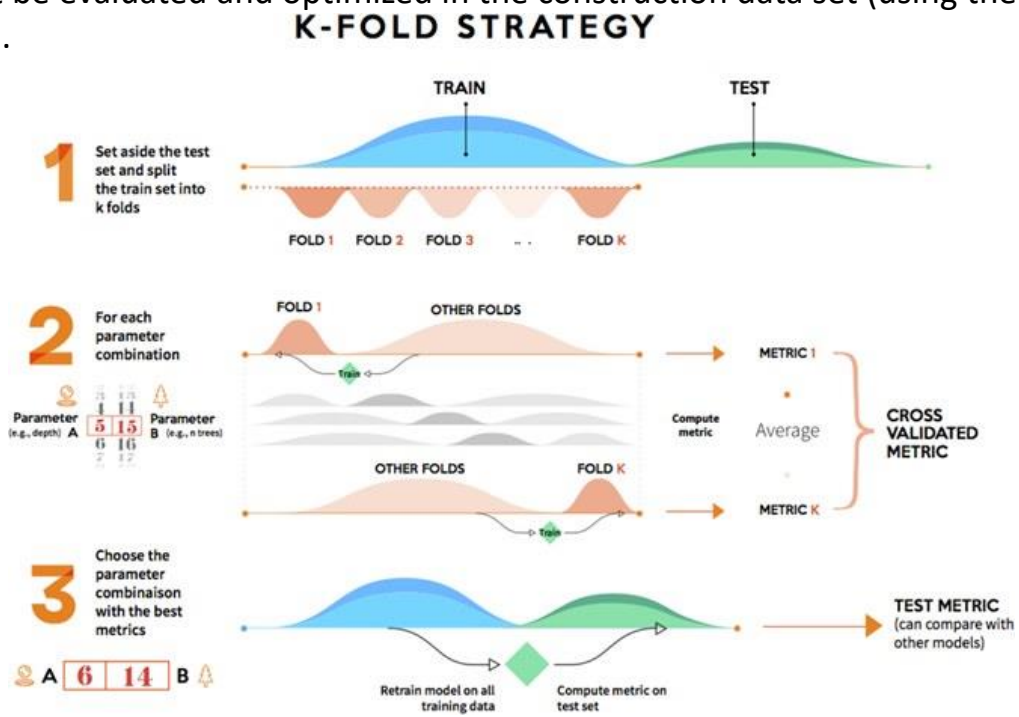
Best practices to avoid overfitting

Train / Test (or validation) Split

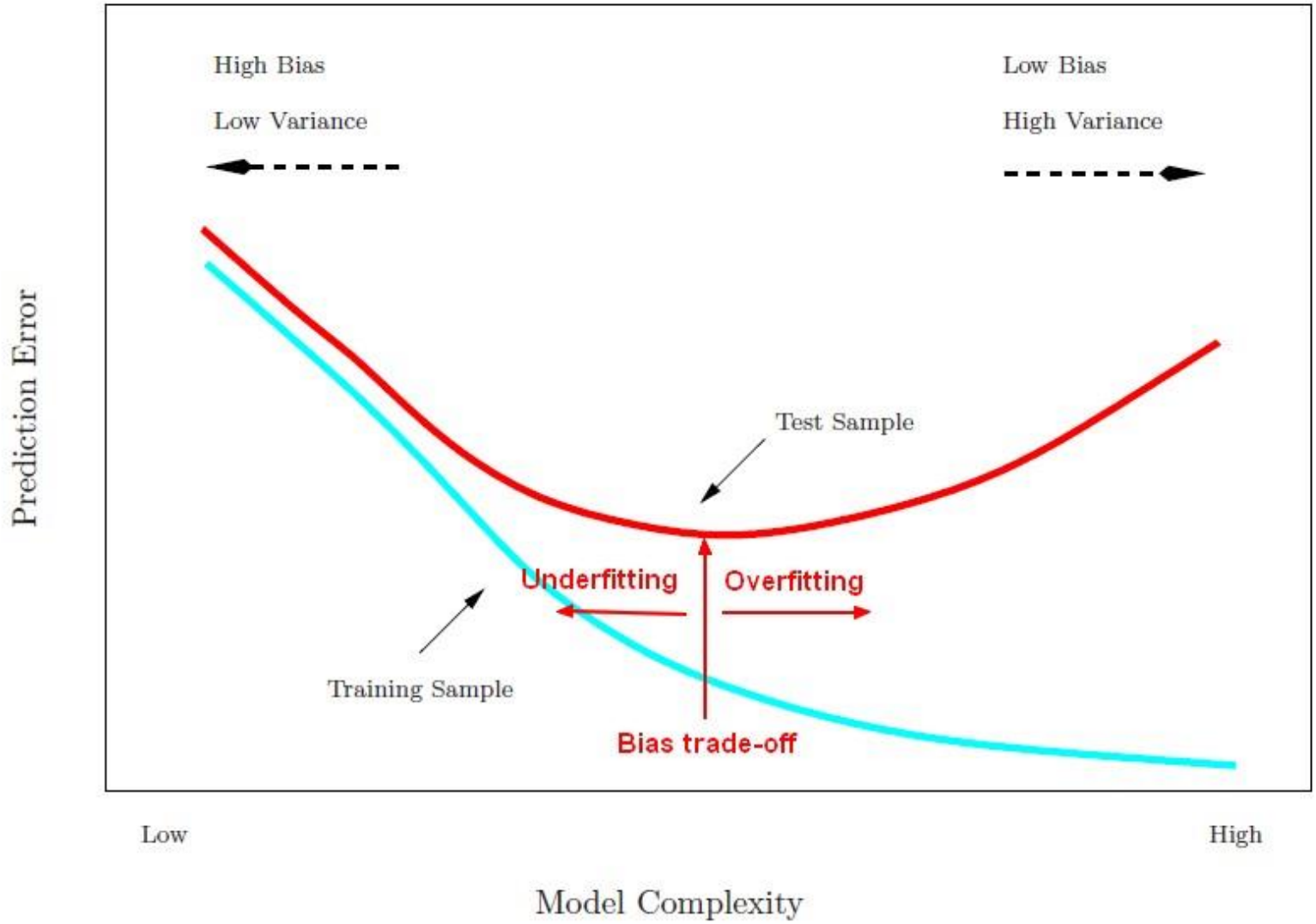
- The construction database should be divided into 70% of the trained set and 30% of the test set (stratified). 70/30 is the usual one but there are many options.
- The test set is usually an "out-of-sample" or "out-of-time" set where true labels are known but ignored until the end of the modeling process. The test set is also called "validation set".
- All transformations should be estimated in training (and only with training information) and then applied to the test (eg, when standardization of numerical variables is performed, we obtain the mean and standard deviations of the training set and apply these parameters altest set)
- Model tuning (imputation strategies for missing data, feature transformations, feature creation, categorical to numerical transformations, adjustment of hyperparameters, etc.) must be evaluated and optimized in the construction data set (using the training dataset / division of evidence or by cross validation).

Train / Test split and K-Fold Cross-Validation

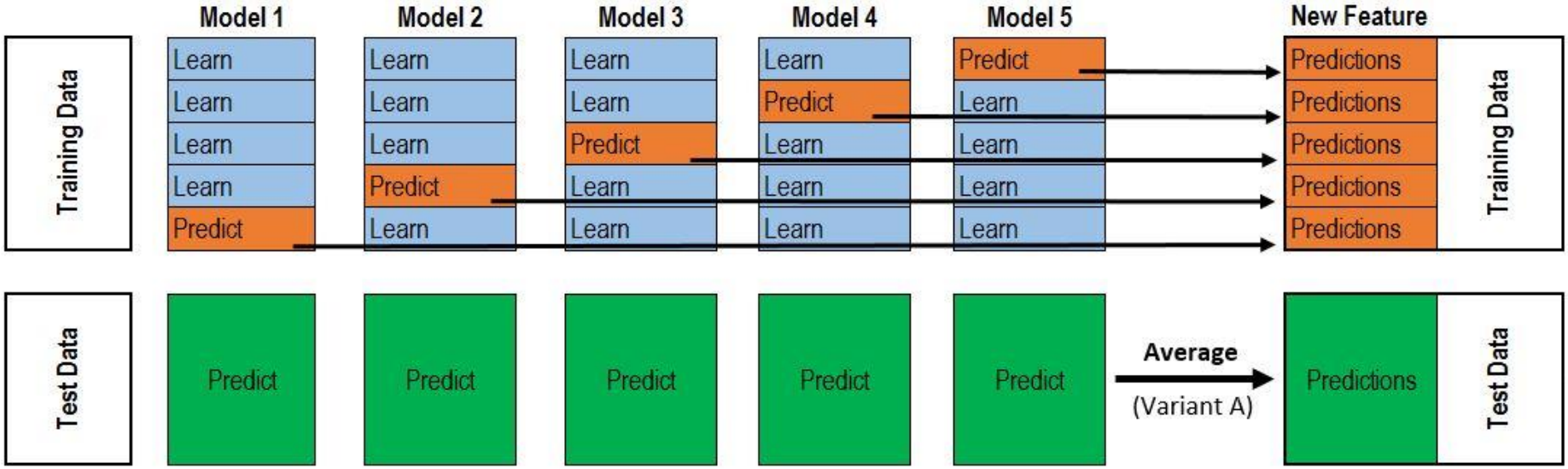
- The model is adjusted in 1 test set or in k test sets (a partition of the trained set). The final model is trained on all the data trained with the optimal hyperparameters



Underfitting -vs- Overfitting



K-folding technique (another picture…)

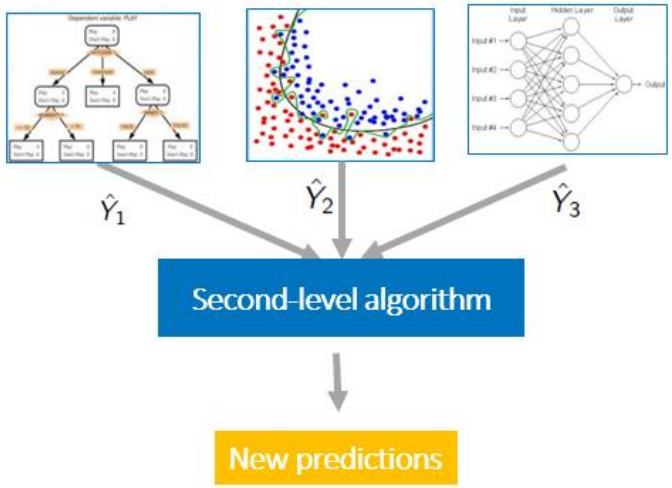


Stacking technique

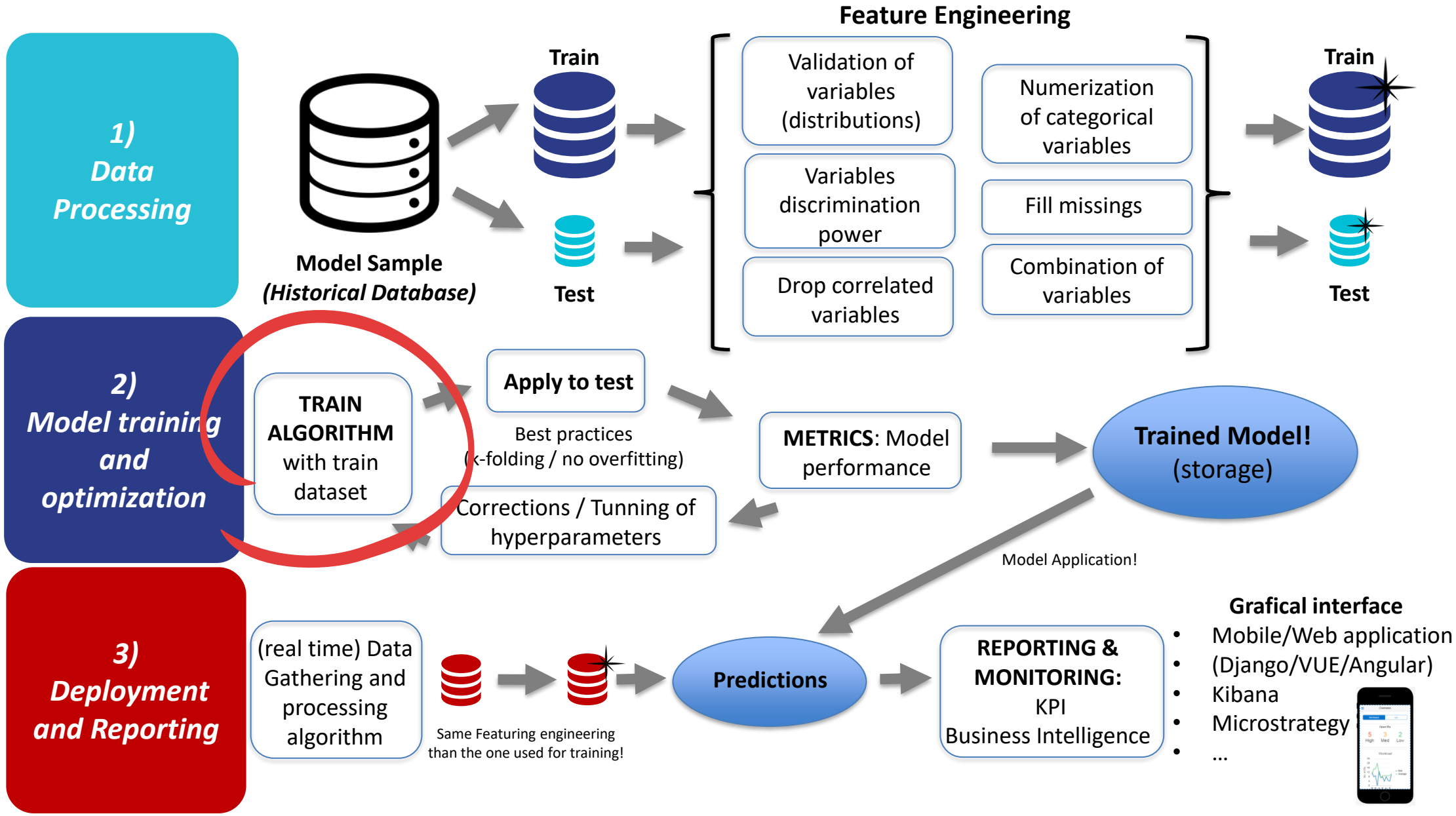
Useful but dangerous!

It is a double edge weapon…

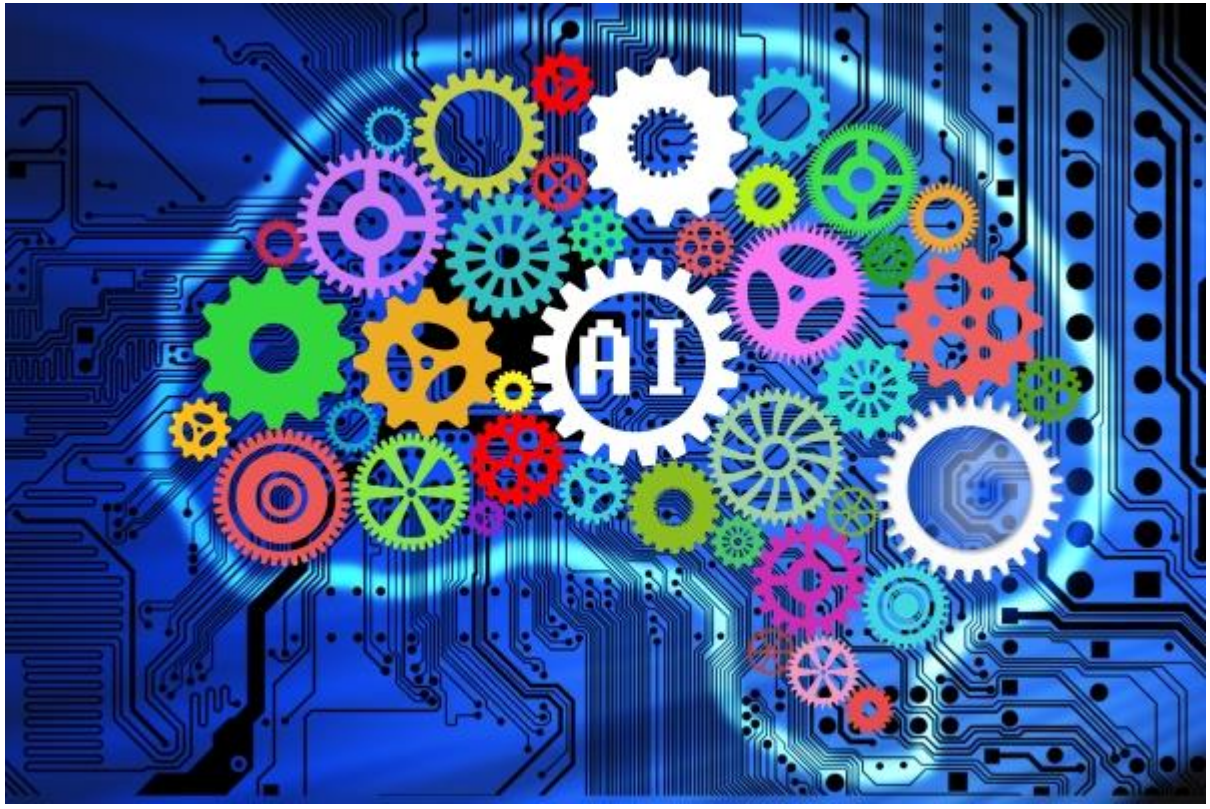
- Create different models and then a level 1 model to "join" the predictions.
- The **empirical** effect is the improvement of the performance of the model. But when done wrong… introduces biases and overfitting to the model…



General flow of a (large) modeling project of supervised learning



Machine Learning Algorithms & Models



1. **Linear Models**
    - 1.1. Basic algorithms of classification and regression
    - 2.2. Application to scoring and real estate models

2. **Non-linear models: Ensemble Trees algorithms**
    - 2.1. Random forests
    - 2.2. Boosted Decision Trees

3 . **Non-linear models: Artificial Neural Networks**
    - 3.1. Basics: The Multi-Layer Perceptron
    - 3.2. Convolutional Neural Networks
        - 3.2.1. Image Recognition
        - 3.2.2. Natural Language (text) Processing
        - 3.2.3. Speech Processing

Main modeling Approaches & (pre-programmed) Algorithms

Regression and Classification (tabular data)

Tabular data → rows (observations) and columns (features)

- **Linear/Logistic Regression** (Ordinary Least Squares / Maximum Likelihood Estimation)
- **Regularized Linear Models** (LASSO, Ridge, ElasticNet)
- **Random Forest**
- **Gradient Boosting Machines** (XGBoost, LightGBM, Catboost)
- **Standard Neural Network** (Multi-Layer Perceptron)

Unsupervised

Tabular data → rows (observations) and columns (features)

- **K-means** (Clustering)
- **Principal Components Analysis (PCA)** (Dimensionality Reduction)

Image Recognition (pixel data)

Each observation is a sequence of numbers between 0 and 255 for each color (RGB):

- **Standard Neural Network** (Multi-Layer Perceptron)
- **2D Convolutional NN**

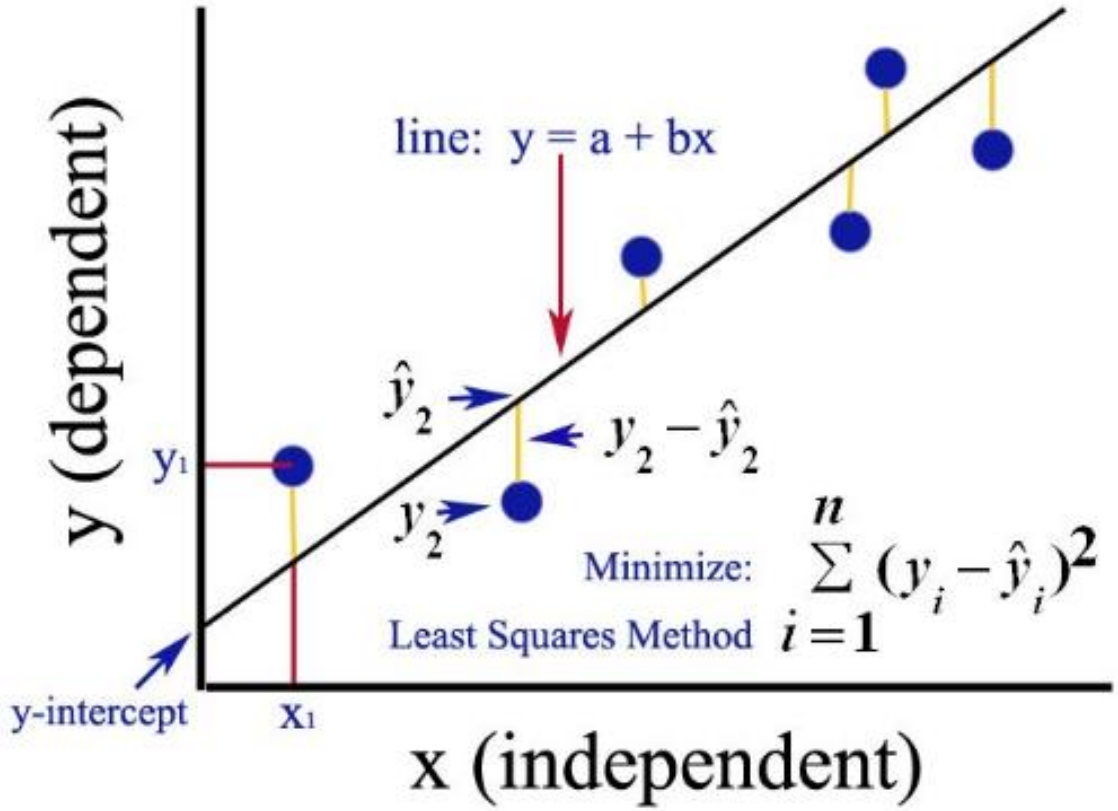
Text Sentiment Analysis (words)

Each observation is a text in some human language

- **Standard Classification with a previous word sparse embedding** (Natural Language Processing: Stop Words, Stemming and Vectorizer)
- **1D Convolutional NN**
- **Recurrent NN** (LSTM: Long Short-Term Memory)

Linear Regression (Ordinary Least Squares)

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m$$



https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Examples

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0000...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Logistic Regression (Classification)

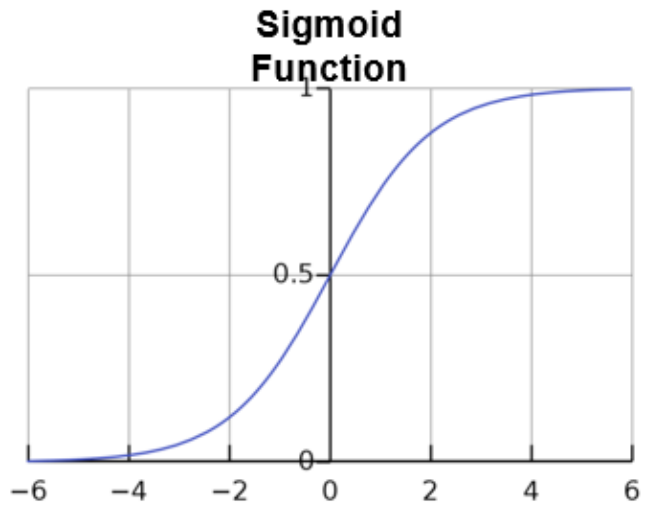
$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m$$

$$L(\beta) = \prod_{x_i \text{ where } y_i=1} p(x_i) \prod_{x_i \text{ where } y_i=0} (1 - p(x_i))$$

$$L(\beta) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$$

$$l(\beta) = \sum_{i=1}^{n} y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))$$

$$\max_{\beta}(l(\beta)) \text{ or } \min_{\beta}(-l(\beta))$$



$$p(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_m x_m)}}$$

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Examples

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0, solver='lbfgs',
...                          multi_class='multinomial').fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Regularized Linear Models

Regularization, significantly reduces the variance of the model, without substantial increase in its bias.

- **LASSO**

(Least Absolute Shrinkage and Selection Operator)

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

L1 norm

- Ridge

(name given in the original paper, as "crest" synonym)

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

L2 norm

OLS model won't generalize well for a data set different than its training data

Tuning parameter λ → controls the impact on bias and variance:

As λ rises, it reduces the value of coefficients and thus reduces the variance. **Till a point, this increase in λ is beneficial as it is only reducing the variance (hence avoiding overfitting), without loosing any important properties in the data.** But after certain value, the model starts loosing important properties, giving rise to bias in the model and thus underfitting. Therefore, the value of λ should be carefully selected.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

Examples

```
>>> from sklearn import linear_model
>>> clf = linear_model.Lasso(alpha=0.1)
>>> clf.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
...
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
>>> print(clf.coef_)
[0.85 0.  ]
>>> print(clf.intercept_)
0.15...
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

Examples

```
>>> from sklearn.linear_model import Ridge
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> clf = Ridge(alpha=1.0)
>>> clf.fit(X, y)
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Example: Credit Risk Scoring classical (i.e. linear) models (1/2)

The objective of such kind of models is to provide an score for each physical/juridical entity asking for a credit to a given banking firm. **The model quantifies the Probability of Default (PD).** Its main advantages are:

- Easy understanding of the underlying behavior of the model.
- Measurable contribution of each of the variable (and their categories) categories to the final result.
- User friendly result (Intuitive result: The more score, the better the customer is).
- Easy maintenance.

Firsts steps having a training set:

The TARGET variable is Binary:
- 0 non-defaulter
- 1 defaulter

$$\mathrm{WoE_a} = \ln\left(\frac{\frac{N_a^{good}}{N_{total}^{good}}}{\frac{N_a^{bad}}{N_{total}^{bad}}}\right) = \ln\left(\frac{\%good_a}{\%bad_a}\right)$$

POOL of variables

Selection of the suitable ones for the problem

- Demographic
- Socio-economic
- External-Bureaus
- …

Categorization of the variables

Each variable will be divided in buckets or categories. A priori the number of categories is arbitrary, however the categories must contain enough population to have statistical sense.

Substitute each category for the equivalet Weight of Evidence (WoE)

The WoE must have economical sense. In case it has not sense, revise the categorization.

Possibility of crossing variables combining their categories

Example: Credit Risk Scoring classical (i.e. linear) models (2/2)

At this point one has the "WoEized" variables, the original form of the variables is usually abandoned.

Check the univariant Power Index (PI)
It is basically a Sommer's index

→

Drop Correlated Variables

→

Logistic Regression

Coefficients + Intercept value

$$\ln\left(\frac{1}{1-p}\right) = \beta_0 + \sum \beta_k x^k$$

All ingredients are then available to compute the **score** for the category (a) of the variable (i). The offset is usually subjective (i.e. 500)

$$Score_{ia} = \frac{Offset}{n} + \frac{Factor}{\ln(2)}\left(\frac{\beta_0}{n} - \beta_i WOE^{ia}\right)$$

The weight of each variable in the model is found with the **Score Contribution Ratio (SCR)**:

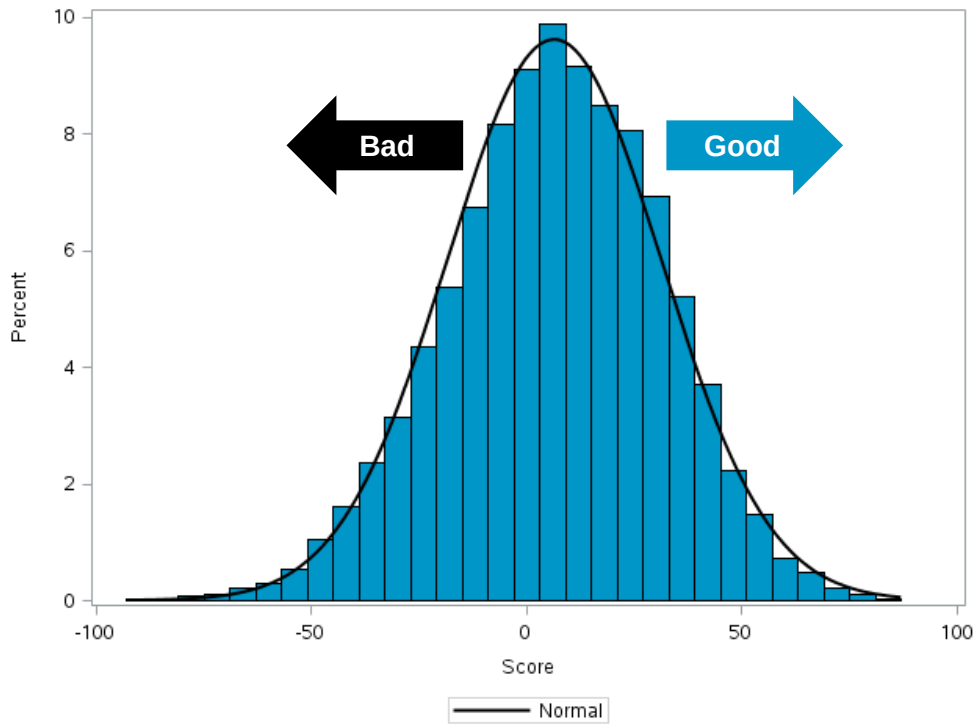
$$SCR_i = \frac{\sum_a^{a=categ\ variable_i}\left[\left(Score_{ia}^2 \cdot \%N_a\right) - \left(Score_{ia} \cdot \%N_a\right)^2\right]}{\sum_i^{i=total\ variables}\sum_a^{a=categ\ variable_i}\left[\left(Score_{ia}^2 \cdot \%N_a\right) - \left(Score_{ia} \cdot \%N_a\right)^2\right]}$$

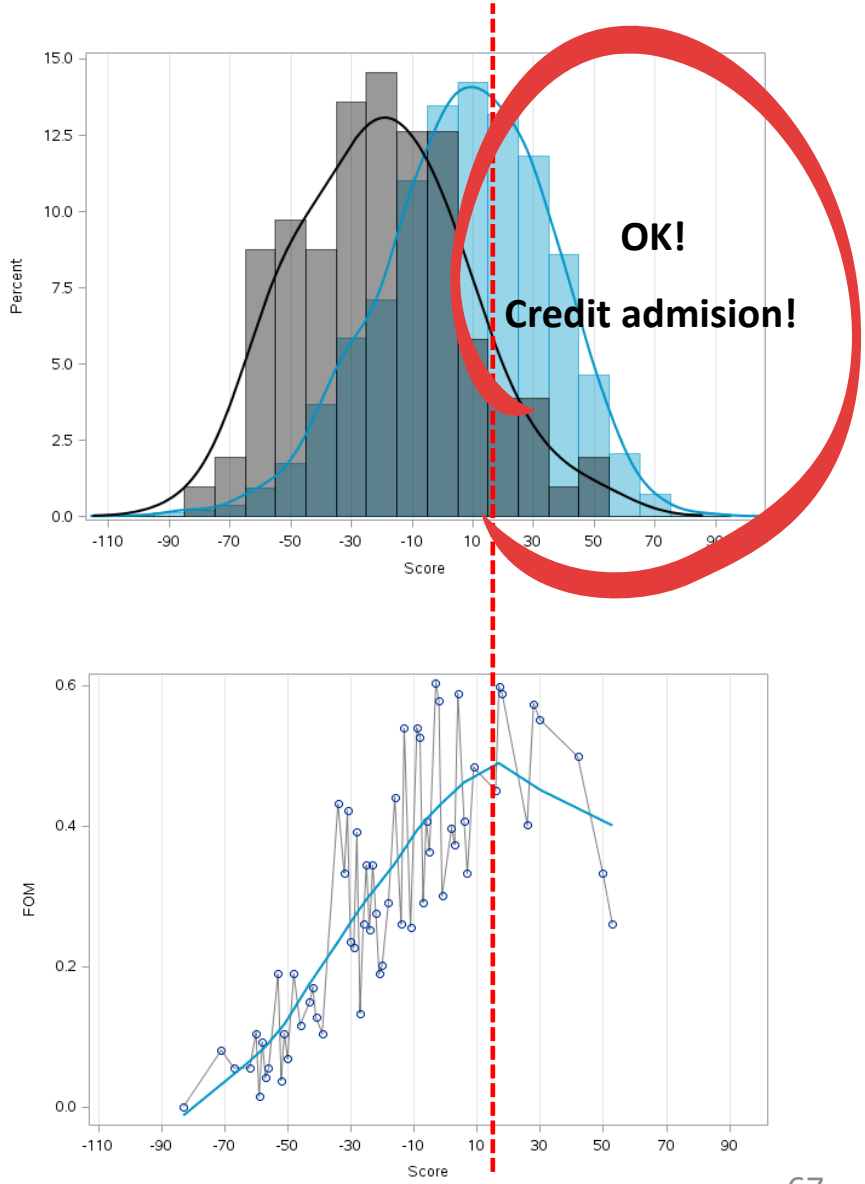
Example of scorecard

-24
-7
-3
17
0
17
7
-11
0
11
-9
-11
0
-15
3
12
0
4
-1
-3
-8
0
2
12
-7
-1
-2
3
0

-9
-5
-3
0
3
4
0
11
-7
-8
-2
-6
0
9
0
3
0
-1
-2
-7
0
5
10
15
1
0
5
12
16
1
-5
-29
-9
0
3
4

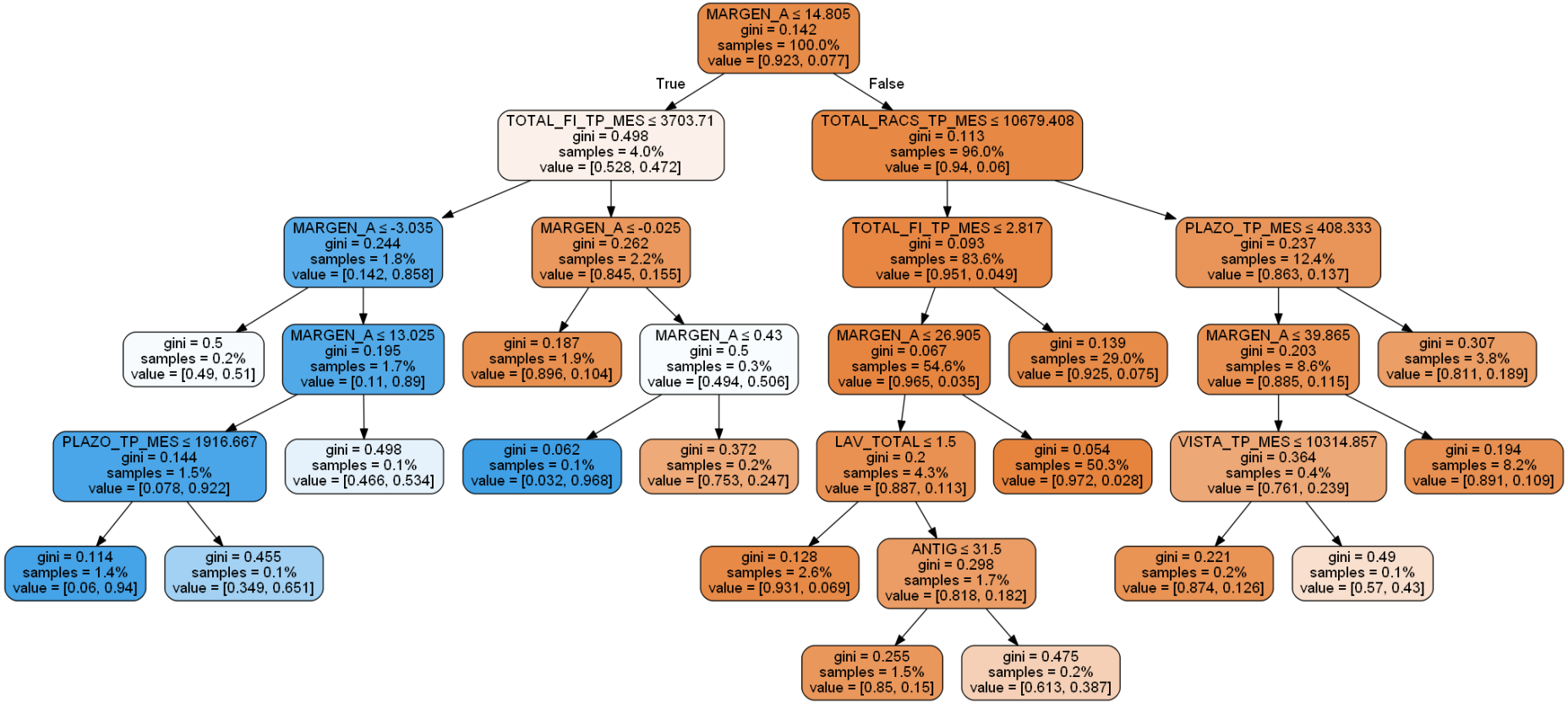
Example of Final result and credit admission criteria





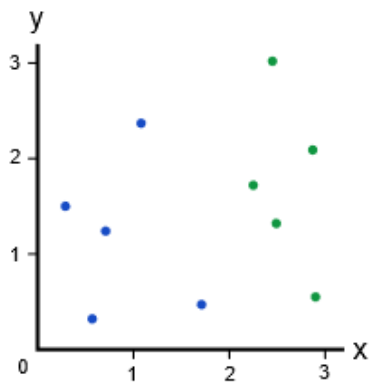
Simulation! Very useful that the final score (sum of individual score is not only centerd in one value

Non-Linear models: Classification and Regression trees



Classification and Regression trees

In a Classification problem the (Max) **Gini Gain** is one of the criteria to determine the optimal split of a node

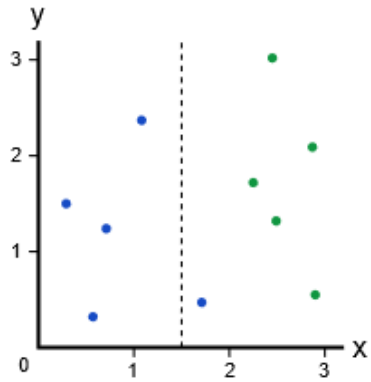


Gini Impurity is the probability of *incorrectly* classifying a randomly chosen element in the dataset if it were randomly labeled *according to the class distribution* in the dataset

$$G = \sum_{i=1}^{C} p(i)(1 - p(i))$$

P(i) is the probability of picking a datapoint with class i

$$G = 0.5(1 - 0.5) + 0.5(1 - 0.5) = 0.5$$



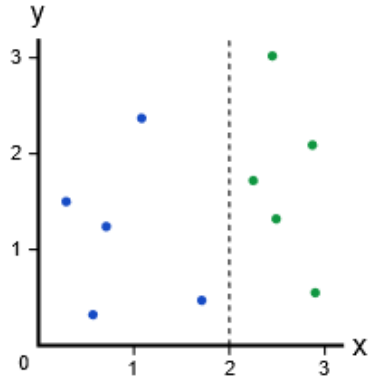
Imperfect Split

$$G_{\text{left}} = 1(1 - 1) + 0(1 - 0) = 0$$

$$G_{\text{right}} = \frac{1}{6}(1 - \frac{1}{6}) + \frac{5}{6}(1 - \frac{5}{6}) = 0.278$$

$$\text{Gain} = 0.5 - (40\%(0) + 60\%(0.278)) = 0.333$$

Gini Gain is obtained by weighting the impurity of each branch by how many elements it has and subtracting it from the impurity before the split



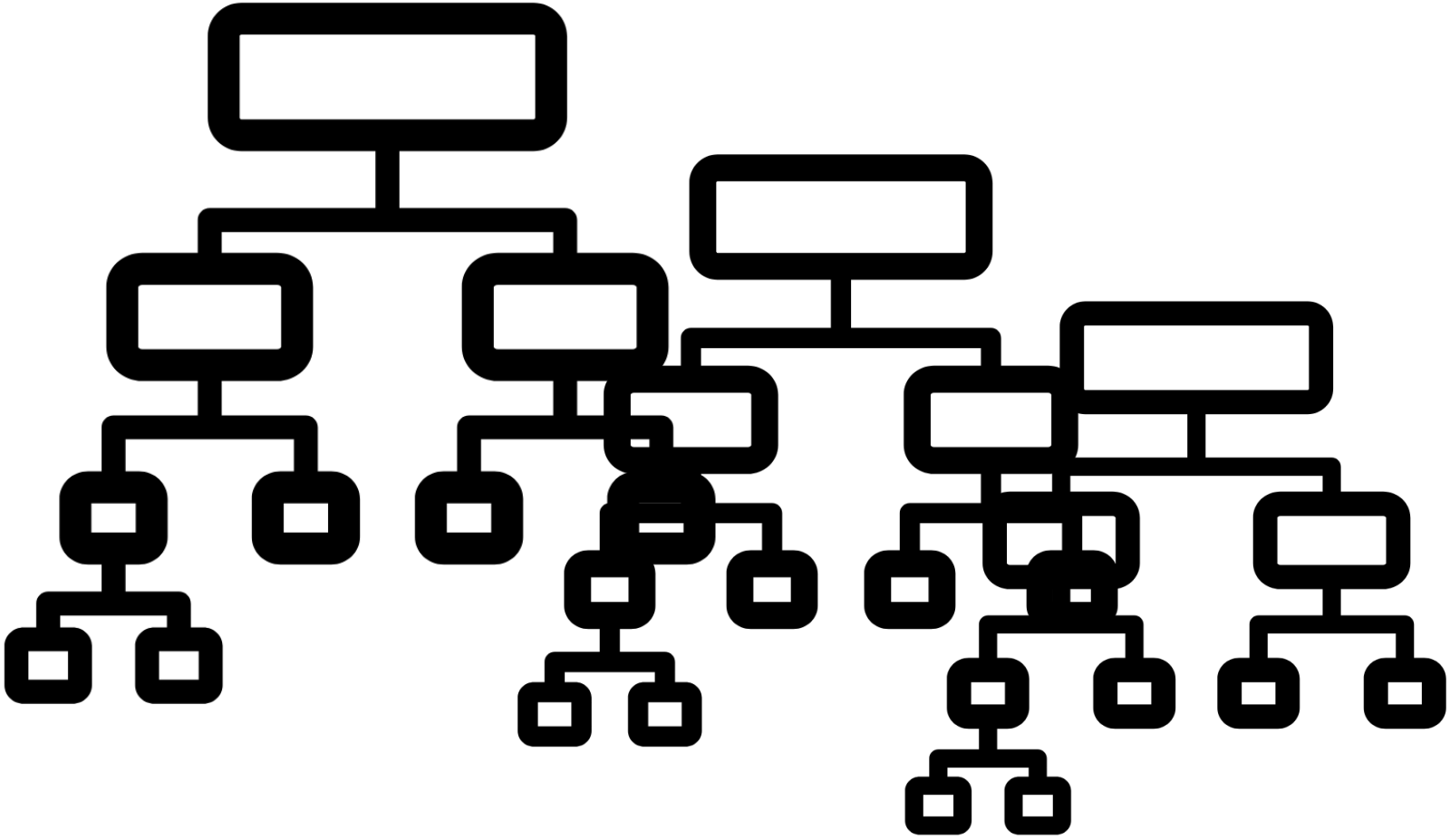
Perfect Split

$$G_{\text{left}} = 1(1 - 1) + 0(1 - 0) = 0$$

$$G_{\text{right}} = 1(1 - 1) + 0(1 - 0) = 0$$

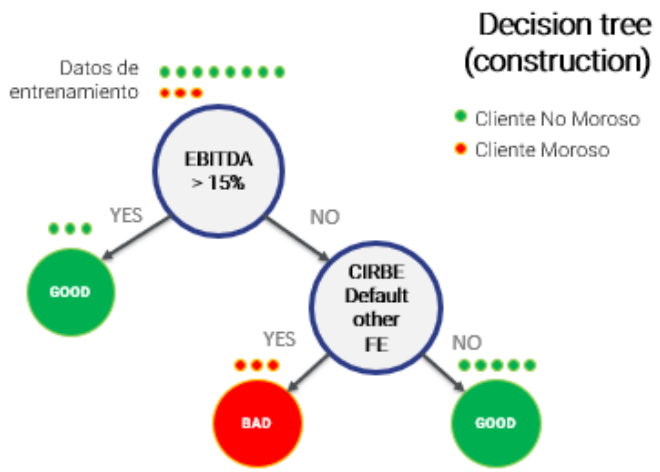
$$\text{Gain} = 0.5 - (40\%(0) + 60\%(0)) = 0.5$$

Ensemble trees

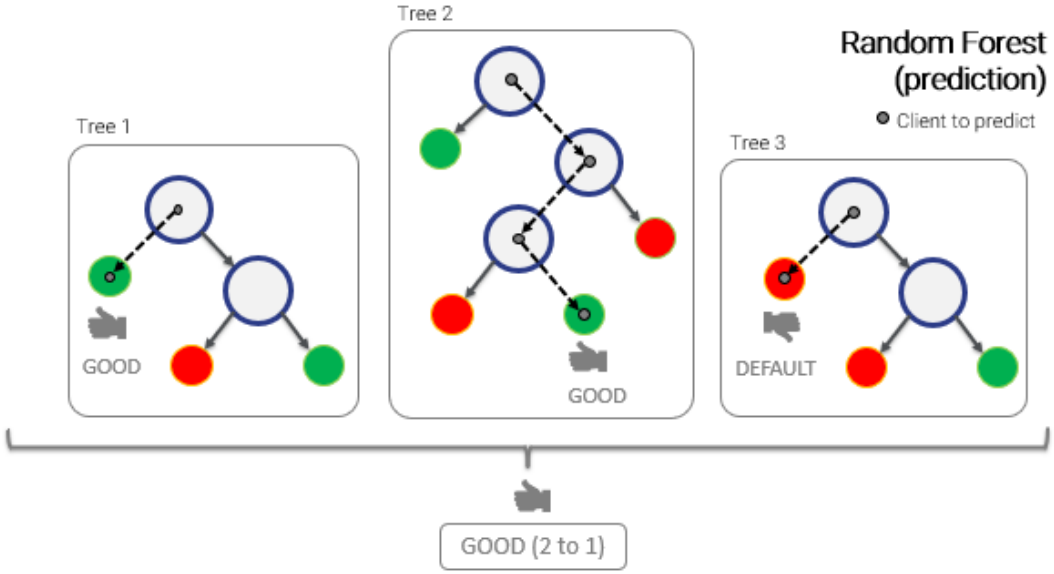


Random Forest

It is an ensemble of decision trees. **Each tree has been grown with different data** (available samples, columns or both) **so that they are experts in different type of observations** (bagging). To take a decision, each tree evaluates an observation and then vote or aggregate the predictions



A **Decision Tree** looks at the variables and identifies the optimal "questions" to separate the observations till a leaf node where (almost) all of the observations are of the same "color"



A **Random Forest** is a collection of many Decision Trees that are grown with subsamples of the total data (bootstrapping)

When it's time to predict, an observation is "thrown" through every tree and when answering the questions of the modes, it reaches a leaf where the tree has already made a decision. **The final decision is produced by the aggregation of the output of all the trees**

Random Forest

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
```

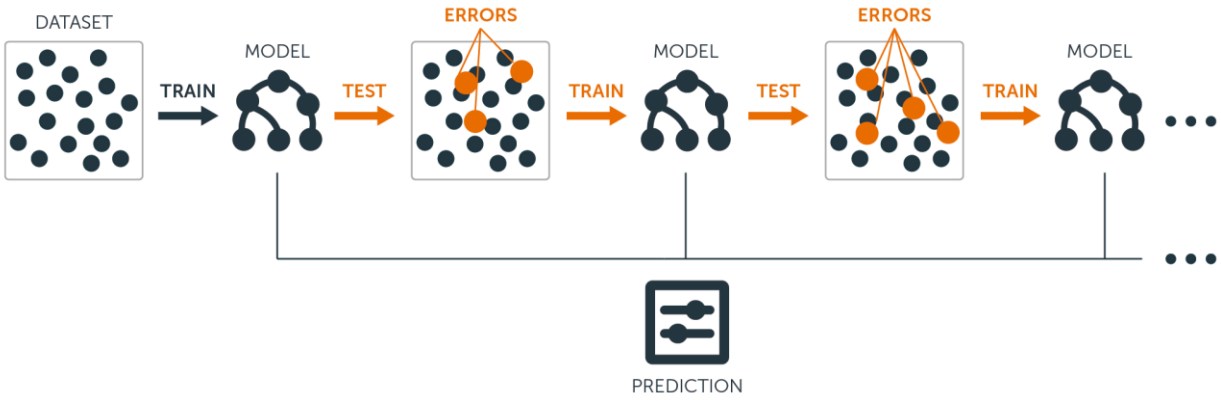
```
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(max_depth=2, random_state=0)
>>> print(clf.feature_importances_)
[0.14205973 0.76664038 0.0282433  0.06305659]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.datasets import make_regression
```

```
>>> X, y = make_regression(n_features=4, n_informative=2,
...                        random_state=0, shuffle=False)
>>> regr = RandomForestRegressor(max_depth=2, random_state=0)
>>> regr.fit(X, y)
RandomForestRegressor(max_depth=2, random_state=0)
>>> print(regr.feature_importances_)
[0.18146984 0.81473937 0.00145312 0.00233767]
>>> print(regr.predict([[0, 0, 0, 0]]))
[-8.32987858]
```

Boosted Decision Trees: Gradient Boosting

Boosting consists in “adapting” the data in each iteration to better predict those observations that in the previous iterations have not been correctly classified.



1. A first model is trained (the most common case is a tree model)
2. The differences between the real target and the value predicted by the first model are analyzed
3. In the next iteration, more weight is given to the incorrectly predicted observations and less to the correctly predicted observations
4. This algorithm is repeated N times, and the result of the trained models is combined

Algorithms

There are several Boosting algorithms that implement small differences between them
Some of these algorithms, such as LightGBM or CatBoost, allow the “automatic treatment” of missing values and categorical variables
In general, you can "test" the different types of algorithms and check which one works best with the data being treated.

XGBoost

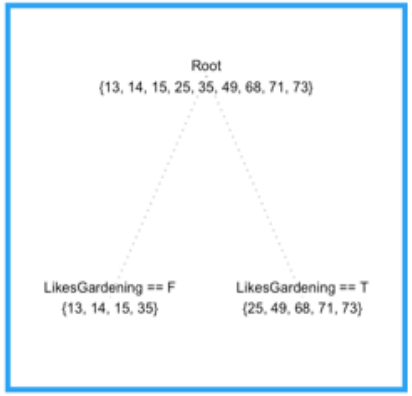
CatBoost

LightGBM

Boosted Decision Trees: Gradient Boosting

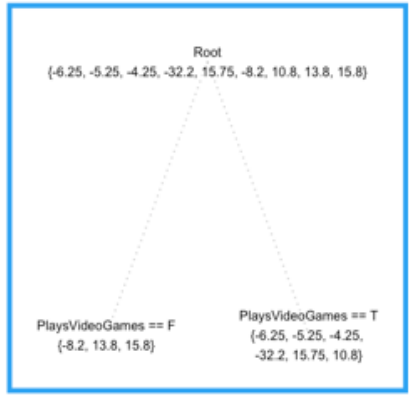
ID	Age	LikesGardening	PlaysVideoGames	LikesHats
1	13	FALSE	TRUE	TRUE
2	14	FALSE	TRUE	FALSE
3	15	FALSE	TRUE	FALSE
4	25	TRUE	TRUE	TRUE
5	35	FALSE	TRUE	TRUE
6	49	TRUE	FALSE	FALSE
7	68	TRUE	TRUE	TRUE
8	71	TRUE	FALSE	FALSE
9	73	TRUE	FALSE	TRUE

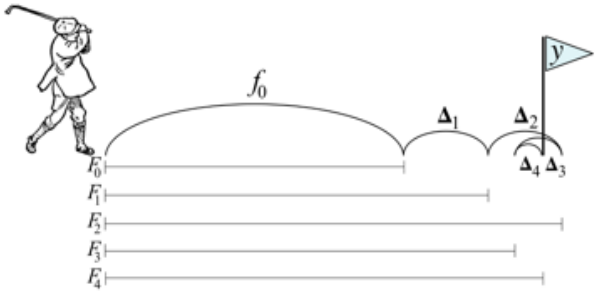
Tree 1



ID	Age	Tree1 Prediction	Tree1 Residual
1	13	19.25	-6.25
2	14	19.25	-5.25
3	15	19.25	-4.25
4	25	57.2	-32.2
5	35	19.25	15.75
6	49	57.2	-8.2
7	68	57.2	10.8
8	71	57.2	13.8
9	73	57.2	15.8

Tree2





ID	Age	Tree1 Prediction	Tree1 Residual	Tree2 Prediction	Combined Pred.	Final Residual
1	13	19.25	-6.25	-3.567	15.68	2.683
2	14	19.25	-5.25	-3.567	15.68	1.683
3	15	19.25	-4.25	-3.567	15.68	0.6833
4	25	57.2	-32.2	-3.567	53.63	28.63
5	35	19.25	15.75	-3.567	15.68	-19.32
6	49	57.2	-8.2	7.133	64.33	15.33
7	68	57.2	10.8	-3.567	53.63	-14.37
8	71	57.2	13.8	7.133	64.33	-6.667
9	73	57.2	15.8	7.133	64.33	-8.667

Boosted Decision Trees: Gradient Boosting algorithms

March, 2014

XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016

Jan, 2017

Microsoft released first stable version of LightGBM

April, 2017

Yandex, one of Russia's leading tech companies open sources CatBoost

Gradient Boosting algorithms: XGBoost

```
##########################################################################
import xgboost as xgb

# Dmatrix format
dtrain = xgb.DMatrix(train_text_counts,
                     label = target,
                     feature_names=stem_vectorizer.get_feature_names())

# Hyperparameter specification
params = {"objective":'binary:logistic',
          "booster":'gbtree',
          "eval_metric":'auc',
          "min_child_weight":0,
          "max_depth":5,
          "learning_rate":0.01,
          "colsample_bytree":0.7,
          "subsample":0.7,
          "gamma":5,
          "alpha":0.1,
          "silent":True,
          "verbose_eval":True}

num_folds=5
# Train the model CV to test parameters
model_cv = xgb.cv(params,
                  dtrain,
                  nfold=num_folds,
                  num_boost_round=3000,
                  early_stopping_rounds=100,
                  verbose_eval=50)

```

Gradient Boosting algorithms: LightGBM

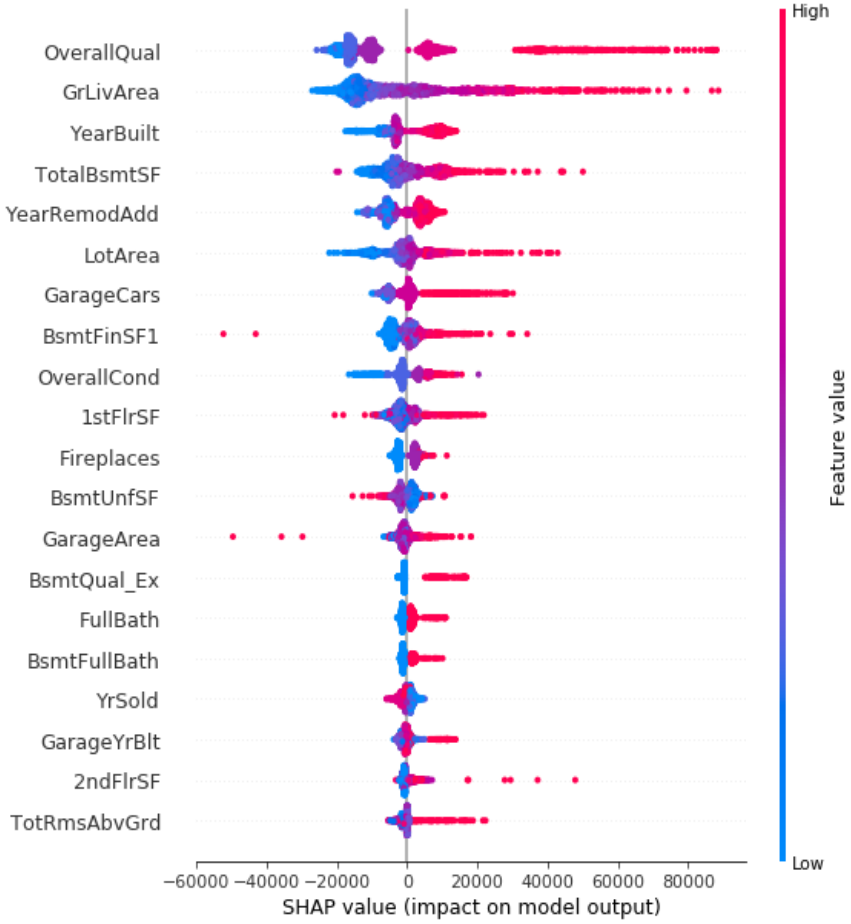
```
##############################################################################
from lightgbm import LGBMClassifier

# Parameters of the CV:
RS=2305 # Seed for the fold partition and model
n_folds=5 # Number of CV folds

# Model parameters:
params = {'objective': 'binary',
          'learning_rate': 0.005,
          'num_leaves': 40,
          'min_data_in_leaf': 5,
          'colsample_bytree': 0.7,
          'max_bin': 20,
          'random_seed': RS}

# Define the model with the parameteres and perform a search of the oprimal
# number of trees
model_lightgbm=LGBMClassifier()
model_lightgbm.set_params(**params)
iter=[2000,2500,3000]

print('\nLightGBM CV...')
print('#######################################################')
scores=[]
for nrounds in iter:
    model_lightgbm.set_params(n_estimators=nrounds)
    print('\nn rounds: ',nrounds)
    Pred_train, Pred_test, s = Model_cv(model_lightgbm,n_folds,X_train,X_test,
                                        Y_train,RS,makepred=True)
```

Gradient Boosting algorithms: CatBoost

```
##############################################################################
model_catboost_val = CatBoostClassifier(
          eval_metric='AUC',
          objective='Logloss',
          iterations=20000, # High value, to find the best model
          od_type='Iter', # Overfitting detector based on iterations (when validation set is used)
          learning_rate=0.1, # learning rate (lower -> slower but better prediction)
          depth=5, # Depth of the trees (betwwen 5 and 10, higher -> more overfitting)
          l2_leaf_reg=10, # L2 regularization (between 3 and 20, higher -> less overfitting)
          rsm=0.1, # % of features to consider in each split (lower -> faster and reduces overfitting)
          bootstrap_type='Bernoulli', # 'Bayesian' is de default, Bernoulli accelerates
          subsample=0.5, # Only with the last parameter in 'Bernoulli' or 'Poisson'
          max_ctr_complexity=2, # Max number of categorical features combinations (default=4)
          random_seed=RS,
          verbose=100) # shows train/test metric every 100 trees

print('\nCatboost Fit (Validation)...\n')
model_catboost_val.fit(X=pool_tr, eval_set=pool_val, early_stopping_rounds=esr)
```

Traceability in Black Box models: Shap Values

The Shap Values encode the influence on each Feature on each observation

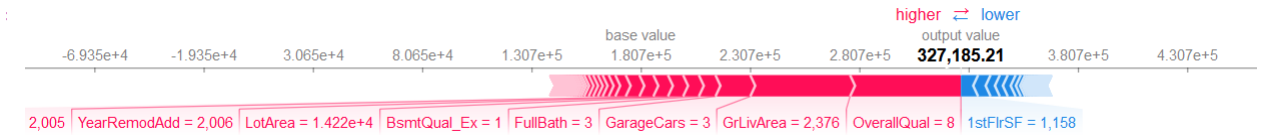


$$\phi_i(p) = \sum_{S \subseteq N/i} \frac{|S|!(n-|S|-1)!}{n!}(p(S \cup i) - p(S))$$

$$\text{Importance of i} = p(\text{with i}) - p(\text{without i})$$

Summary plot (global): orders the variables by importance (higher the more important), and shows the impact on the prediction (x-axis) based on the value of the variable (red high value, blue low value)

Force plot (local): shows the impact that the value of each variable has on the final prediction with respect to a base value of the prediction



Model Calibration

The need for Probability Calibration in classification models

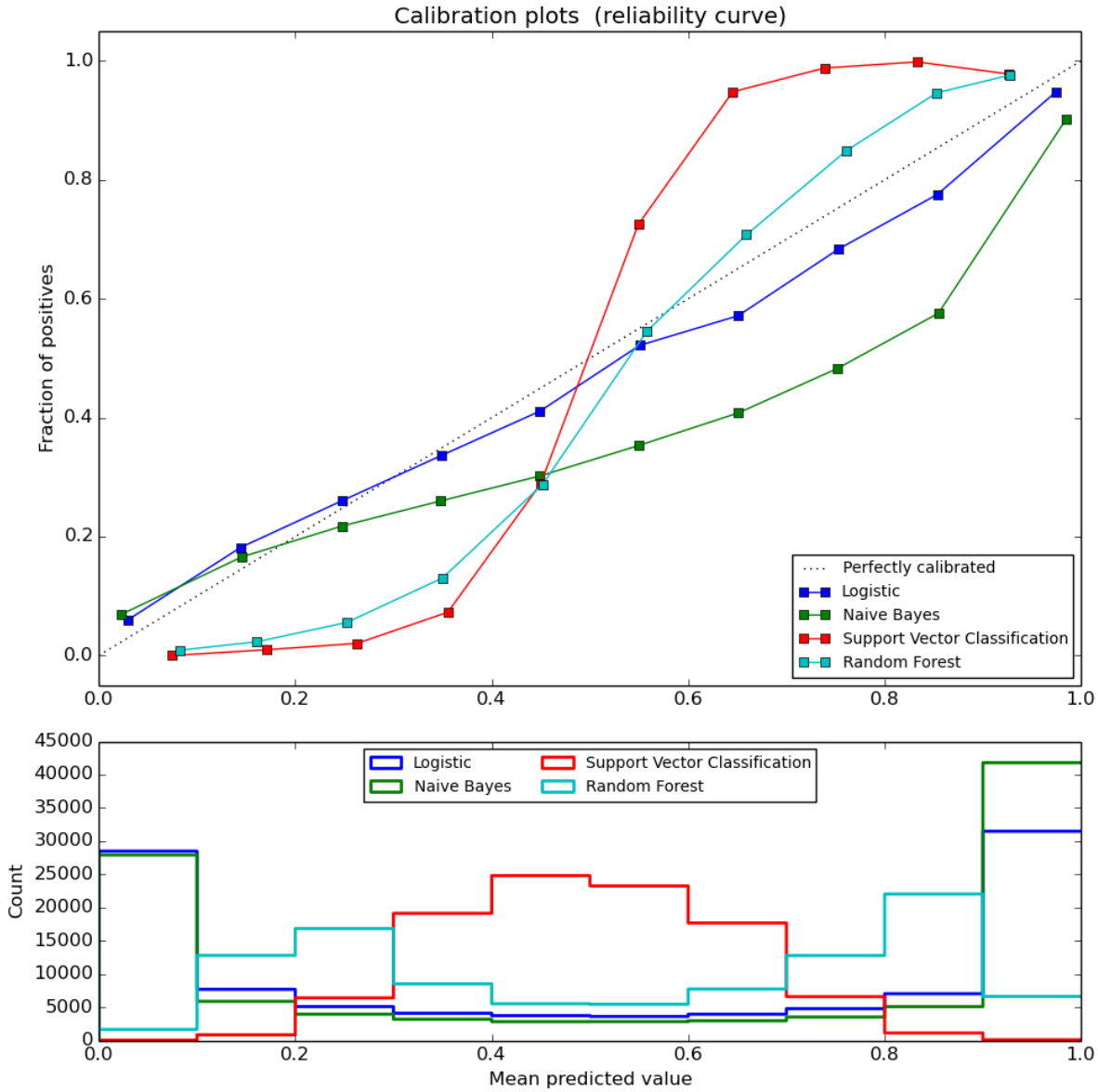
Model calibration

Mapping from a

Model output

to a

0-1 probability



The calibration of a model is essential to compare 2 different models trained on the 'same data'.

Examples of 'different':

- Different algorithmic origin
- Or trained with different historical deepness

The need for a calibration

The calibration of a model is simply the assignation of a probability (of class 1) to a ranking model output using an historical observed database.

Given a non-calibrated model we have 2 options:

1. If we only need a "threshold of acceptance" we can construct the **confusion matrix** on a validation set and find the optimal cut which maximizes our business objectives
2. If we need to quantify how much an observation (a client) is worse or better than another (for example, to offer different credit interest conditions) **we should assign probabilities via calibration**

When to calibrate if we care about probabilities?

- Non-Linear Model

Always (just in case...)

- To change Central Tendency
- Calibration sample ≠ Construction sample

The models can be re-estimated once in years but the calibration can be changed every year for instance

- One class sample correction

In the Construction database we have eliminated 0's o 1's (usually 0s if we have a low-default portfolio). We need to re-center the probabilities

The confusion matrix payoff approach

More conservative cut

Model Prediction (Given cutoff value C_1)	Reality: TARGET 1 (Default)	Reality: TARGET 0 (Good)	
Predicts 1 (Default)	950	400	1,350
Predicts 0 (Non-Default)	50	8,600	8,650
	1,000	9,000	9,550

1.96 M€

Payoff Matrix

	0	-300	*Opportunity and Reputational cost*
Loss Given Default (LGD)	-10,000	300	*Mean client margin*

Model Prediction (Given cutoff value C_2)	Reality: TARGET 1 (Default)	Reality: TARGET 0 (Good)	
Predicts 1 (Default)	900	50	950
Predicts 0 (Non-Default)	100	8,950	9,050
	1,000	9,000	9,850

1.67 M€

Given a Payoff Matrix, the optimal cutoff value can be calculated via a numerical maximization problem

Classical calibrations

Classic Approach:

Binomial Step

1. Take the calibration sample (with REAL 0/1) and use the model to make predictions
2. Bucketize the model predicted probabilities in equal sized quantiles
3. Calculate the REAL number of events and non-events in each bucket
4. Calculate the mean of the predicted probability in each bucket. Transform this value into a log odds
5. Fit a Binomial model (in each bucket events/total observations ~ predicted log odds)

Bayes step to change the Central Tendency (CT, a global theoretical 0/1 proportion)

- For each bucket (b), calculate the new %default (iterate until the Central Tendency is achieved):

$$p(\text{default}|\text{b} = i) = \frac{p(\text{b} = i|\text{default}) \cdot \text{CT}}{p(\text{b} = i|\text{default}) \cdot \text{CT} + p(\text{b} = i|\text{good}) \cdot (1 - \text{CT})}$$

- ...and go to step 5

Tasche: Quasi Moment Matching

The art of probability-of-default curve calibration

Dirk Tasche*

First version: December 15, 2012
This version: November 26, 2013

6v6 [q-fin.RM] 26 Nov 2013

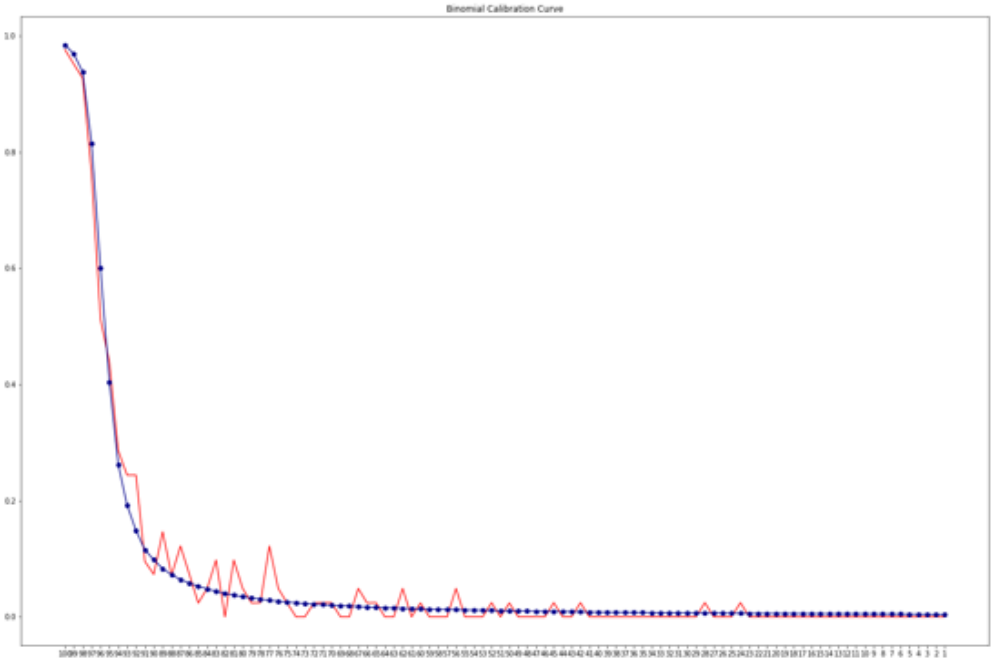
PD curve calibration refers to the transformation of a set of rating grade level probabilities of default (PDs) to another average PD level that is determined by a change of the underlying portfolio-wide PD. This paper presents a framework that allows to explore a variety of calibration approaches and the conditions under which they are fit for purpose. We test the approaches discussed by applying them to publicly available datasets of agency rating and default statistics that can be considered typical for the scope of application of the approaches. We show that the popular 'scaled PDs' approach is theoretically questionable and identify an alternative calibration approach ('scaled likelihood ratio') that is both theoretically sound and performs better on the test datasets.

KEYWORDS: Probability of default, calibration, likelihood ratio, Bayes' formula, rating profile, binary classification.

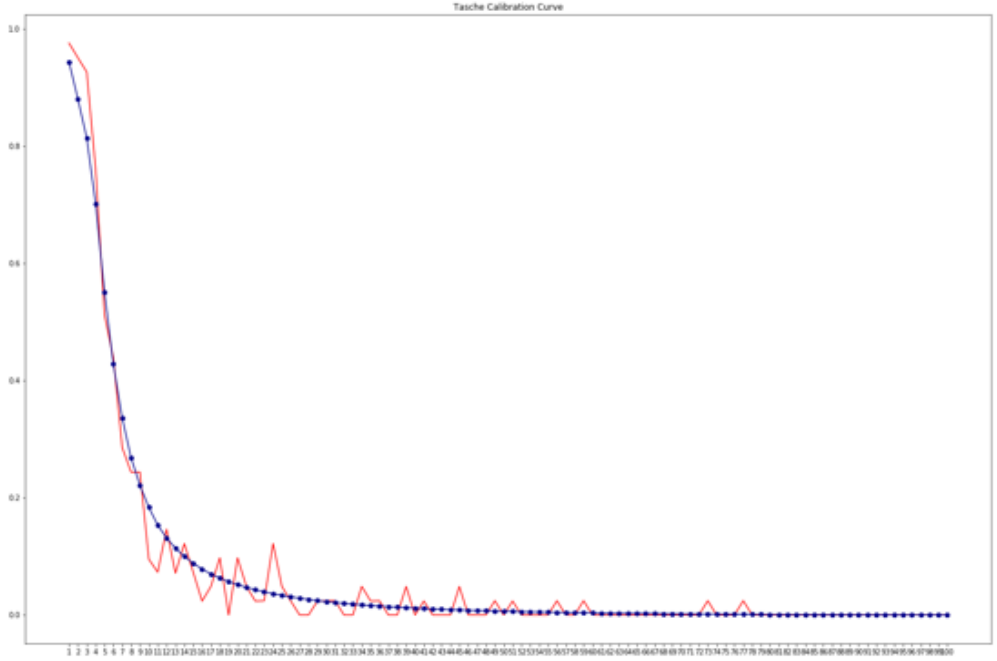
😩 Dr. Tasche has not yet accepted me in LinkedIn btw...

Tasche -vs- Classical approach

Classic

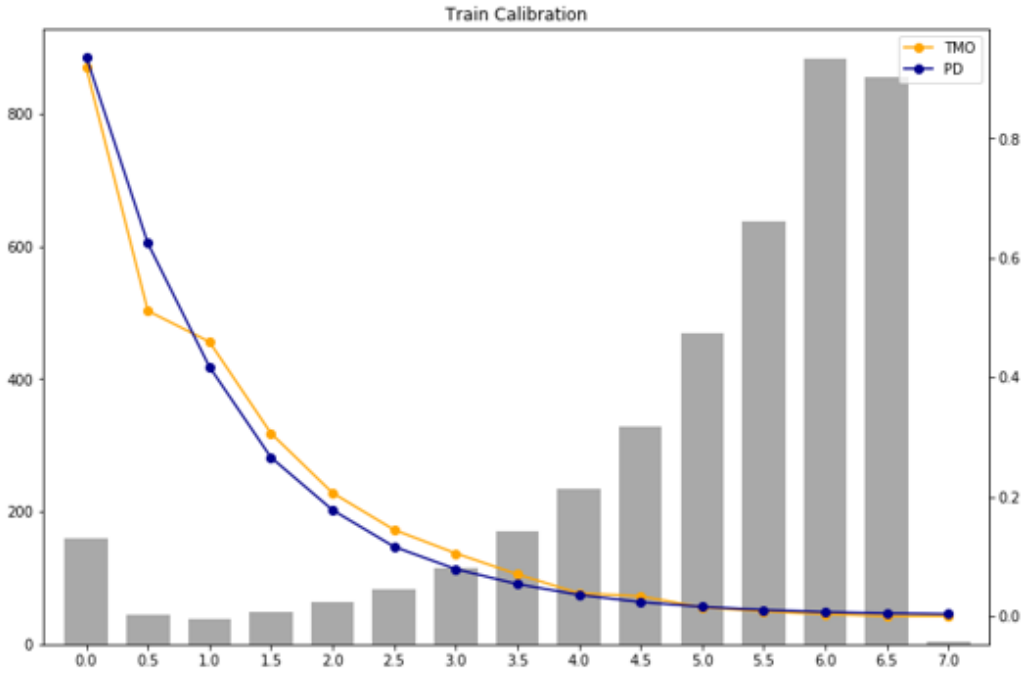


Tasche

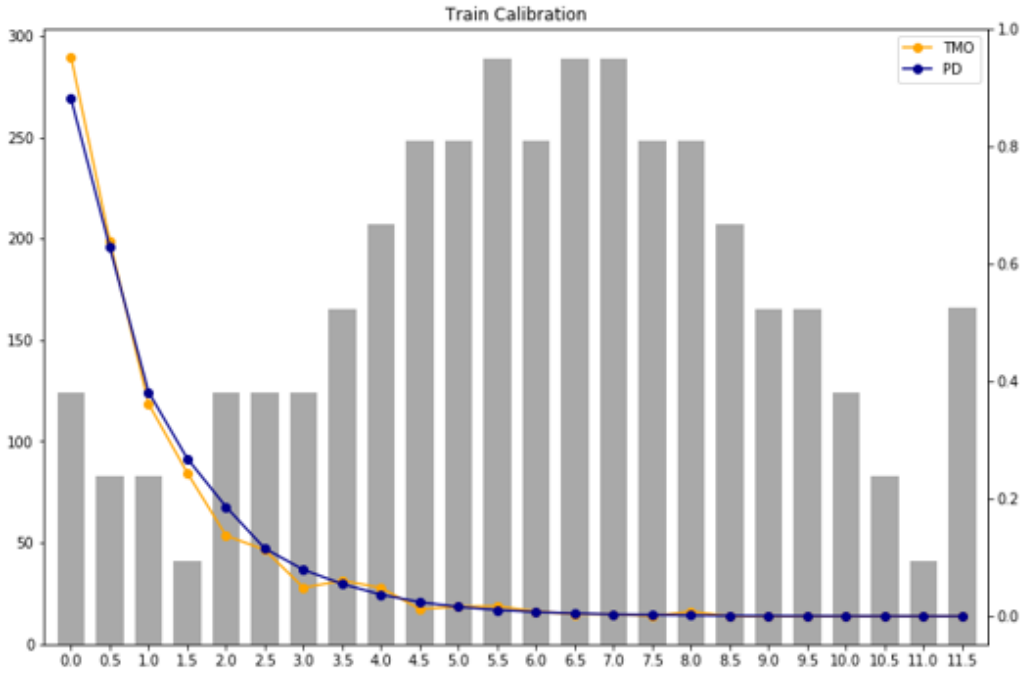


Tasche -vs- Classical approach

Classic



Tasche



Tasche theory implementation with Python

```python
import numpy as np
from statsmodels import distributions
from scipy.optimize import fsolve
from scipy import stats
from operator import sub
import math
import os
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot

class LDPortfolio:
    '''
    Basic functionality for all LDP calibration facilities
    :attribute self.ar: Estimated Accuracy ratio given por
    :attribute self.ct: Central Tendency (mean PD) ratio g
    '''
    def __init__(self, portfolio, pd_cnd=None):
        '''
        :param portfolio: Unconditional portfolio distributio
        ::::::Each 'portfolio' item contains total number of ob
        :param pd_cnd: Current conditional PD distribution fr
        ::::::Used for current AR estimation.
        '''
        self.portfolio = np.array(portfolio)
        self.pd_cnd = np.array(pd_cnd)
        self.portfolio_size = self.portfolio.sum()
        # self.portfolio_dist = self.portfolio.cumsum() / self.portfolio_size
        # self.portfolio_dist = (np.hstack((0, self.portfolio_dist[:-1])) + self.p
        self.rating_prob = self.portfolio / self.portfolio_si

        self.ct = None
        self.ar = None
        if not pd_cnd is None:
            self.ct, self.ar = self._ar_estimate(self.pd_cnd)

    def _ar_estimate(self, pd_cnd):
        ct = self.rating_prob.T.dot(pd_cnd);print(ct);print(s
        ar_lint_1 = self.rating_prob * pd_cnd
        ar_lint_1 = np.hstack((0, ar_lint_1[:-1]))
        ar_lint_1 = (1 - pd_cnd) * self.rating_prob * ar_lin
        ar_1 = 2 * ar_lint_1.sum()
        ar_2 = (1 - pd_cnd) * pd_cnd * self.rating_prob * se
        ar = (ar_1 + ar_2.sum()) * (1.0 / (ct * (1 - ct))) -
        return ct, ar.sum()
```

```python
class QMM(LDPortfolio):
    """
    Calibrates conditional probabilities of default according to Quasi Moment Matching algorithm
    :attribute self.pd_cnd: calibrated conditional PD
    :attribute self.alpha: intercept calibration parameter
    :attribute self.beta: slope calibration parameter
    """
    def __init__(self, portfolio, portfolio_cnd_no_dft = No
        """
        :param portfolio: Unconditional portfolio distribution
        ::::::Each 'portfolio' item contains total number of obs
        :param portfolio_cnd_no_dft: Conditional on no default
        ::::::portfolio distribution is used as a proxy)
        :return: initialized QMM class object
        """
        super().__init__(portfolio, pd_cnd=None)
        if portfolio_cnd_no_dft is None:
            self.portfolio_cnd_no_dft = self.portfolio
            self.cum_cond_ND = (np.hstack((0, self.portfolio_cnd_
        else:
            self.cum_cond_ND = (np.hstack((0, portfolio_cnd_no_df

        self.alpha = None
        self.beta = None

    def fit(self, ct_target, ar_target):
        """
        :param ct_target: target Central Tendency
        :param ar_target: target Accuracy Ratio
        :return: calibrated QMM class
        """
        a = self.__get_pd((0, 0))
        tf = lambda x: tuple(map(sub, self._ar_estimate(self._
        params = fsolve(tf, (0, 0))
        self.alpha, self.beta = params
        self.pd_cnd = self.__get_pd(params)
        self.ct, self.ar = self._ar_estimate(self.pd_cnd)
        return self

    def __get_pd(self, params):
        return self._robust_logit(self.cum_cond_ND, params)

    @staticmethod
    def _robust_logit(x, params):
        alpha, beta = params
        return 1 / (1 + np.exp(- alpha - beta * stats.norm.ppf
```

```python
class Tasche_train_test_calibration:
    ##########################################################################
    ##########################################################################
    ##########################################DISC / CONT METHODOLOGY#########
    ##########################################################################
    ##########################################################################
    def __init__(self, train_file_path, test_file_path, file_separator=","):
        self.train_file_path=train_file_path
        self.test_file_path =test_file_path
        self.file_separator =file_separator
        self.storagedir="./tasche/"
        if not os.path.exists(self.storagedir):
            os.makedirs(self.storagedir)
    def compute_tasche_calibration(self, results_filename, Normalization=False):
        import pandas as pd
        train = pd.read_csv(self.train_file_path,sep=self.file_separator)
        test  = pd.read_csv(self.test_file_path,sep=self.file_separator)
        # Parameters
        ##############
        n_buckets_discr=len(train['Pred'].unique()) # Number of buckets for Tasche Estimation (len(train) -> Continuous (acore
        n_buckets_show=50 # Number of buckets for either Discrete & Continuous Method Plots
        Central_Tendency_multiplier=1 # Central Tendency scaler from train portfolio default (1 is no shift)
        ##############
        print("+ Train #buckets:",n_buckets_discr, "| Train length:", len(train[train.keys()[0]]))
        print("+ Creating buckets of score and save it to apply to test set")
        bins=pd.qcut(train['Pred'].unique(), q=n_buckets_discr,labels=False,duplicates='drop',retbins=Tru
        train['bucket']=pd.cut(train['Pred'], bins=bins[1], labels=False, include_lowest=True)
        real_bins=len(bins[1])-1
        train['bucket']=real_bins-train['bucket']
        print("+ Building the calibration table")
        calib_data_N=pd.DataFrame(train[['bucket','TARGET']].groupby(['bucket'])['TARGET'].count()).reset
        calib_data_D=pd.DataFrame(train[['bucket','TARGET']].groupby(['bucket'])['TARGET'].sum()).reset_i
        calib_data=pd.merge(calib_data_N,calib_data_D,how='left',on='bucket')
        calib_data.columns=['bucket','N','TARGET']
        calib_data['D']=calib_data['TARGET']
        calib_data.loc[calib_data['N']==calib_data['D'],'D']=calib_data.loc[calib_data['N']==calib_data['
            Adjustment for 100% PD buckets
        calib_data['ND']=calib_data['N']-calib_data['D'] # Goods
        calib_data['COND_PD']=calib_data['D']/calib_data['N'] # Conditional PD
        total_ND=calib_data['ND'].sum() # Total GOODS
        calib_data['COND_ND']=calib_data['ND']/total_ND # GOODS distribution
        print("+ Defining Tasche model")
        pl=LDPortfolio(portfolio=calib_data['N'],pd_cnd=calib_data['COND_PD'])
        exit()
        ql=QMM(portfolio=calib_data['N'], portfolio_cnd_no_dft=calib_data['COND_ND'])
        # We choose if we change the central tendency via the multiplier (or an absolute number instead)
        ##############
        Central_Tendency=pl.ct*Central_Tendency_multiplier
        ##############
```

Model validation

Model Validation

Model validation consists of a set of techniques with the purpose to evaluate the performance of the model when it is already in "production". There are basically 3 families of tests:

Key word: BACKTEST
Test of the features & model properties in a different time horizons

- **Accuracy**

The discriminative power (AUC) is still high when compared to the construction values. Only the quality of the ordering matters

- **Stability**

  - **Of the features:** The recent or actual samples, maintain a similar distribution to the construction ones → PSI, Two sample K-S

  - **Of the model:** The scores of the model, have not experienced a significant shift → Transition Matrix

- **Calibration**

The model calibration is still good, the probabilities are well predicted. The precision and not only the ordering matters

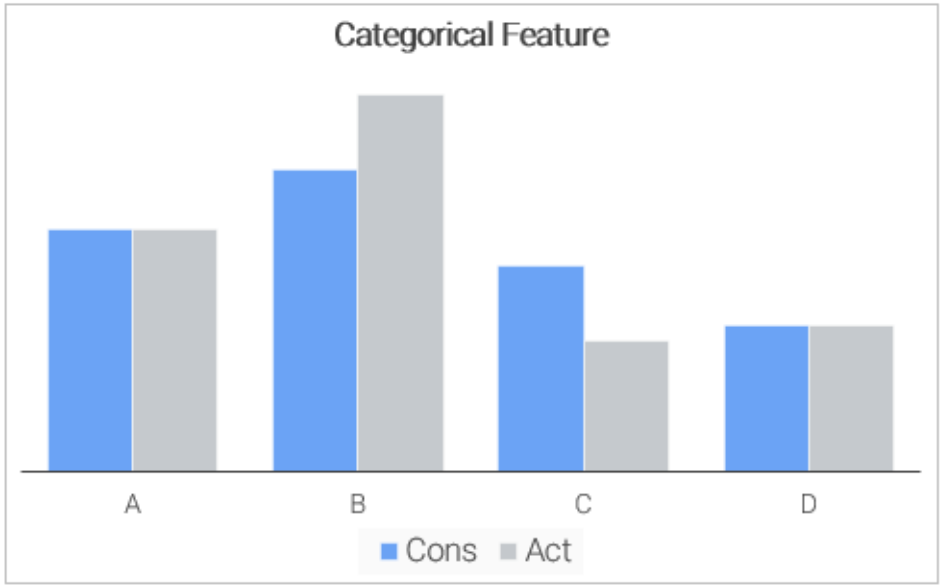
Feature stability

$$\text{PSI} = \sum_{i=1}^{\#\text{buckets}} \text{PSI}_i = \sum_{i=1}^{\#\text{buckets}} (\%\text{Actual}_i - \%\text{Construction}_i) \cdot \log\left(\frac{\%\text{Actual}_i}{\%\text{Construction}_i}\right)$$

	Cons	Act
A	40	45
B	50	70
C	34	24
D	24	27

	Cons	Act
A	27%	27%
B	34%	42%
C	23%	14%
D	16%	16%

	Act - Cons	Log(Act/Cons)	PSI bucket
A	0.08%	0.0030075	0.00%
B	8.38%	0.2216967	1.86%
C	-8.52%	-0.4630822	3.94%
D	0.05%	0.0030075	0.00%
		TOTAL PSI:	5.80%



A change on the distribution of the answers could imply a loss of the discriminant power of the Feature (or Module or Model)

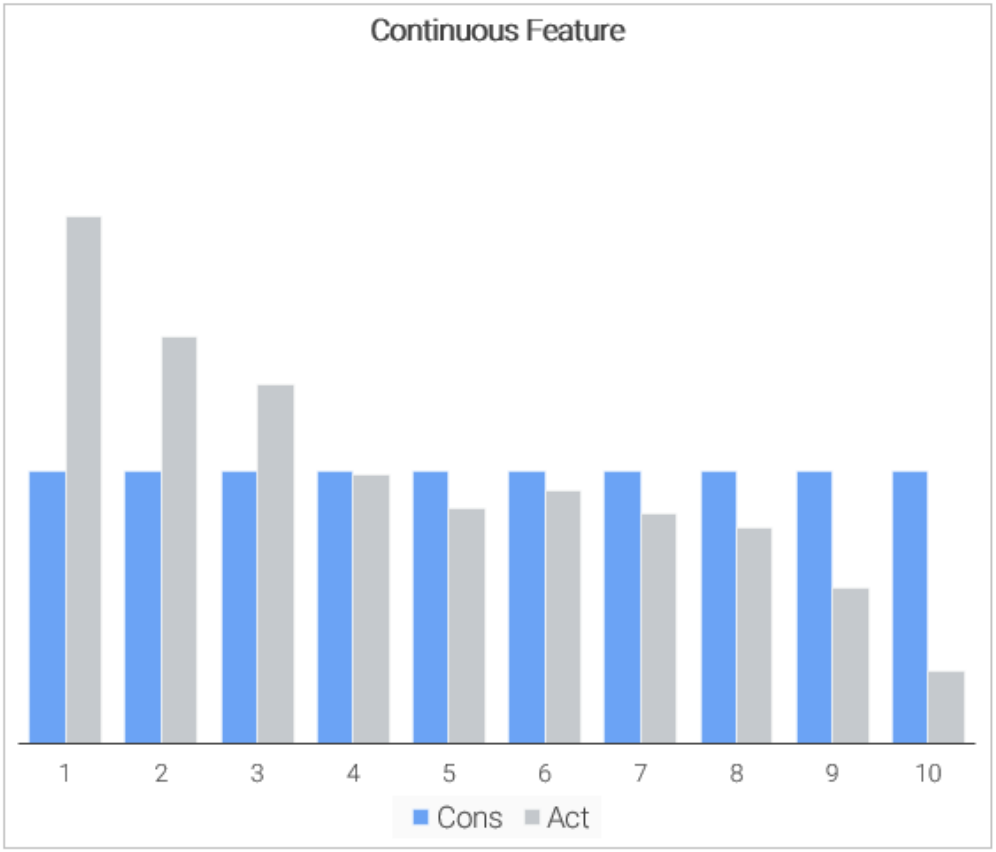
- PSI ≤10%
- 10% < PSI ≤ 25%
- 25% < PSI

Feature stability

	Cons	Act
1	100	220
2	100	170
3	100	150
4	100	112
5	100	98
6	100	106
7	100	96
8	100	90
9	100	65
10	100	30

	Cons	Act
1	10%	19%
2	10%	15%
3	10%	13%
4	10%	10%
5	10%	9%
6	10%	9%
7	10%	8%
8	10%	8%
9	10%	6%
10	10%	3%

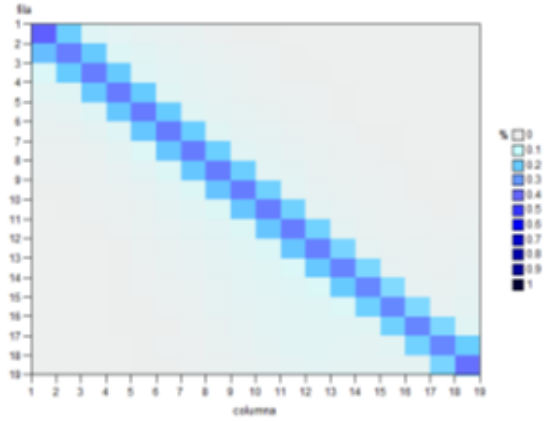
	Act - Cons	Log(Act/Cons)	PSI bucket
1	9.35%	0.6600641	6.17%
2	4.95%	0.4022350	1.99%
3	3.19%	0.2770719	0.88%
4	-0.15%	-0.0150645	0.00%
5	-1.38%	-0.1485959	0.21%
6	-0.68%	-0.0701243	0.05%
7	-1.56%	-0.1692152	0.26%
8	-2.08%	-0.2337537	0.49%
9	-4.28%	-0.5591761	2.40%
10	-7.36%	-1.3323660	9.81%
		TOTAL PSI:	**22.26%**



The transition Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	83,9%	4,3%	3,3%	2,7%	1,9%	1,2%	0,9%	0,7%	0,3%	0,4%	0,1%	0,1%	0,0%	0,1%	0,0%	0,0%	0,0%	0,0%	0,0%
2	13,9%	63,2%	7,0%	4,3%	2,9%	2,6%	2,4%	1,3%	0,7%	0,8%	0,2%	0,2%	0,2%	0,0%	0,1%	0,0%	0,0%	0,0%	0,0%
3	7,4%	6,7%	63,9%	6,2%	4,4%	3,3%	2,6%	1,9%	1,2%	0,7%	0,6%	0,3%	0,3%	0,1%	0,1%	0,0%	0,1%	0,1%	0,1%
4	4,7%	3,3%	6,8%	64,8%	5,7%	3,6%	3,0%	2,9%	1,8%	1,1%	0,7%	0,6%	0,4%	0,2%	0,1%	0,1%	0,1%	0,1%	0,1%
5	2,9%	2,0%	3,8%	7,2%	65,2%	6,0%	3,4%	2,4%	2,4%	1,4%	1,1%	0,8%	0,5%	0,3%	0,3%	0,1%	0,1%	0,1%	0,2%
6	1,7%	1,2%	2,3%	3,9%	7,4%	64,8%	5,6%	3,3%	2,5%	2,2%	1,7%	1,0%	1,0%	0,3%	0,3%	0,2%	0,1%	0,1%	0,2%
7	1,1%	1,0%	1,6%	2,0%	4,0%	7,9%	64,2%	5,8%	3,2%	2,4%	2,0%	1,5%	1,3%	0,6%	0,4%	0,3%	0,2%	0,2%	0,3%
8	0,8%	0,6%	1,1%	1,6%	2,6%	4,8%	8,4%	63,8%	4,8%	3,1%	2,5%	2,1%	1,2%	0,7%	0,6%	0,4%	0,3%	0,2%	0,4%
9	0,6%	0,3%	0,9%	1,1%	1,8%	2,8%	4,5%	8,0%	65,7%	4,1%	2,9%	2,0%	1,7%	1,0%	0,8%	0,5%	0,4%	0,3%	0,7%
10	0,3%	0,3%	0,7%	0,9%	1,3%	2,0%	3,3%	4,5%	8,4%	64,4%	4,6%	2,7%	1,9%	1,4%	1,2%	0,8%	0,5%	0,2%	0,7%
11	0,2%	0,2%	0,4%	0,9%	1,0%	1,9%	2,4%	3,3%	4,7%	7,7%	64,6%	4,1%	2,7%	1,8%	1,2%	0,8%	0,6%	0,4%	1,0%
12	0,1%	0,1%	0,2%	0,4%	0,8%	1,6%	1,7%	2,3%	3,5%	4,7%	7,8%	64,8%	3,7%	2,1%	1,4%	1,2%	1,1%	0,8%	1,6%
13	0,1%	0,1%	0,2%	0,3%	0,6%	1,0%	1,8%	2,4%	3,4%	3,7%	4,7%	7,7%	63,0%	3,5%	1,9%	1,2%	1,3%	0,8%	2,1%
14	0,1%	0,1%	0,1%	0,3%	0,5%	0,9%	1,2%	2,3%	2,5%	3,1%	4,0%	5,3%	7,0%	63,0%	2,8%	1,5%	1,3%	1,1%	2,9%
15	0,1%	0,0%	0,1%	0,1%	0,4%	0,6%	0,9%	1,4%	2,9%	3,3%	3,4%	3,8%	5,1%	6,2%	62,0%	2,6%	1,7%	1,3%	3,9%
16	0,0%	0,0%	0,1%	0,2%	0,5%	0,5%	0,8%	1,5%	2,5%	2,8%	3,3%	3,2%	3,8%	3,9%	4,5%	63,6%	2,7%	1,5%	4,5%
17	0,1%	0,1%	0,1%	0,2%	0,3%	0,5%	0,4%	1,0%	1,2%	3,0%	3,8%	3,6%	2,9%	2,8%	3,2%	4,7%	63,9%	2,8%	5,4%
18	0,1%	0,0%	0,1%	0,1%	0,4%	0,7%	0,8%	1,1%	1,6%	2,0%	2,6%	4,0%	3,3%	2,8%	2,8%	2,9%	3,9%	63,2%	7,6%
19	0,0%	0,0%	0,1%	0,0%	0,2%	0,2%	0,3%	0,4%	0,7%	1,4%	1,3%	2,2%	2,8%	2,2%	2,3%	2,4%	2,4%	2,8%	78,3%

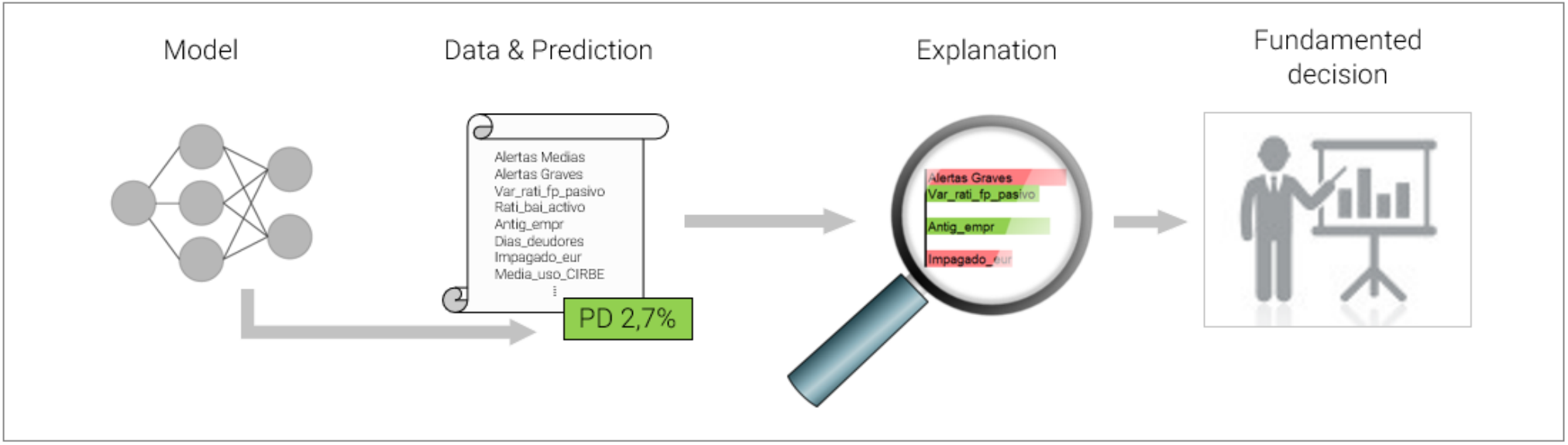
A transition matrix shows the probability of migrating from each rating grade (punctuation bucket) to any other in a given time horizon (1 year). The punctuation buckets should be considered BEFORE calibration



The transition Matrix

One of the main challenges to be solved with the Machine Learning approach is the interpretability of the results of the models

Nowadays, one can develop methodologies that allow to obtain an **easy and direct interpretation of the main drivers** that have led to each specific decision, **analogous to the "weights" of the factors** that have always been used in linear regressions



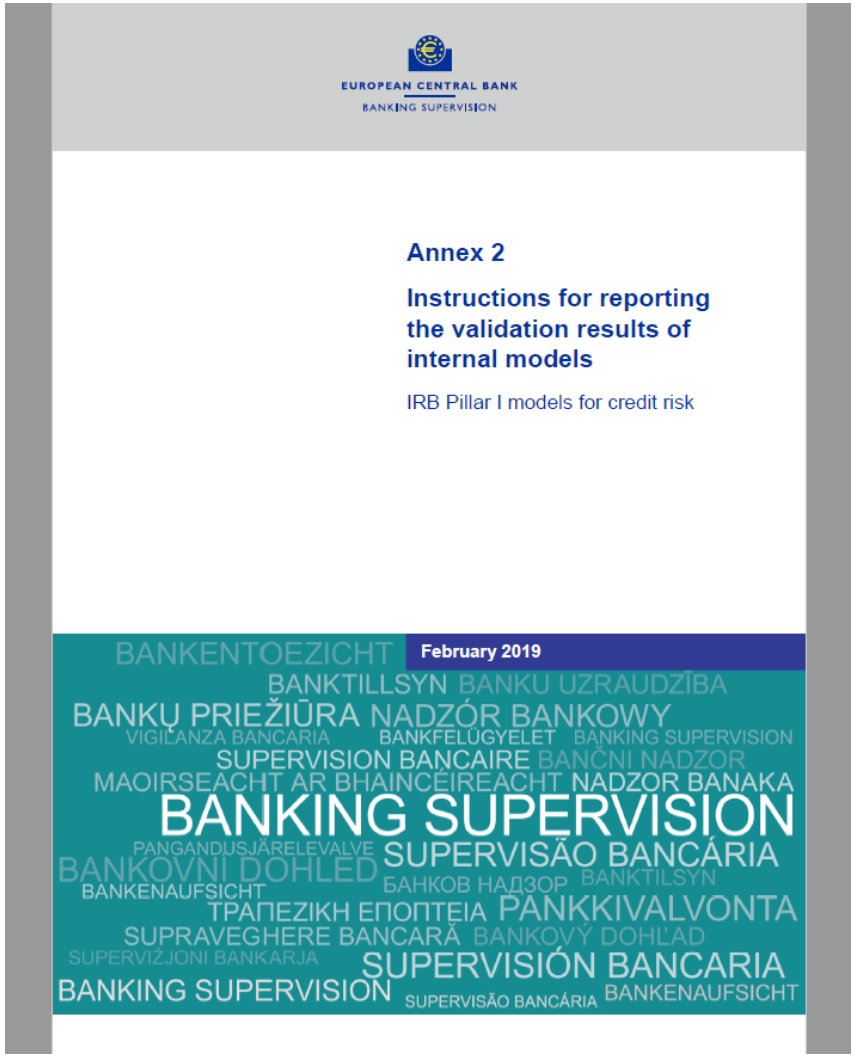
Allowing to:

- Choose between models competing against each other
- Detect and improve models that don't offer confidence
- Gain vision and interpretability of the model

and answer the key question

Why should we trust the model?
(beyond empirical evidences)

Real life example



EUROPEAN CENTRAL BANK
BANKING SUPERVISION

Annex 2

Instructions for reporting the validation results of internal models

IRB Pillar I models for credit risk

February 2019

BANKENTOEZICHT BANKTILLSYN BANKU UZRAUDZĪBA BANKŲ PRIEŽIŪRA NADZÓR BANKOWY VIGILANZA BANCARIA BANKFELÜGYELET BANKING SUPERVISION SUPERVISION BANCAIRE BANČNI NADZOR MAOIRSEACHT AR BHAINCÉIREACHT NADZOR BANAKA BANKING SUPERVISION PANGANDUSJÄRELEVALVE SUPERVISÃO BANCÁRIA BANKOVNI DOHLED БАНКОВ НАДЗОР BANKTILSYN BANKENAUFSICHT ΤΡΑΠΕΖΙΚΗ ΕΠΟΠΤΕΙΑ PANKKIVALVONTA SUPRAVEGHERE BANCARĂ BANKOVÝ DOHĽAD SUPERVIŽJONI BANKARJA SUPERVISIÓN BANCARIA BANKING SUPERVISION SUPERVISÃO BANCÁRIA BANKENAUFSICHT

Notice the wording used by the ECB and the one we are using in class, very similar isn't it?

Contents

1	**Background and rationale**	**2**
1.1	Introduction	2
1.2	Regulatory framework	3
1.3	The role of credit institutions' validation reporting in the ECB's ongoing monitoring of internal models	3
2	**Supplementary validation reporting on credit risk**	**5**
2.1	Scope of application	6
2.2	Reporting instructions	8
2.3	Definitions	9
2.4	General information	11
2.5	Probability of default	14
2.6	Loss given default	27
2.7	Expected loss best estimate	38
2.8	LGD in-default	42
2.9	Credit conversion factor	47
2.10	Slotting approach for specialised lending exposures	59
3	**Annexes**	**65**
3.1	Annex 1 – Statistics on the discriminatory power of PD models	65
3.2	Annex 2 – Statistics on the discriminatory power of LGD/CCF models	66
Abbreviations		**69**

Index of the ECB document

1) Analysis for models of Probability of Default (PD)	**p14**
1.1)Quantitative validation tools	**p17**
Rating process statistics	**p17**
1.2)Occurrence of overrides	**p19**
1.3)Occurrence of technical defaults	**p19**
1.4)Predictive ability	**p20**
PD back-testing using a Jeffreys test	**p20**
1.5) Discriminatory power	**p21**
Current AUC vs. AUC at initial validation/development	**p22**
Stability	**p23**
Customer migrations	**p23**
Stability of the migration matrix	**p24**
Concentration in rating grades	**p26**
2) Analysis for models of Loss Given Default (LGD)	**p27**
2.1) Predictive ability	**p29**
LGD back-testing using a t-test	**p30**
2.2) Discriminatory power	**p32**
Current gAUC vs. gAUC at initial validation/development	**p33**
2.3) Qualitative validation tools	**p34**
LGD assignment process statistics	**p35**
LGD application portfolio distribution	**p35**
LGD defaulted portfolio (recovery process not closed)	**p36**
LGD defaulted portfolio (recovery process closed within the observation period)	**p37**

$$EL = PD \cdot LGD \cdot EAD$$
$$= PD \cdot LGD \cdot [GC+(DC \cdot CCF)]$$

Probability of default

The aim of the validation tools below is to monitor the performance of PD models in the following areas of investigation:

(a) rating process;

(b) predictive ability (or calibration);

(c) discriminatory power (or rank-ordering performance);

(d) stability.

The validation tools are applied at rating grade[21] level, at portfolio level or both, as specified in the relevant sections below. The individual rating grades should be those used to calculate own funds requirements for non-defaulted exposures. Institutions which have more rating grades than the reporting templates allow or which use models with continuous PD should use the rating scale employed by the institution for validation and reporting purposes. In such cases, the PD derived from the model is mapped to that rating scale and the results for predictive ability and stability are reported on the basis of number-weighted average PD per grade. If a rating scale is used under these conditions, this is to be indicated in the relevant comment field for that area of investigation.

Loss given default

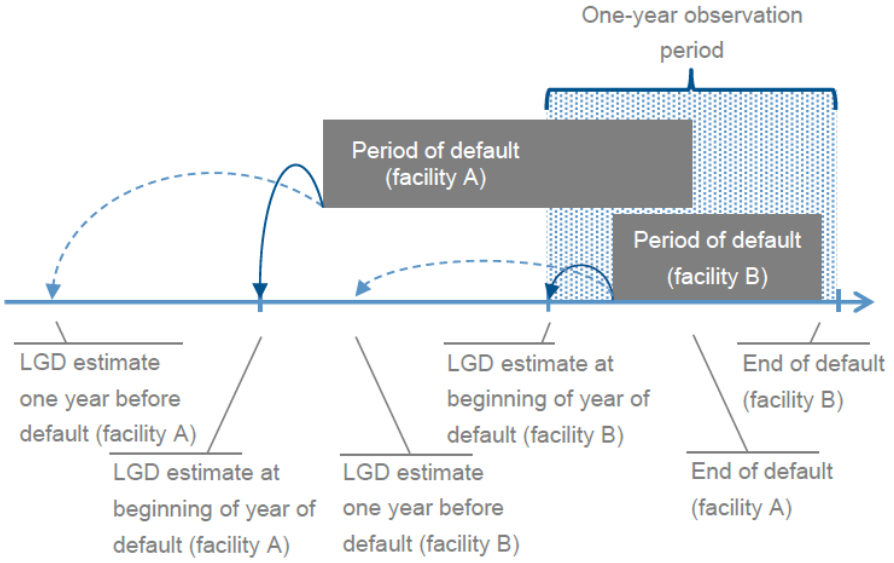
The aim of the validation tools outlined in this section is to monitor LGD models' performance in the following areas of investigation:

(a) predictive ability (or calibration);

(b) discriminatory power;

(c) qualitative validation tools.

Those validation tools are applied at facility grade or pool level, at portfolio level or both, as specified in the relevant sections below. Institutions which have more facility grades or pools than the reporting templates allow or which use models with continuous LGD should apply the validation tools on the basis of the 12 standardised segments defined by LGD estimates in this section.



Source: ECB Banking Supervision.

Implementation

Calculate the t-test statistic as follows:

$$T = \sqrt{N}\,\frac{\frac{1}{N}\sum_{i=1}^{N}\left(LGD_i^R - LGD_i^E\right)}{\sqrt{s_{LGD}^2}},$$

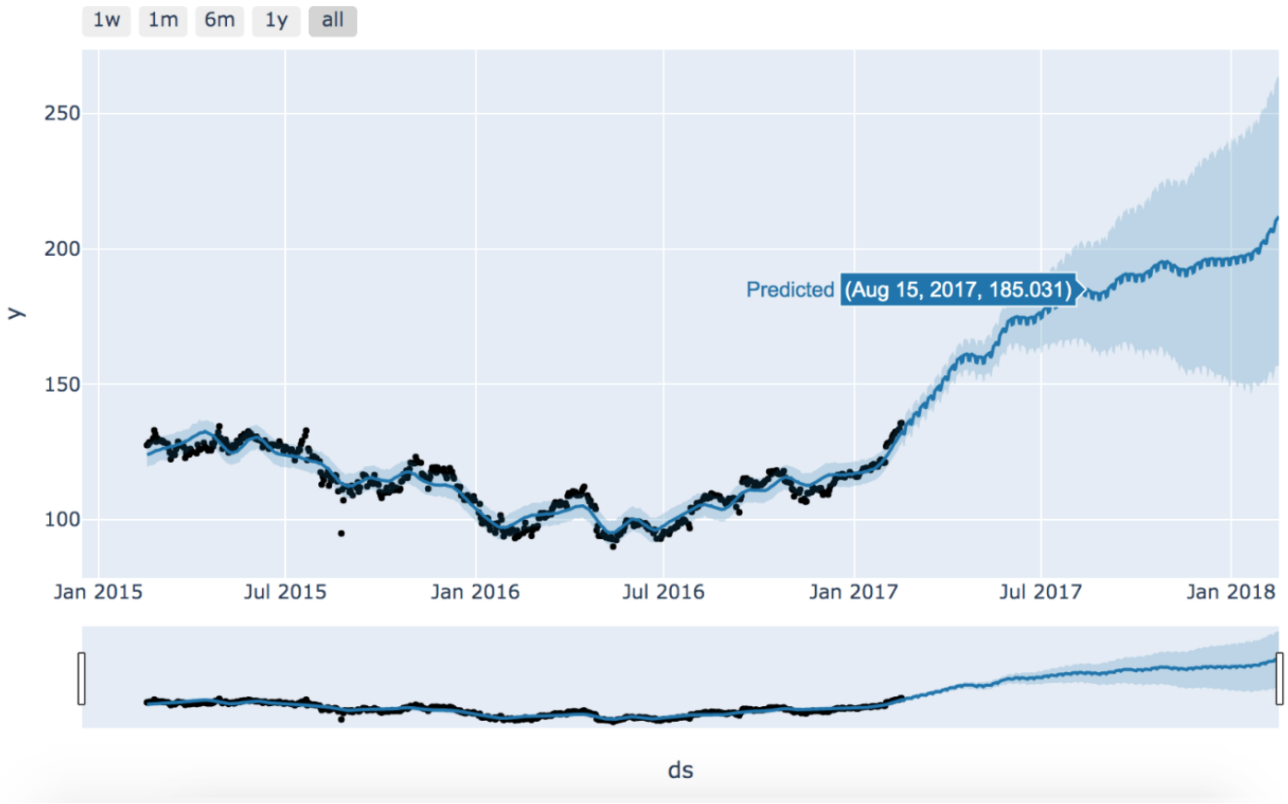
$$s_{LGD}^2 = \frac{\sum_{i=1}^{N}\left(\left(LGD_i^R - LGD_i^E\right) - \frac{1}{N}\sum_{j=1}^{N}\left(LGD_j^R - LGD_j^E\right)\right)^2}{N-1},$$

where:

- N is the number of facilities (back-testing) as defined in point (d) of Section 2.6.1;
- LGD_i^E denotes the estimated LGD for facility i as defined in point (a) of Section 2.6.1;
- LGD_i^R denotes the realised LGD for facility i as defined in point (b) of Section 2.6.1.

Calculate the p-value $1 - S_{N-1}(T)$, where S_{N-1} is the cumulative distribution function of the Student t-distribution evaluated using the test statistic (T) with $(N-1)$ degrees of freedom.

Time-series forecasting

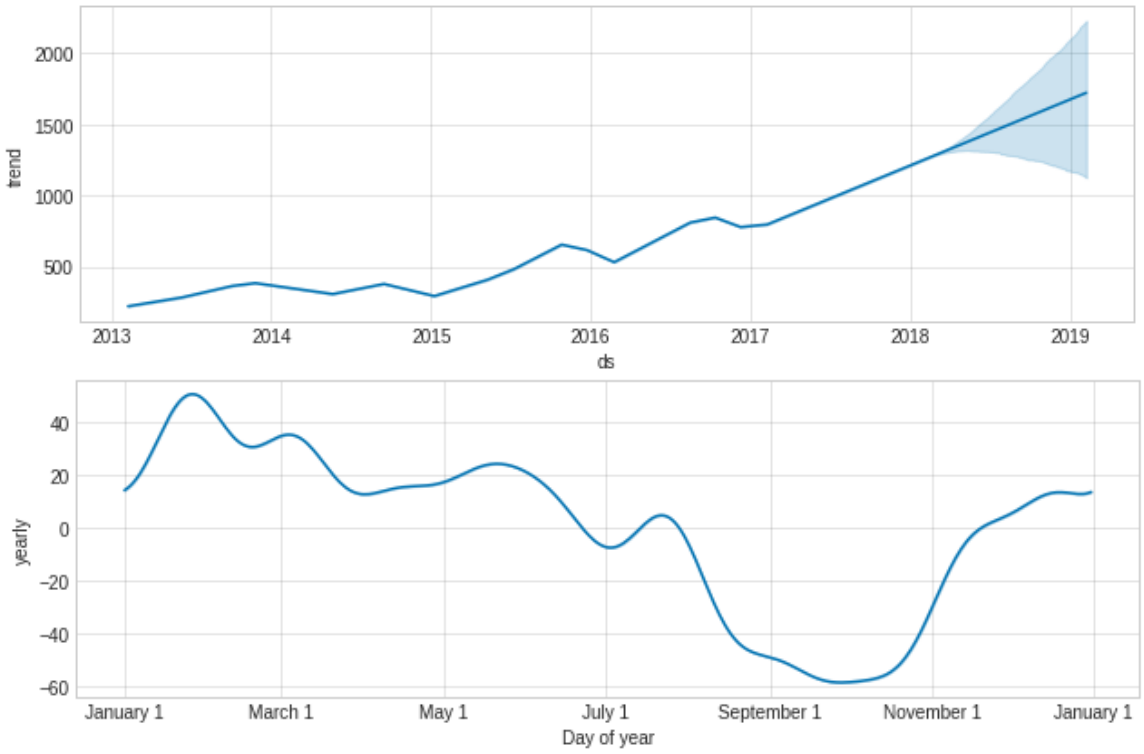


The 'classical approach'

The classical approach assumes that the data has an intrinsic time dependent pattern

$$y = \text{level} + \text{trend} + \text{seasonality} + \text{noise}$$

1. **Level**. The baseline value for the series if it were a straight line.
2. **Trend**. The optional and often linear increasing or decreasing behavior of the series over time.
3. **Seasonality**. The optional repeating patterns or cycles of behavior over time.
4. **Noise**. The optional variability in the observations that cannot be explained by the model.



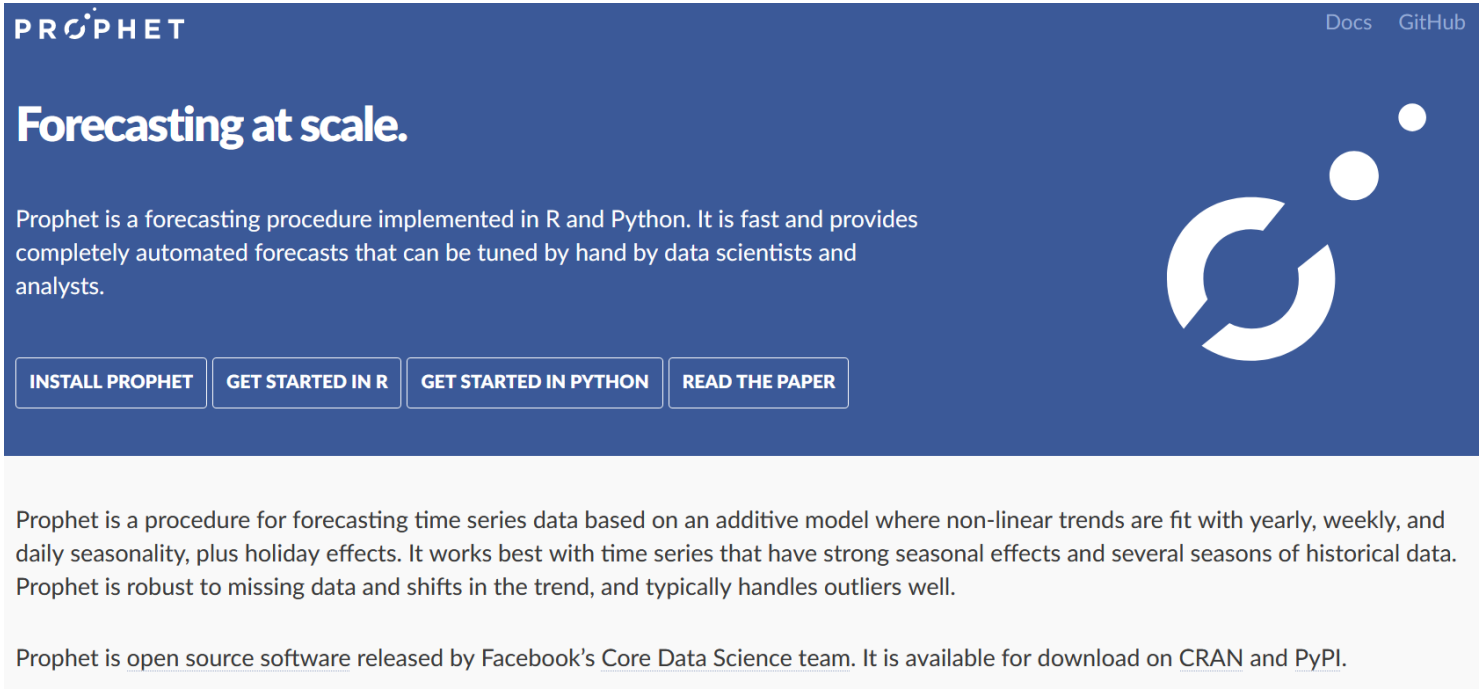
The 'classical approach' algorithms

ARIMA Auto Regressive Integrated Moving Average

- **AR**: *Autoregression*. A model that uses the dependent relationship between an observation and some number of lagged observations.
- **I**: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- **MA**: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations

Facebook's Prophet

from fbprophet import Prophet



Example: Predicting stock market with prophet (1/2)

```
import matplotlib.pyplot as plt
from fbprophet import Prophet


amzn_df = pd.read_csv('amazon_stocks.csv')
amzn_df.head()


# Drop the columns
ph_df = .drop(['open', 'high', 'low','volume', 'Ticks'], axis=1)
ph_df.rename(columns={'close': 'y', 'date': 'ds'}, inplace=True)
ph_df.head()

m = Prophet()
m.fit(ph_df)

# Create Future dates
future_prices = m.make_future_dataframe(periods=365)

# Predict Prices
forecast = m.predict(future_prices)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()

fig = m.plot(forecast)
ax1 = fig.add_subplot(111)
ax1.set_title("Amazon Stock Price Forecast", fontsize=16)
ax1.set_xlabel("Date", fontsize=12)
ax1.set_ylabel("Close Price", fontsize=12)

plt.show()



fig2 = m.plot_components(forecast)
plt.show()
```

The Dataset: Amazon stocks

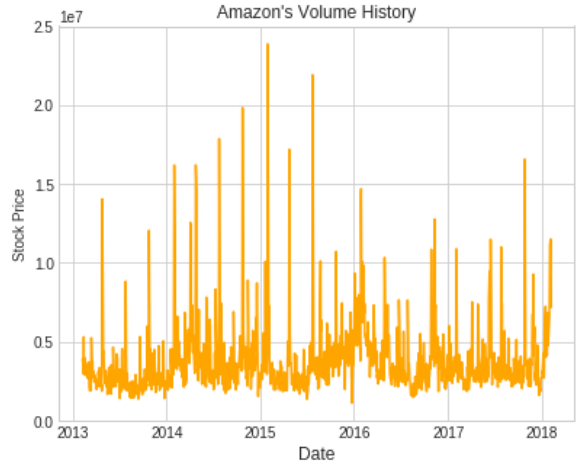
In [5]:

```
# Let's analyze some of the stocks.
amzn = df.loc[df['Ticks'] == 'AMZN']
amzn.head()
```

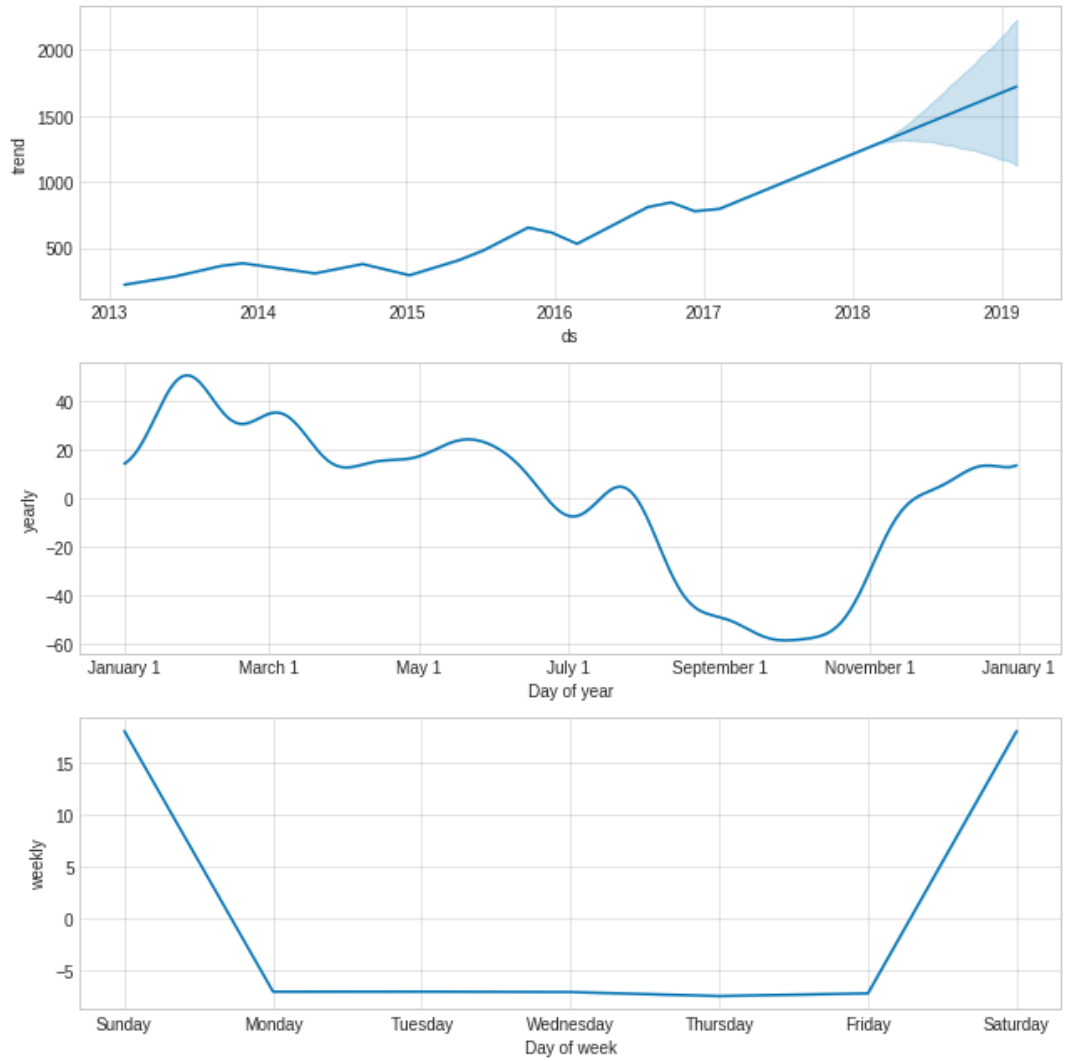
Out[5]:

	date	open	high	low	close	volume	Ticks
46387	2013-02-08	261.40	265.25	260.555	261.95	3879078	AMZN
46388	2013-02-11	263.20	263.25	256.600	257.21	3403403	AMZN
46389	2013-02-12	259.19	260.16	257.000	258.70	2938660	AMZN
46390	2013-02-13	261.53	269.96	260.300	269.47	5292996	AMZN
46391	2013-02-14	267.37	270.65	265.400	269.24	3462780	AMZN

The Dataset: Visualization



Example: Predicting stock market with prophet (2/2)



Note:
A common practice is to combine several algorithmic approaches in one

The 'modern' approach

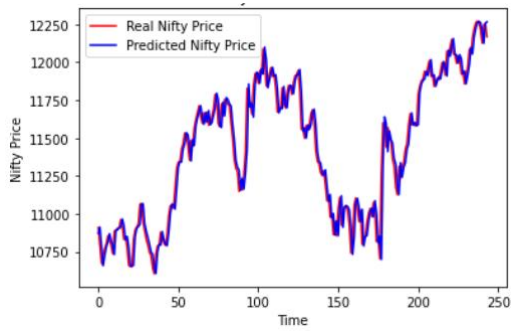
Use boosted decision trees or Artificial Neural Networks

A comparison of the different approaches using Nifty-50 index (India) (very noisy data)
Predicting 1 year time span

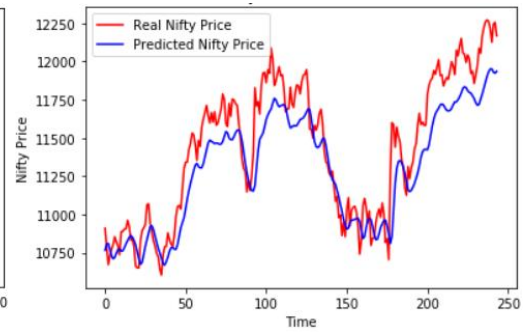
The data

Nifty50 chart

ARIMA



LSTM



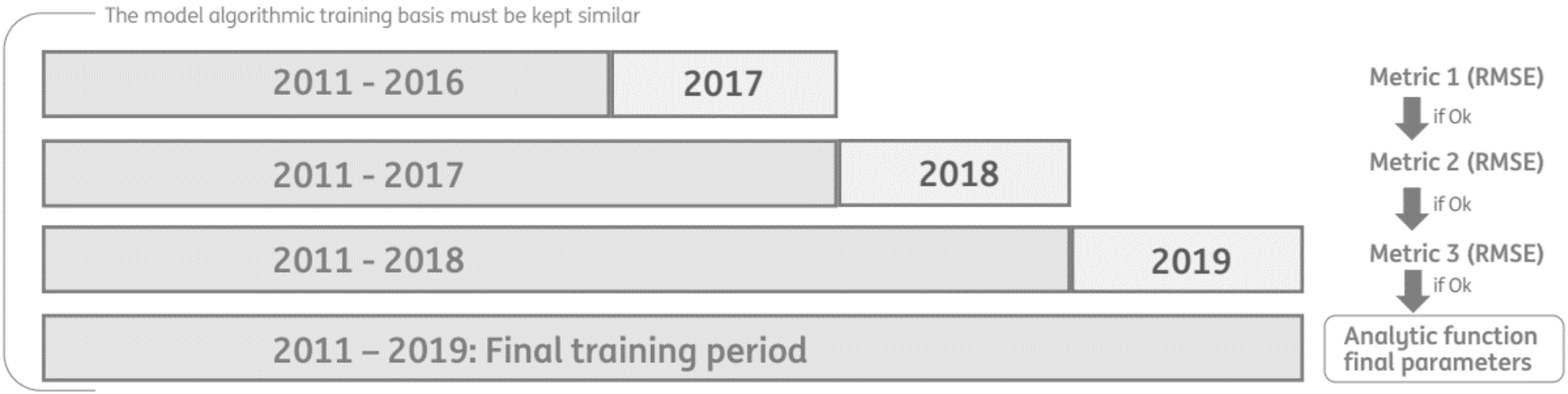
Prophet

How would the results have been testing with 2020 data?

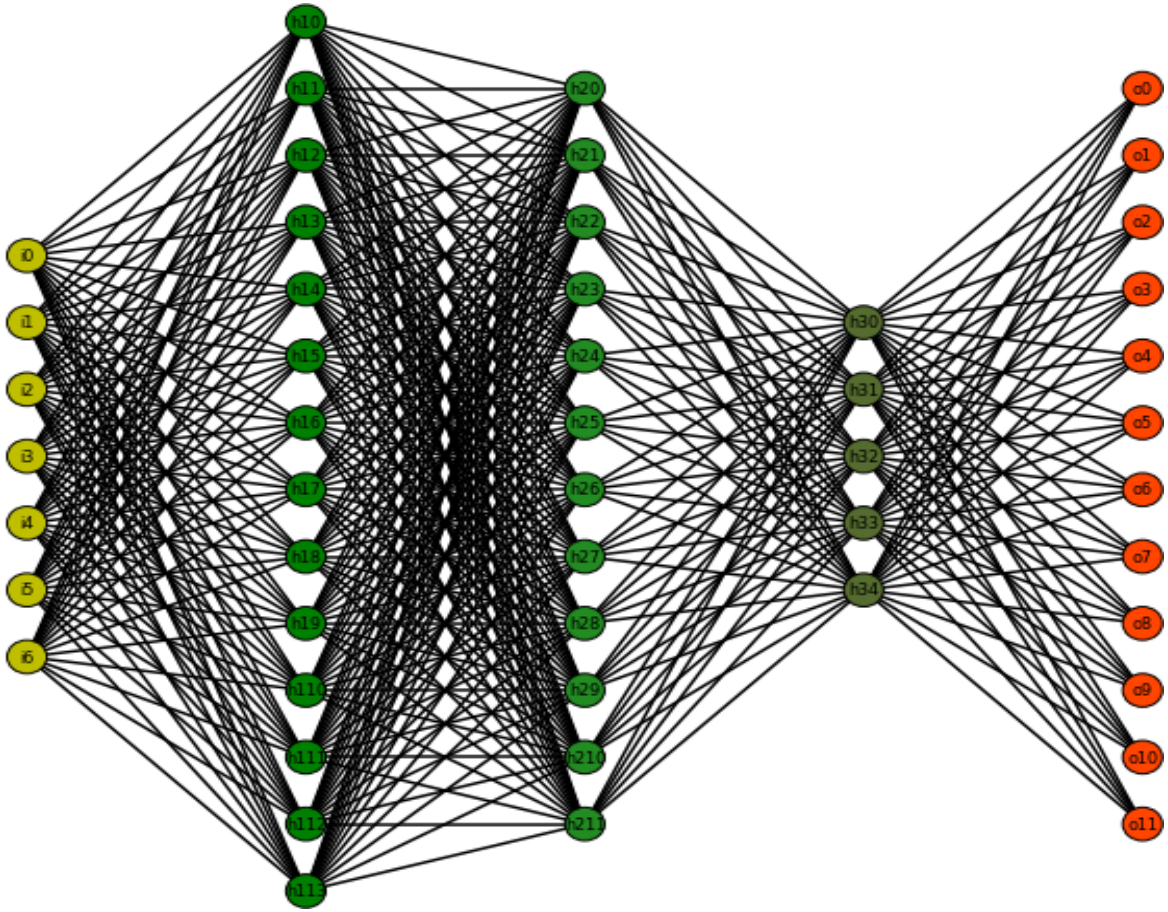
Metrics	ARIMA	LSTM	Facebook Prophet
MAE	73.24	175.74	504.00
MSE	9682.33	44148.30	409786.00

Training and Testing

The use of time windows is a must!!



Basics on Artificial Neural Networks and Deep Learning

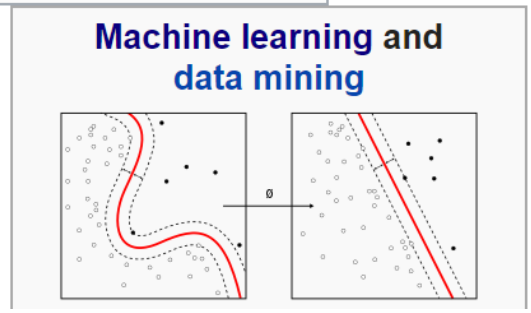


Artificial neural network

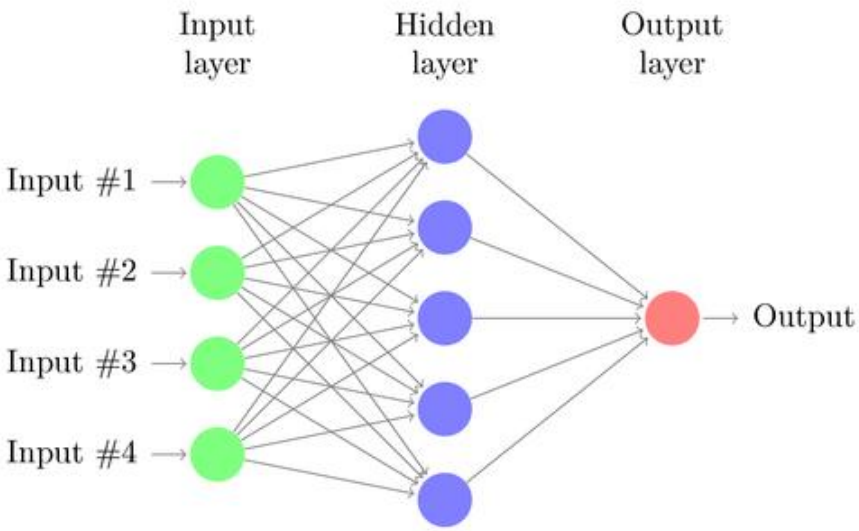
From Wikipedia, the free encyclopedia

It has been suggested that *Neural network* be merged into this article. (Discuss) *Proposed since May 2019.*

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks and astrocytes that constitute animal brains.[1][2][3] The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs.[4] Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like



Artificial neural networks have the ability to learn patterns of events, such as the **human brain**, and therefore solve impossible problems for other computing algorithms.



Basic characteristics of a Neural Network

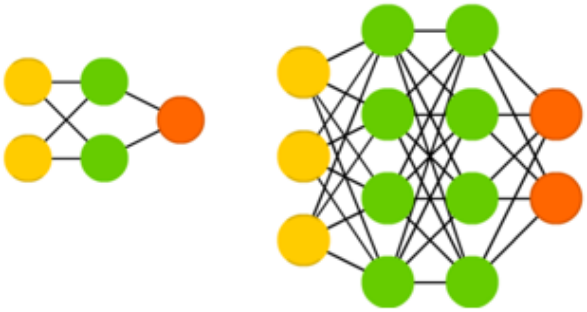
1) *Neurons (Cells)*

2) *Conexions (Synapsis)*

3) *Structure (Brain)*

Main types of Neural Networks

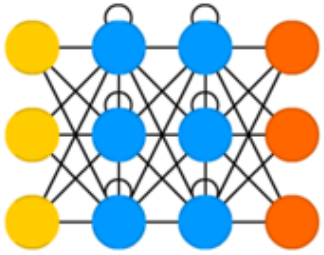
Feed Forward NN



(Learning the basics)

- The first kind of NN that were devised.
- Helpful for the typical exercises of classification and regression.
- All the neurons are connected with all the neurons of the adjacent layers (dense layers).

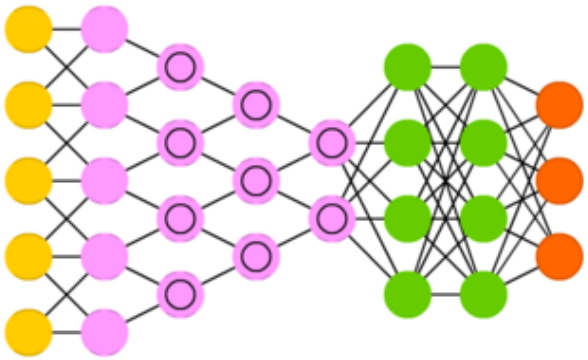
Recurrent NN



(Just to mention them)

- Very useful for working with time series and with problems that imply causality.
- They introduce neurons with memory.
- The results of the previous iterations affect the present.

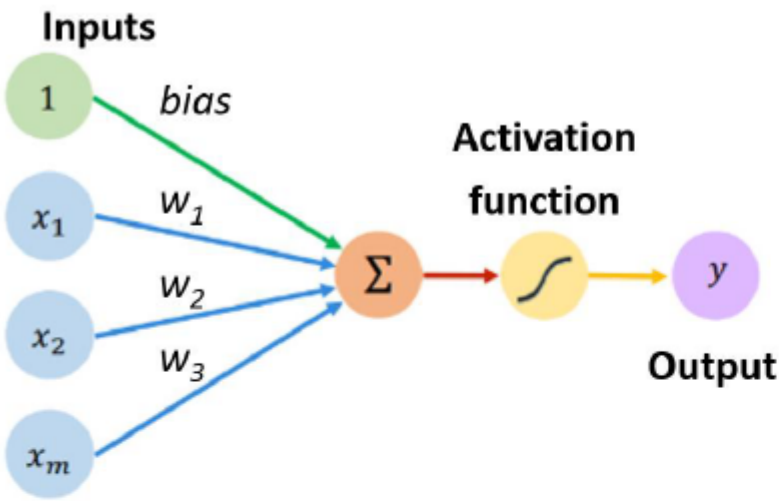
Convolutional NN



(Nowadays stars)

- Very useful for recognition of images, text and speech.
- They separate the input into subgroups that are processed first individually to later combine their information.

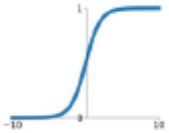
Basic structure of a standard neuron



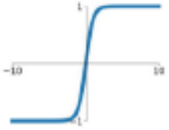
- Activation of the neuron (x_i): Its value indicates how much the corresponding neuron participates in the whole process
- Weights (w_i): Its value indicates the degree of connection (synapse for biologists) between the corresponding neurons
- Bias: Indicates the "sensitivity" of the neuron to be activated, the higher is the bias the more sensitive is the neuron
- Activation Function (f): Non-linear part of the algorithm that describes the behavior of the neuron when activated by some value.

There are different activation functions

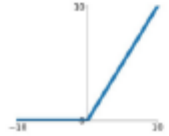
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



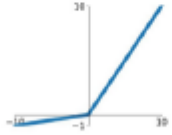
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

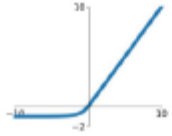


Leaky ReLU
$\max(0.1x, x)$



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

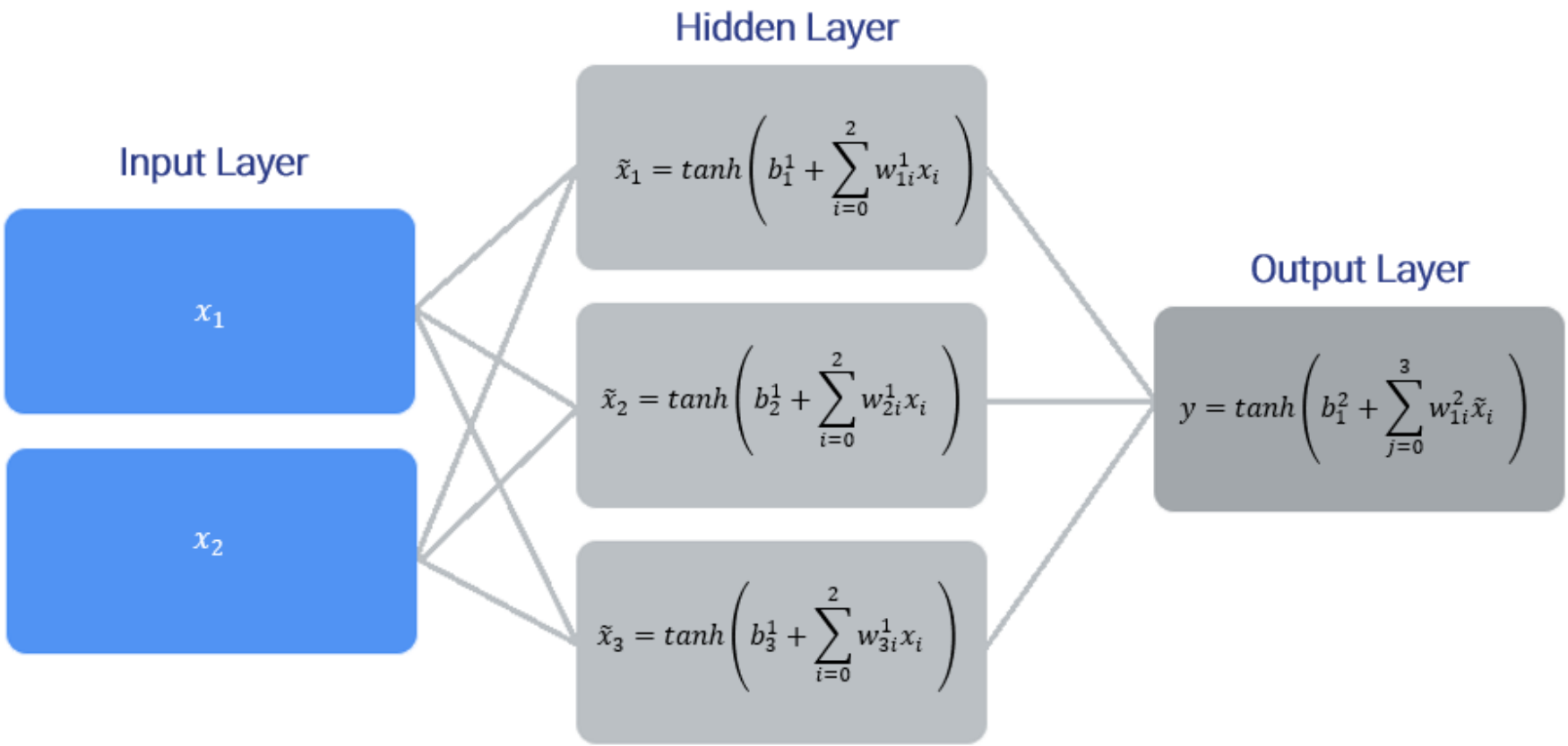
ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Output (y):

$$y = f\left(bias + \sum_{i=0}^{n} w_i x_i\right)$$

Basic behavior of a NN understood as a set of neurons



The Multi-Layer Perceptron

We already have the structure… but how does a NN learn?

As with any other ML algorithm, we need to train the Neural Network

- **Input**: representative data set with its corresponding result tags.
- The parameters of the network are adapted to obtain an output as similar as possible to the expected one.

The NN free parameters are the weights and the biases

- The main training algorithm of a NN is the **Gradient Descent (+ Backpropagation)**

The Gradient Descend algorithm

- A gradient measures how much the output varies when the input varies slightly.
- The aim is to improve the accuracy of the result by varying the parameters (weights and biases) in order to minimize the cost function (error of the output with respect to the expected result).

$$C = \frac{1}{N}\sum_{j}^{j=N}\left(y_{real}^{j} - y_{pred}^{j}\right)^2$$

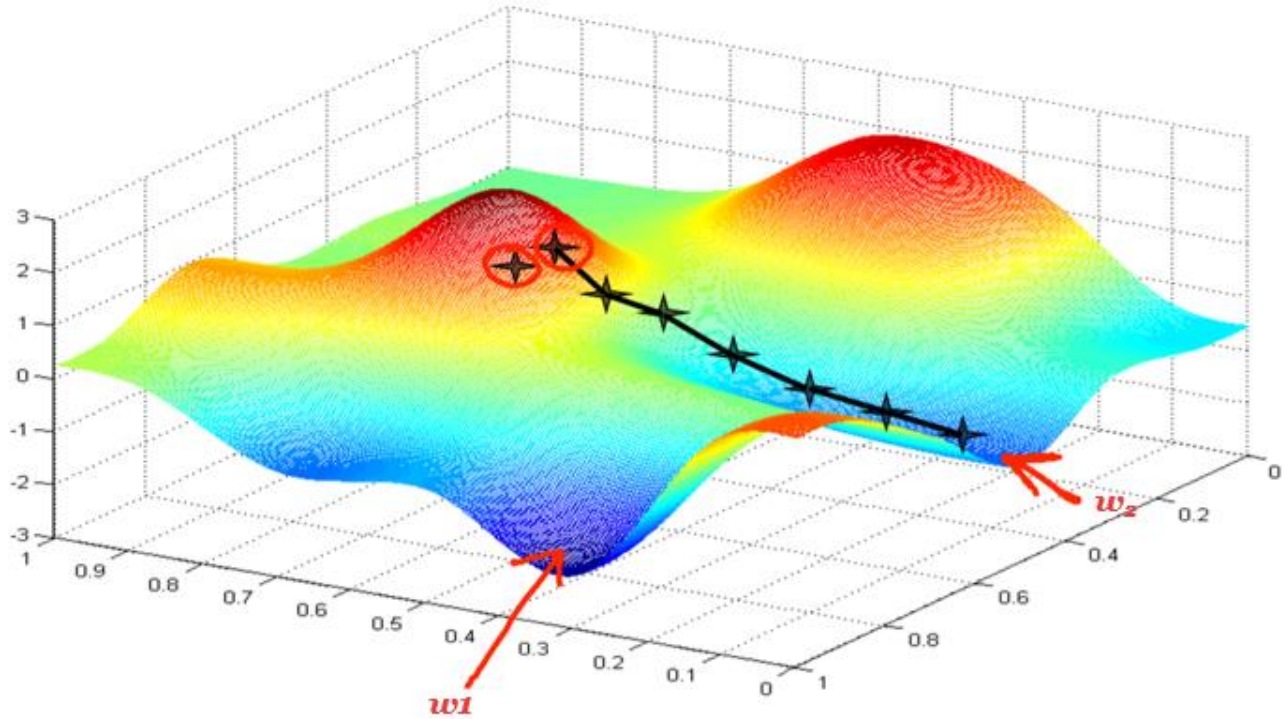
- The GD algorithm works in the following way:

1. Initialize weights an biases randomly.
2. Iterate until the maximum convergence is reached.

$$w_k \rightarrow w_k' - \frac{\eta}{N}\sum_i \frac{\partial C_{batch_i}}{\partial w_k}$$

$$b_k \rightarrow b_k' - \frac{\eta}{N}\sum_i \frac{\partial C_{batch_i}}{\partial b_k}$$

Learning rate (η): It is a hyperparameter that controls the length of the "step" that is used in the optimization of the Neural Network. It is a very sensitive parameter that can make the difference between a good optimization and a great overfitting.



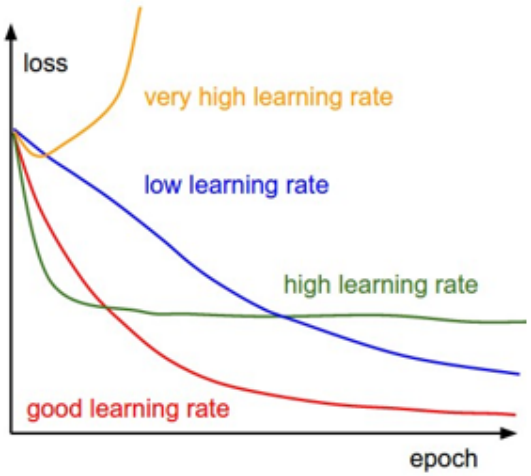
The Multi-Layer Perceptron

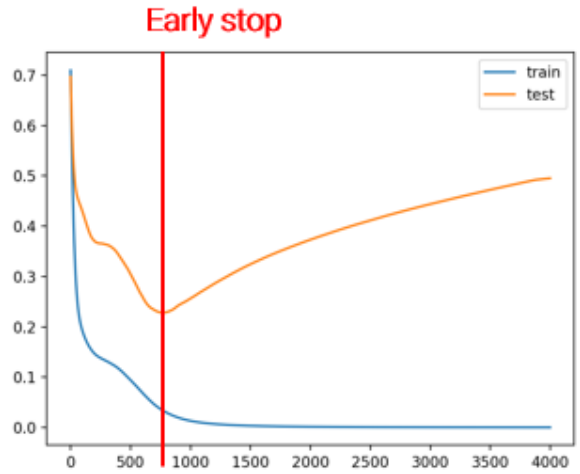
- **Batch size:** the batch size is a hyperparameter that defines the number of samples to be processed before updating the internal parameters of the model.
- **Number of epochs:** the number of epochs is a hyperparameter that defines the number of times that the learning algorithm will process the whole training data set.

Depending on the "batch size" and the "number of epochs" there are different types of gradient descent :

Batch GD	Stochastic GD	Mini-batch GD
Batch Size = Size of Training Set	Batch Size = 1	1 < Batch Size < Size of Training Set
Slow but safe and stable. Requires the entire dataset in memory.	Fast but computationally expensive, constant model update.	Good combination of the previous ones. (usually 50 <batch <250).

It may seem that the more epochs the better... But beware of overfitting!



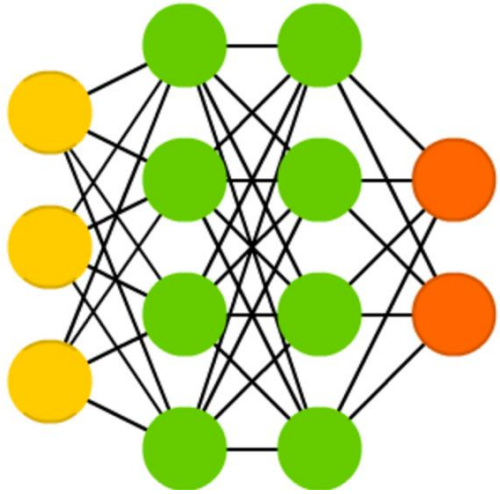


Typical steps to train a MLP

Data pre-processing (assuming already feature engineering)

Since neural networks are a highly nonlinear system, preprocessed data is an important step to avoid instabilities. It is advisable:

- Standardize (optional)
- Normalize



Design of the structure of the Neural Nerwork

- **Input Layer** : # of neurons: # of input features (+ activation function)
- **Hidden Layers: #** of layers, # of neurons (per layer): # of input features (+ activation function)
- **Output Layer** : # of output neurons (+activation funtions)

Hyper-parameter optimization

- Learning rate
- Batch number
- Epoch number

Correct validation with the TEST dataset

IMPORTANT: NNs tend to overtraining

The optimization of the topology, the hyperparameters, and the available computing quota must go hand in hand. Using k-folding techniques is highly recommended.

Convolutional Neural Networks (CNN)

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Kernel

1	0	1
0	1	0
1	0	1

Kernel is a feature detector of the input layer
Kernel is also called a **filter**

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

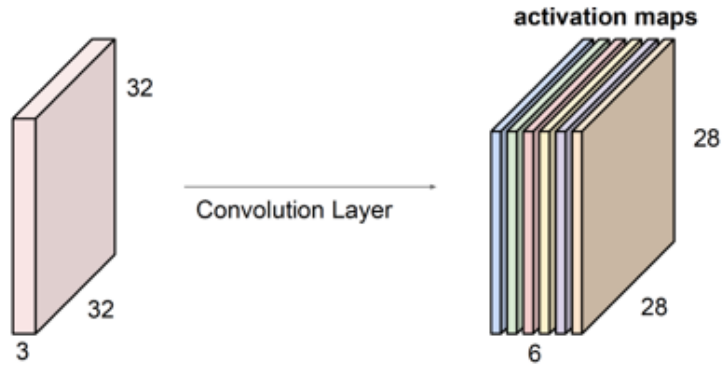
Image

4		

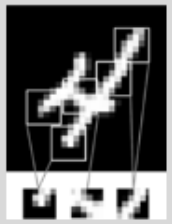
Convolved Feature

The convolved image is also called a **Feature Map**

- CNN (Convolutional Neural Network – CONVNET) is made of layers that preserve the spatial characteristics of an image.
- Dense Layers (MLP) learn global patterns, CNN learn local patterns.
- Highly efficient on solving perceptual problems, for recognition of images, text and speech.

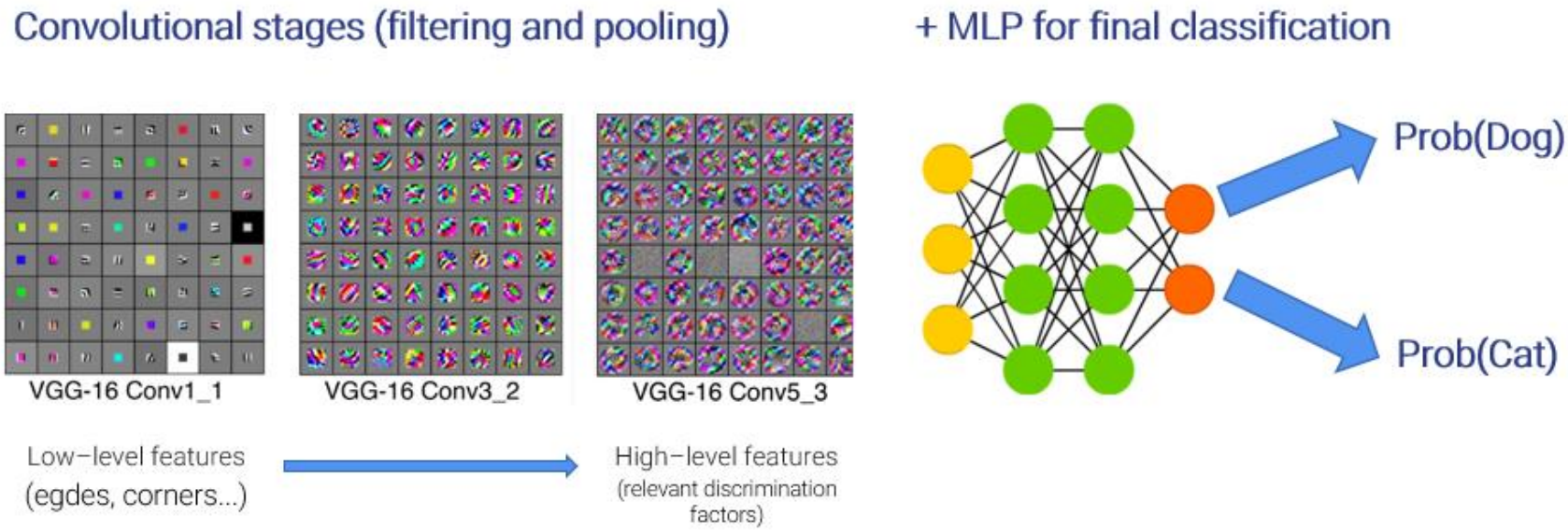


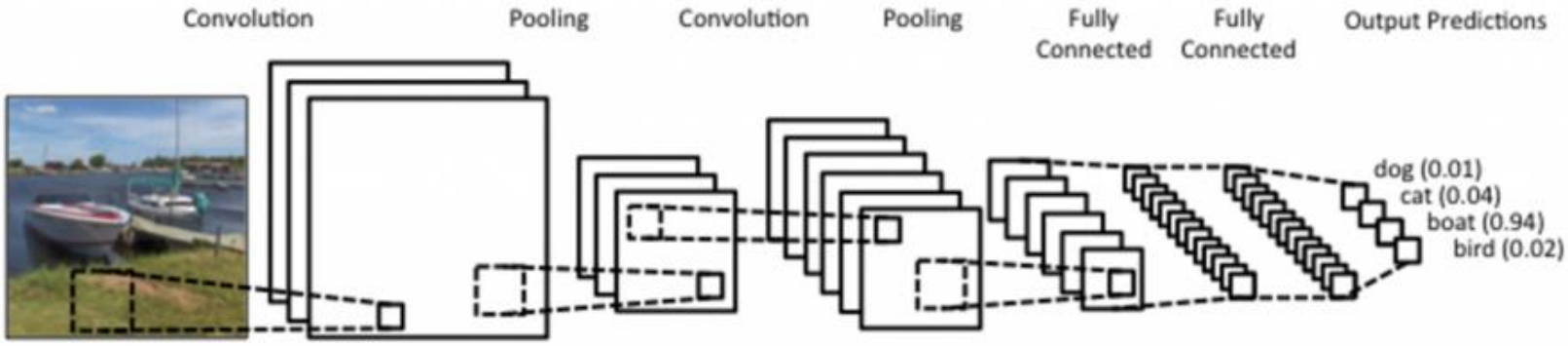
We stack these up to get a "new image" of size 28x28x6!



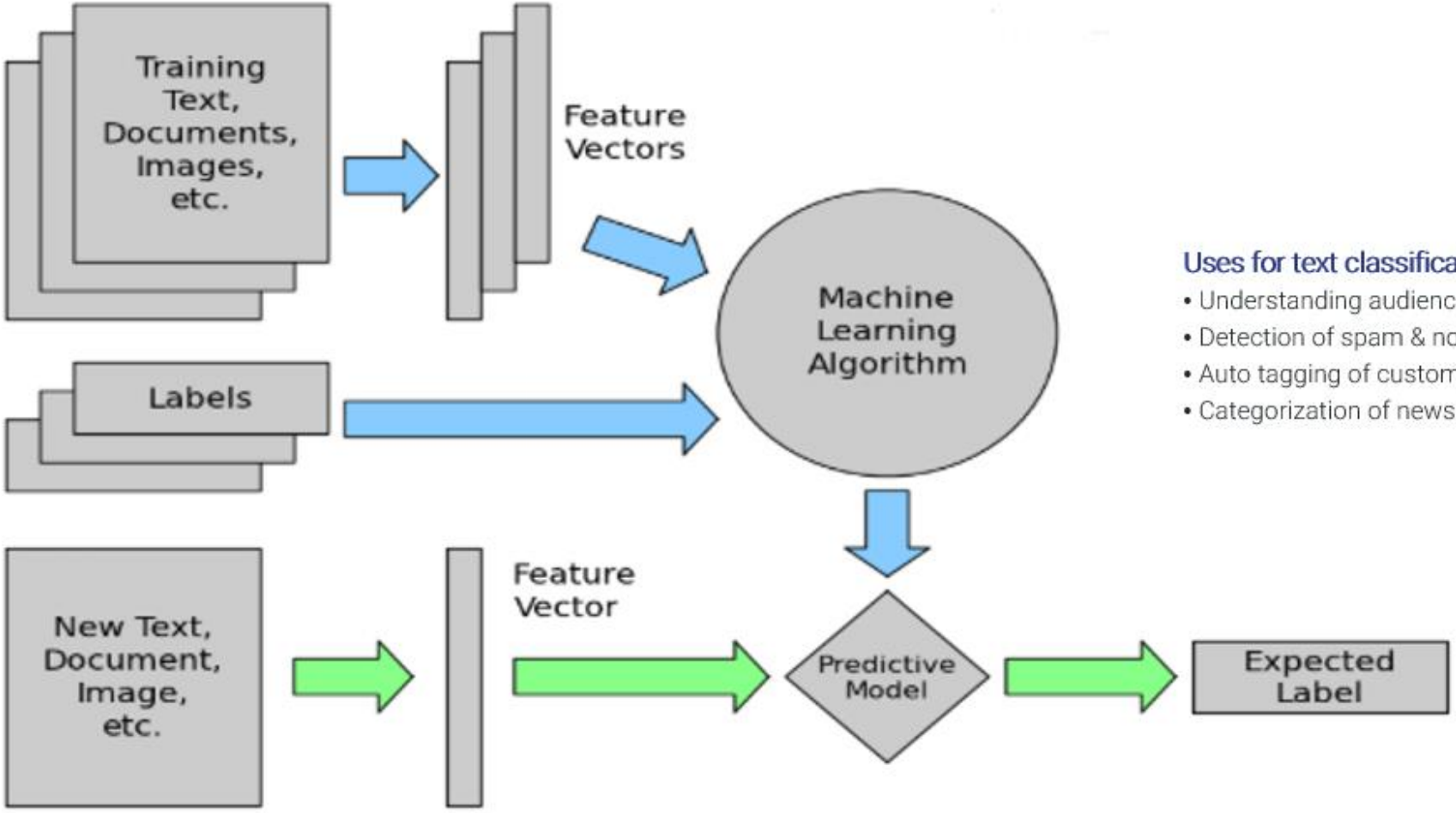
MNIST Neural Network Visualization: Recognition of handwritten digits example

Typical structure of a Neural Network





CNN applications: Text classification



Uses for text classification

- Understanding audience sentiment, e.g. from social media
- Detection of spam & non-spam emails
- Auto tagging of customer queries
- Categorization of news into predefined topics

Natural Language Processing (NLP)

- **Paragraph tokenization:** The process of breaking down a text paragraph into smaller chunks such as words or sentence is called Tokenization. Token is a single entity that is building blocks for sentence or paragraph.
- **Word tokenization:** Word tokenizer breaks text paragraph into words.
- **Frequency distribution of words.**
- **Stopwords:** Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc. Usually the stopwords are removed to stop biases.
- **Lexicon Normalization:** Lexicon normalization considers another type of noise in the text. For example, connection, connected, connecting word reduce to a common word "connect". It reduces derivationally related forms of a word to a common root word.
    - **Stemming:** Stemming is a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes. For example, connection, connected, connecting word reduce to a common word "connect".
    - **Lemmatization:** Lemmatization reduces words to their base word, which is linguistically correct lemmas. It transforms root word with the use of vocabulary and morphological analysis. Lemmatization is usually more sophisticated than stemming. Stemmer works on an individual word without knowledge of the context. For example, The word "better" has "good" as its lemma. This thing will miss by stemming because it requires a dictionary look-up.
- **POS Tagging:** The primary target of Part-of-Speech(POS) tagging is to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based on the context. POS Tagging looks for relationships within the sentence and assigns a corresponding tag to the word.

Building CNN text Classification Model with KERAS (Tensorflow) library

```
print('\nTokenizing and padding data... (vocab_size: %d)' % vocab_size)
tok = Tokenizer(num_words=vocab_size)
tok.fit_on_texts(full_data[text_col_proc])
sequences_train = tok.texts_to_sequences(train[text_col_proc])

print('Pad sequences... (max_len: %d)' % max_len)
train_input_f = sequence.pad_sequences(sequences_train, maxlen=max_len)
```

Text preprocessing

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

```
from keras.models import Sequential
from keras.layers import (Embedding, Dropout, Conv1D, GlobalMaxPooling1D, Dense, Activation, Flatten)
# from keras.callbacks import EarlyStopping
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score
```

Importing the libraries...

```
params = {
    'loss_func': 'binary_crossentropy', # binary_crossentropy
    'optimizer': 'adam', # adam, rmsprop
    'metrics': ['accuracy'],
    'embedding_size': 150,
    'batch_size': 128,
    'epochs': 5,
    'filters': 250, # 128
    'kernel_size': 3,
    'hidden_dims': 250,
    'pool_size': 2,
    'max_features': vocab_size,
    'max_len': max_len
}
```

Defining hyperparameters

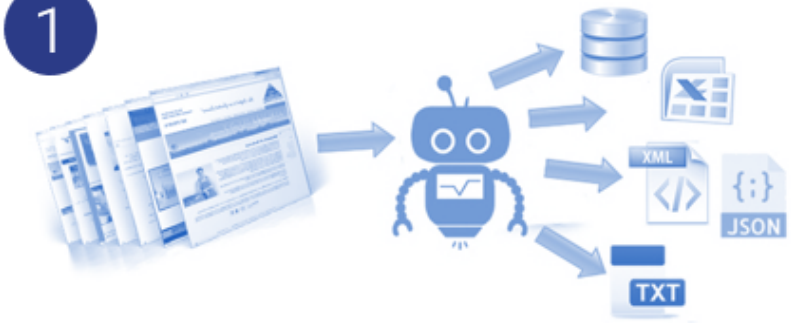
Building CNN text Classification Model with KERAS (Tensorflow) library

```
# Cross-validación
kfold = StratifiedKFold(n_splits=4, shuffle=False, random_state=12345)
cvscores = []

for tr_index, te_index in kfold.split(train_input_f, target):
    model_CNN = Sequential()
    model_CNN.add(Embedding(params['max_features'], params['embedding_size'], input_length=params['max_len']))
    model_CNN.add(Dropout(0.2))
    # we add a Convolution1D, which will learn filters
    # word group filters of size filter_length:
    model_CNN.add(Conv1D(filters=params['filters'], kernel_size=params['kernel_size'], padding='valid', activation='relu', strides=1))
    # we use max pooling:
    model_CNN.add(GlobalMaxPooling1D())
    # We add a vanilla hidden layer:
    model_CNN.add(Dense(params['hidden_dims']))
    model_CNN.add(Dropout(0.2))
    model_CNN.add(Activation('relu'))
    # We project onto a single unit output layer, and squash it with a sigmoid:
    model_CNN.add(Dense(1))
    model_CNN.add(Activation('sigmoid'))
    model_CNN.compile(loss=params['loss_func'],optimizer=params['optimizer'],metrics=params['metrics'])
    model_CNN.fit(train_input_f[tr_index],target[tr_index],batch_size=params['batch_size'],epochs=params['epochs'],verbose=0)
    # evaluate the model
    score=(2*roc_auc_score(target[te_index],model_CNN.predict(train_input_f[te_index])[:, 0])-1)*100
    print('Fold IP:',score)
    cvscores.append(score)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

CNN real life examples: Sentiment Analysis

Through a scraper all the news of **Expansión** are downloaded from 2001 to the present, to later model the 'Sentiment' of each news item

1

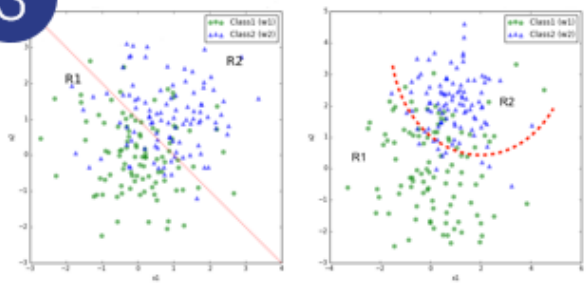


The scraper tracks the web of the newspaper library and reads and downloads all the news with its date, content, summary and sectoral and geographic indicators
→ (about 300,000 since 2008)

2

A series of unambiguous keywords are defined to mark the 'positive', 'neutral' and 'negative' news. It is possible to classify approx. 20% of the sample

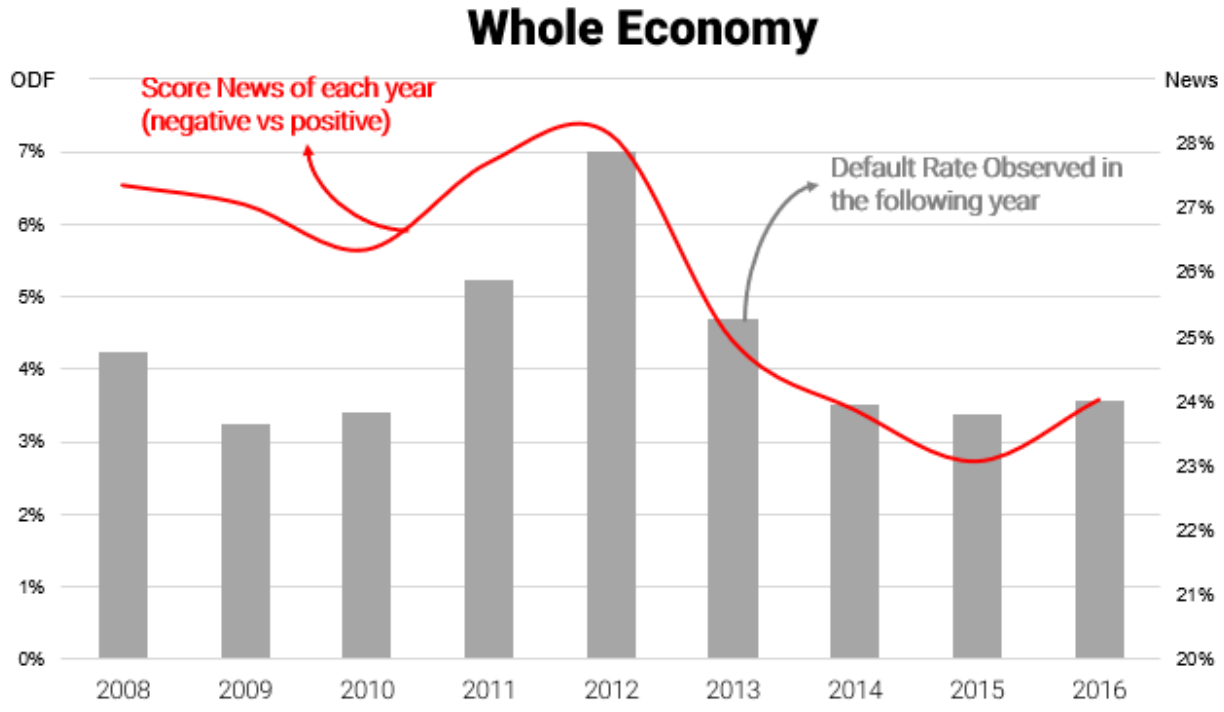
3



A Deep Learning model is trained on the labeled sample and the 'Sentiment' is predicted on the remaining set of news. The result is a complete database with the perception at all times.

CNN real life examples: Sentiment Analysis

Historical data of Observed Default Rate have been obtained for the portfolio of Companies of a banking entity (both the total and segmented by sector of activity)



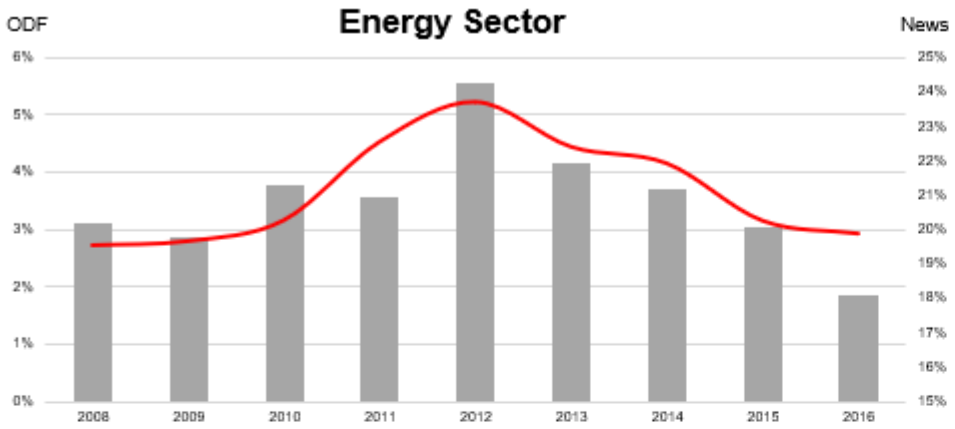
The columns are the default rate at 12m of the companies to which a rating has been made during the year in question (most during the 4th quarter) and therefore are a proxy for the default of the following year. The red line is the "score" of negative news against positive news (higher, worse news). The news are those observed during the year in question

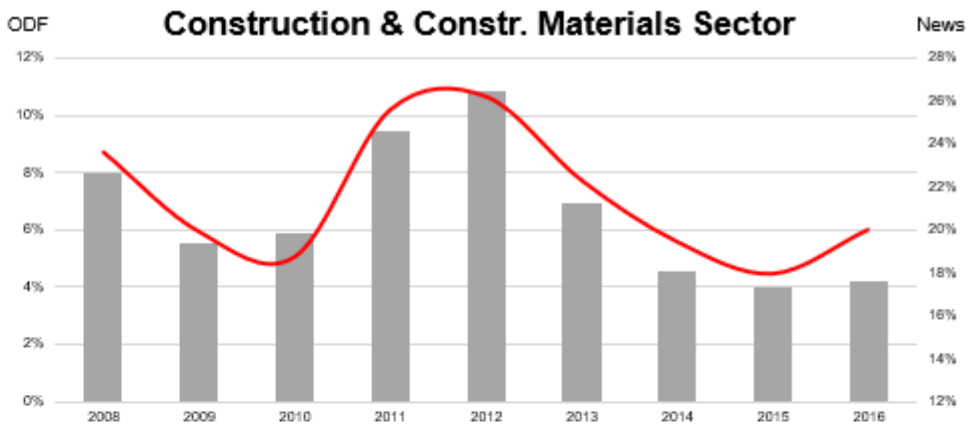
The main objective is to anticipate the behavior of the economy:

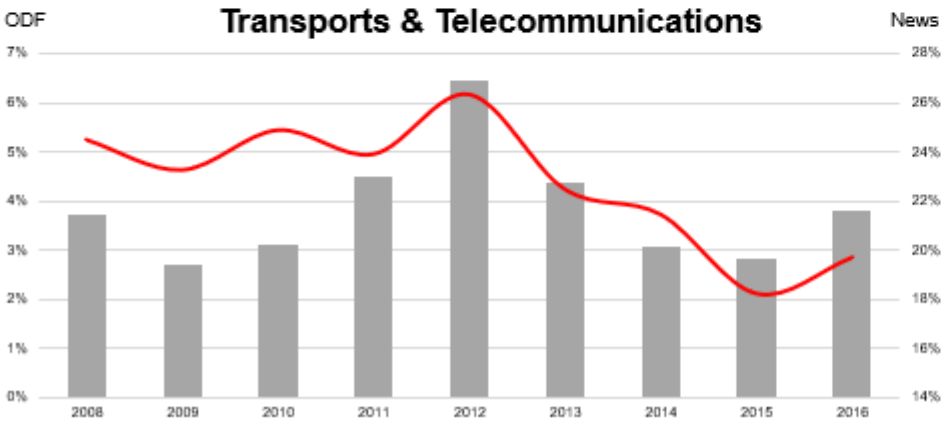
For this reason, the correlation between the one-year news developed score (weight of negative vs. positive news) against the economic evolution of the following year (business ODF) has been verified.

The results at the global level are very positive and allow, through the current information, to predict the general state of the future, being very useful, for example, for the calibration of regulatory models or to predict the evolution of indexes.

Sentiment Analysis: Sectorial TMO predictions -vs- observation





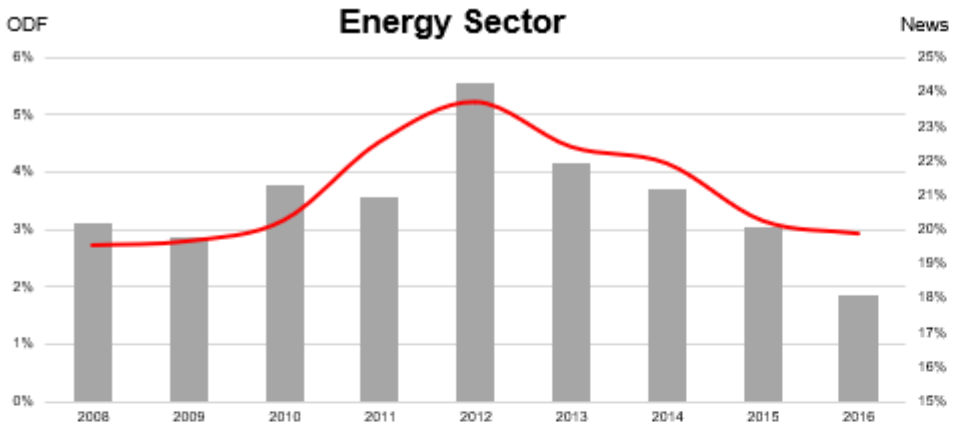


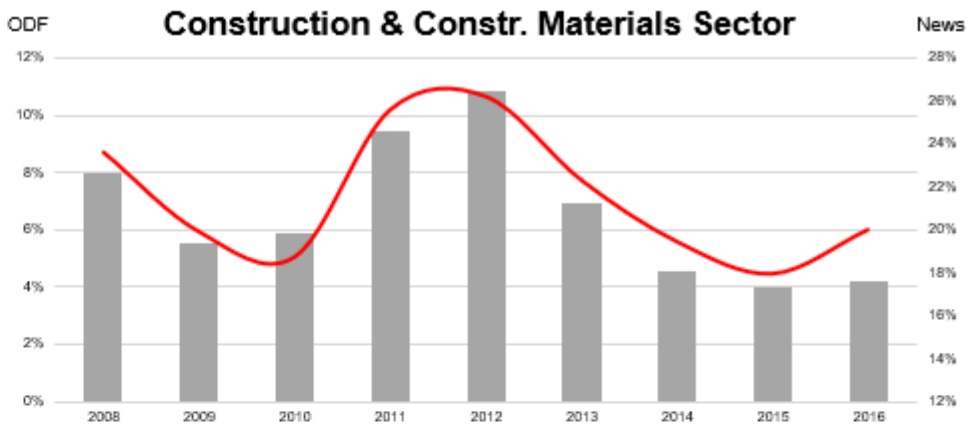
The prediction of the 'sentiment' of the news works well by restricting us to the different sectors of activity

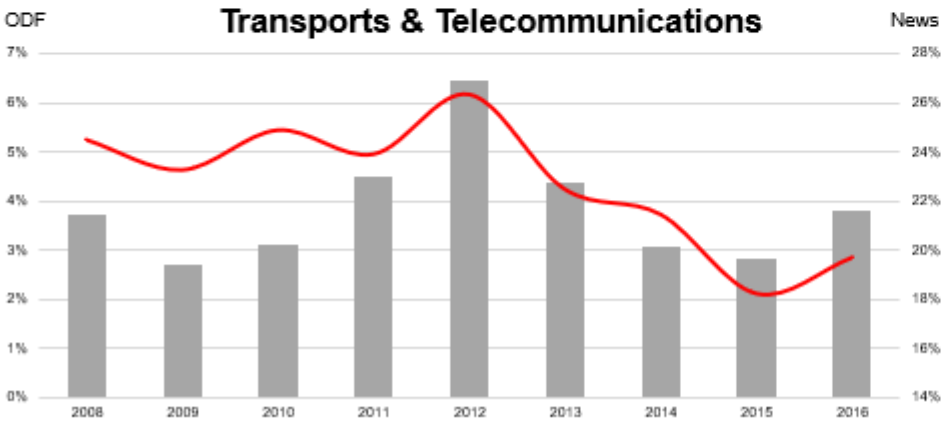
The news sample is restricted to those that deal with a particular sector and is compared with the delinquency of the companies in the sector.

The results at sector level are also very positive. In this case, they allow a greater discrimination in rating models as well as predict the macroeconomic evolution of the sector.

Sentiment Analysis: Sectorial TMO predictions -vs- observation







The prediction of the 'sentiment' of the news works well by restricting us to the different sectors of activity

The news sample is restricted to those that deal with a particular sector and is compared with the delinquency of the companies in the sector.

The results at sector level are also very positive. In this case, they allow a greater discrimination in rating models as well as predict the macroeconomic evolution of the sector.

CNN real life examples: Bio-metrics security (voice & image)

Corporate

CaixaBank develops new biometrics-based financial services

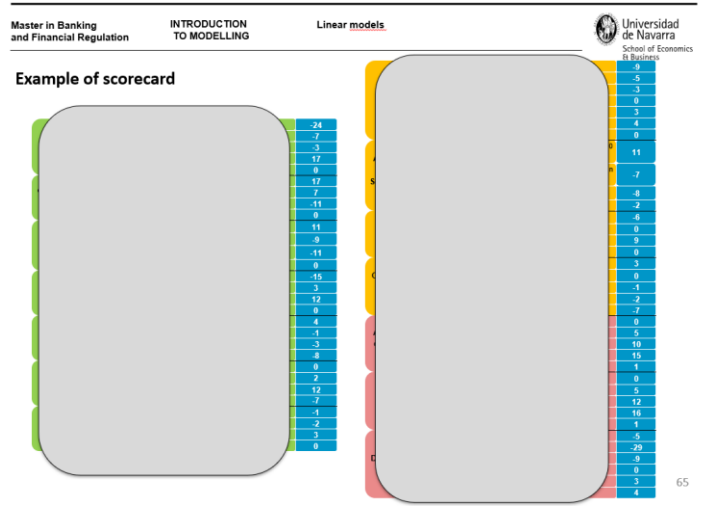
A small taste of predictive models in Retail Banking

Credit Risk department: Credit admision models

- Usually, admission models are logistic models
- The main featutes are related with
  - Socio-demographic information
  - Existing debt
  - Bureaus information
- The target is usually delinquency at 30 days
- One of the main difficulties of building these models is the lack of information about the customer

- A special case of this kind of models are **Credit Fraud models**
  - I this case the flag is usually to look for delinquency on the first payment

Based on calibrated scorecards



Credit Risk department: Customer monitoring

Probability of default (PD) models

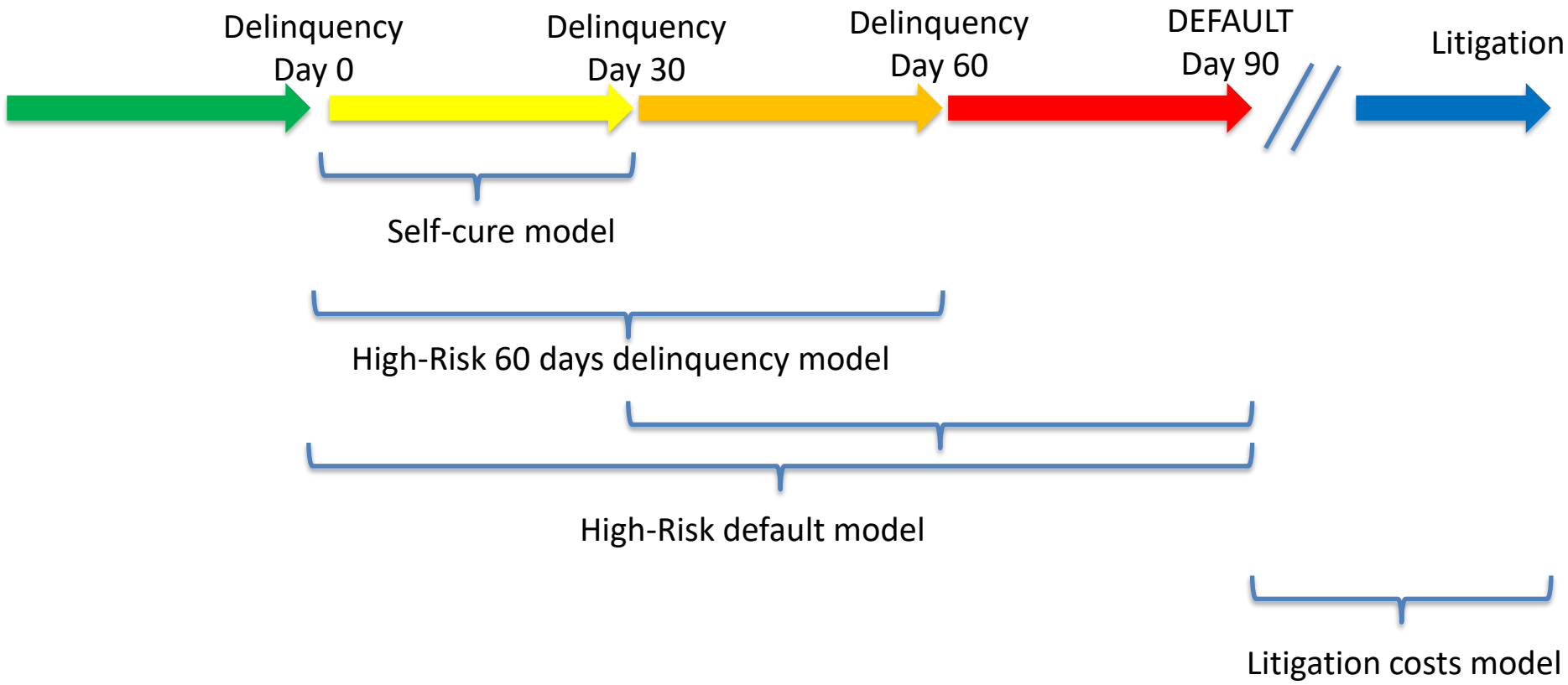
- Binary supervised models which monitor the heath of a credit portfolio at customer level
- They are usually based in behavioral information of the customer
- Such models are critical for financial institutions
- A special pipeline must be used for 'new' customers in order to a biased risk
- The use of Machine learning algorithms is common. A good calibration is key.

Deterioration signals & Early Warnigs Systems (EWS)

- EWS are a set of business rules with anticipate the delinquency/default of customers
- They provide signals and levels so the financial institution can launch preemptive actions to avoid delinquency and thus an increase of the COR
- A 'good' EWS must provide a customer profiling in order to adequate the solution actions
- Usually relying on PD models and balances evolution as input parameters
- Deterioration Signal models are usually EWS at very long timespan (~1y). High non-linear models (deep learning) are common.

+ Regulation obligations (LGD & EAD)

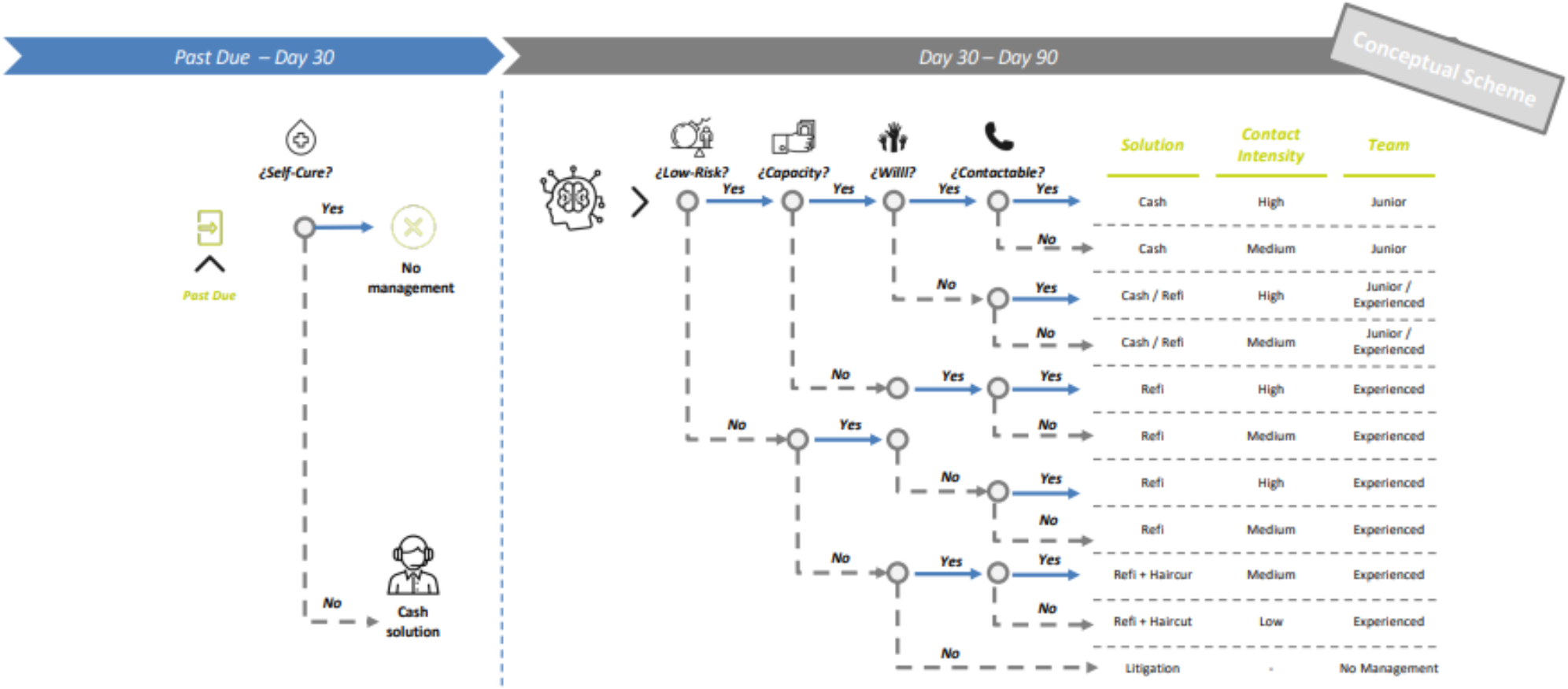
Collections department



+ Behavioural models to adecuate solution

- Payment capacity
- Payment willingness

Collections department



Debt condonation is another possible solution…

Fraud prevention department

Models to detect External transactional frauds: Money laundering & Money Mules

- The goal is to put on hold suspicious transactions so a human analyst
- To detect such cases of behaviors, machine learning models are needed
- The main difficulty is the lack of training samples
- Such models should be implemented in real time in a big data platform
- Such models are now a regulatory must

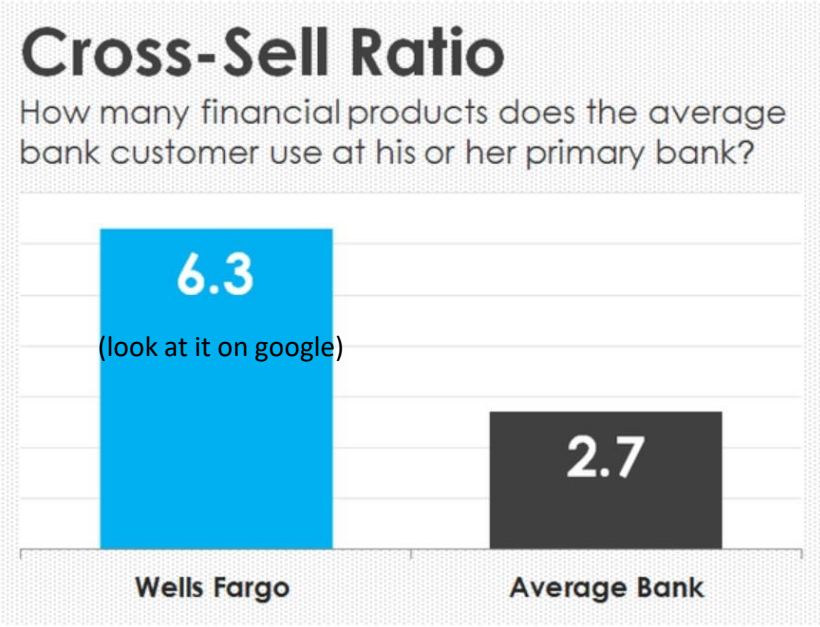
Sales & Interactions department

Propensity models (X-sell and Up-sell)

- The goal is to identify cutomers which are more likely to buy
- They increase the efficiency for the sales department
- Usually there is a models for each financial product
- They are usually machine learning models

Prelation models

- The goal is to provide a customer 'ordering' for several customer service actions (call center)
- Such models are intended to increase efficiency by giving a higher preference to the 'best' customers
- They are usually machine learning models



Marketing department

Campaign decision models

- The models are intended to choose the most optimal and efficient channel of marketing
- They choose the bast product to be announced by modelling the expected customer evolution

Attribution models

- Such models are intended to be able to know from which marketing channel comes the new sales influx
- These models are generally less accurate for analogical channels than for digital channels
- The final results are user to compensate the marketing providers and to optimize future campaigns

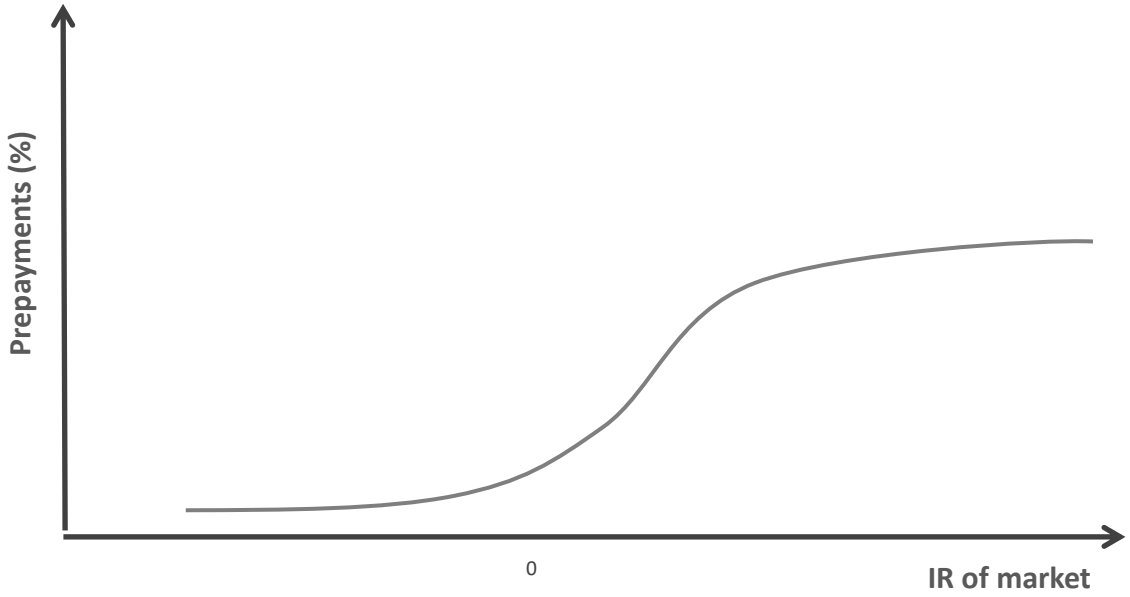
Market Risk department

Duration Assest (IR sensitive)	≈	Duration Liab. (IR sensitive)

In retail banking,

- The duration of liabilities depends exclusively on the interests rates when compared to competition
- Whereas the duration of assets depend of many more factors due to the prepayment options

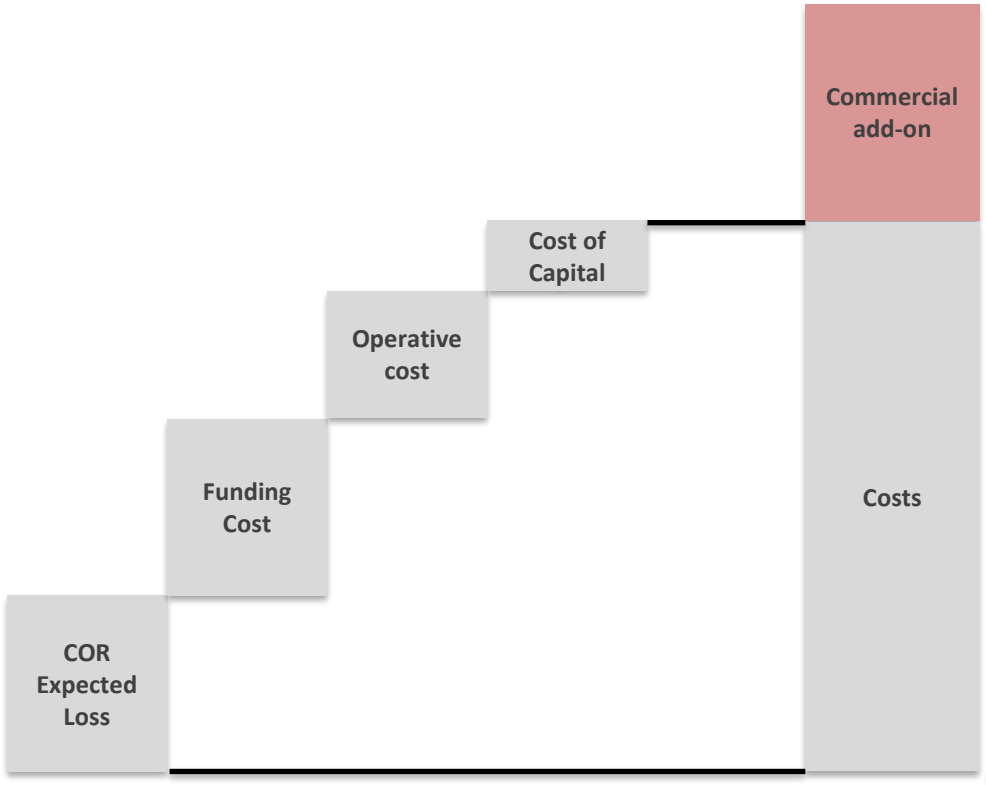
So a financial institution can have a healthy balance, it needs certainty about the lifetime of assets & liab



- 'Univariant analytic model' IR sensitive
- It is extremely important to optimally segment the portfolio

Pricing department

Optimized through customer profile models



$$\frac{\left\{\begin{array}{c}\left[\sum_i^T\left((Index_i + \textbf{spread}) \times Loan\ outstanding_i\right) + \sum_i^T\left(Loan\ available_i \times utilisation\ fee\right)\right] \\ - \\ \left[\begin{array}{c}\left(\sum_i^T\left((Index_i + FTP) \times Loan\ outstanding_i - Capital\ credit\ risk\right)\right) \\ + \\ Administrative\ costs \\ + \\ \left(\sum_i^T\left(PD_i \times LGD_i \times EAD_i\right)\right) \\ \times \\ \frac{1}{(1+r_F)^i} \times (1 - Tax)\end{array}\right]\end{array}\right\}}{\begin{array}{c}LGD\ DT_i \times \left[\Phi\left(\left(\frac{1}{(1-correl_i)^{1/2}}\right) \times \Phi^{-1}(PD\ TTC_i) + \sqrt{\frac{correl_i}{(1-correl_i)}} \times \Phi^{-1}(solvency)\right) - PD\ TTC_i\right] \times \\ \times \left(\frac{1}{1 - 1.5 \times (0.11852 - 0.05478 \times \ln(PD\ TTC_i))^2}\right) \times \\ \times \left[1 + (max\{1; min\{5; M_i\}\} - 2.5) \times (0.11852 - 0.05478 \times \ln(PD\ TTC_i))^2\right] \\ + \\ \frac{15\% \times CET1\left(EL + Administrative\ costs + Cost\ of\ funding + Capital\ credit\ risk \times \frac{RAR\ target}{1 - Tax}\right)}{\left(1 - 15\,\% \times \frac{CET1\ \times RAR\ target}{1 - Tax}\right)}\end{array}}$$

Cost Model

Must-know data “tech” topics for the current financial world

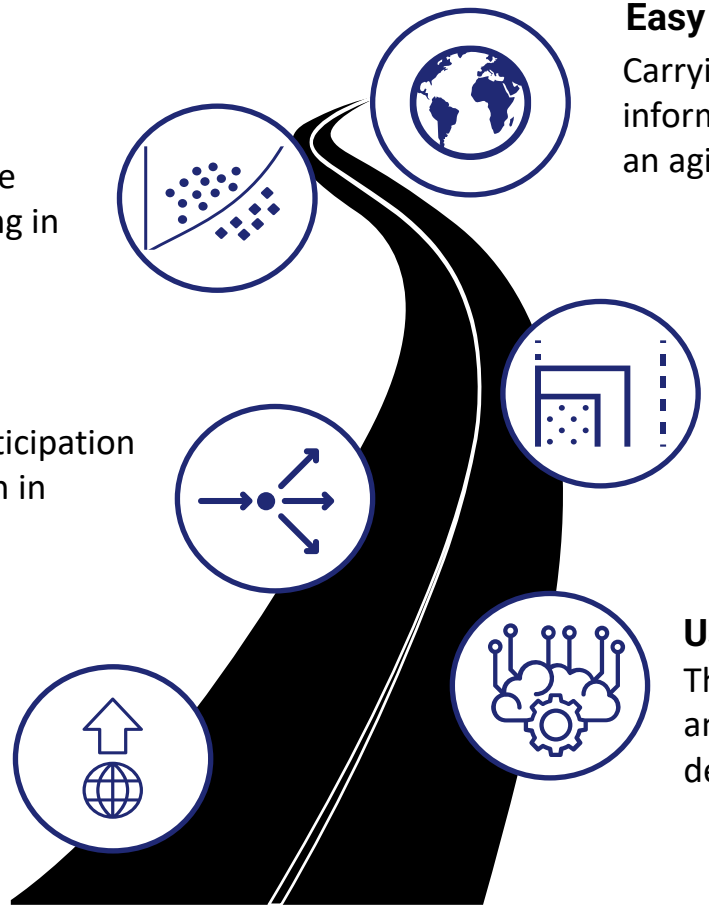
Basics on (distributed) Data-Warehouse systems

Data Governance model

Storage, administration and monetization of data

Ensuring the quality of all data
It allows to obtain a high reliability in the results of data analysis that allow moving in the right direction

Easy data access
Carrying out ad-hoc queries and analysis of any information in a simple way allows you to work in an agile way



Data management as an asset
The 3 above properties allow early anticipation and the consequent proactive reaction in changing business situations

Guarantees in the integrity of the data
Homogenization of data that comes from different sources to be able to be processed in a robust and automated way

Meet business demands
The combination of the previous points allows to satisfy in an agile way the needs of the business management

Use of advanced analytics
The use of cutting edge statistical techniques allows anticipating and managing events with a high degree of detail and precision

Technological environment

Main goal: Optimize the decision taking

Maximizing the use of data for decision making

Provide a coherent view of the reality of the information, resulting in better knowledge of the client and greater use of the information for decision-making

Consistent data generation

Homogenization, interconnectivity, reconciliation and standardization of data and constant construction of the different data sources

Efficient data domain management

Establish data domains and strategies for the domain of customer information, services, products and critical variables

Help prevent fraud and comply with regulations

Use consistent information to prevent fraud, creating positive synergies that also support regulatory compliance and reduce risks

Establish a culture of data governance

Establishment, promotion, promotion and monitoring of a corporate culture of data governance among the different areas of the entity

Strengthening of management and control tools

Continuous innovation in the tools and processes used for data management, as well as the homogenization of processes and licenses within the entity

Establishment of security procedures

Establishment of standards, rules and guidelines for monitoring data protection procedures that prevent risks and optimize the use of information

Efficient management of the data supply chain

Documentation, analysis, administration, improvement and optimization of the different processes that follow the data safely

The Cloud system: Dealing with *Big* data

Cloud computing

From Wikipedia, the free encyclopedia

For the winner of the 2017 Preakness Stakes, see Cloud Computing (horse).

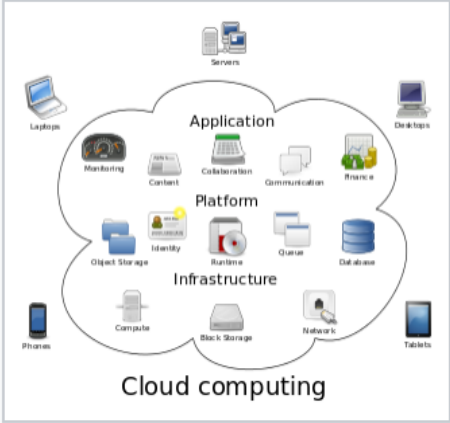
Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet. Large clouds, predominant today, often have functions distributed over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server.

Clouds may be limited to a single organization (enterprise clouds[1][2]), or be available to many organizations (public cloud).

Cloud computing relies on sharing of resources to achieve coherence and economies of scale.

Advocates of public and hybrid clouds note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand.[2][3][4] Cloud providers typically use a "pay-as-you-go" model, which can lead to unexpected operating expenses if administrators are not familiarized with cloud-pricing models.[5]

The availability of high-capacity networks, low-cost computers and storage devices as well as the widespread adoption of hardware virtualization, service-oriented architecture and autonomic and utility computing has led to growth in cloud computing.[6][7][8] By 2019, Linux was the most widely used operating system, including in Microsoft's offerings and is thus described as dominant.[9] The Cloud Service Provider (CSP) will screen, keep up and gather data about the firewalls, intrusion identification or/and counteractive action frameworks and information stream inside the network.[10]



Cloud computing metaphor: the group of networked elements providing services need not be individually addressed or managed by users; instead, the entire provider-managed suite of hardware and software can be thought of as an amorphous cloud.

Technological environment: A big data platform minimum standards

Data ingestion on the platform

INGESTION OF RAW DATA
The data is collected from the sources of origin and stored raw, with minimal quality processes in the area designated for Raw Data

STANDARD DATA ETL
Business rules, validation, and transformations are applied to generate data ready for exploitation and analysis

DERIVED DATA AND BATCH ANALYTICS
Data prepared and transformed from Standard Data by processes and analytical models through propagation mechanisms

Data processing and enrichment

INTAKE AND ANALYSIS NEAR REAL TIME (NRT)
Real-time data is provisioned and stored in the Data Lake

LOW LATENCY ENRICHMENT
Ingested events are enriched with the latest customer information (already stored in the Data Lake)

INTEGRATION WITH SYSTEMS
Integration services feed systems with the results of NRT analytics and data from the Data Lake

Online services for users

ONLINE DATA REPLICA
Relational databases are replicated in NRT and prepared for indexing and cataloging

ONLINE DATA INDEXING
The data is indexed and made available for online consultation, search and analysis

CHANNEL SERVICES
Channel consult business services to download online transactions inquiries

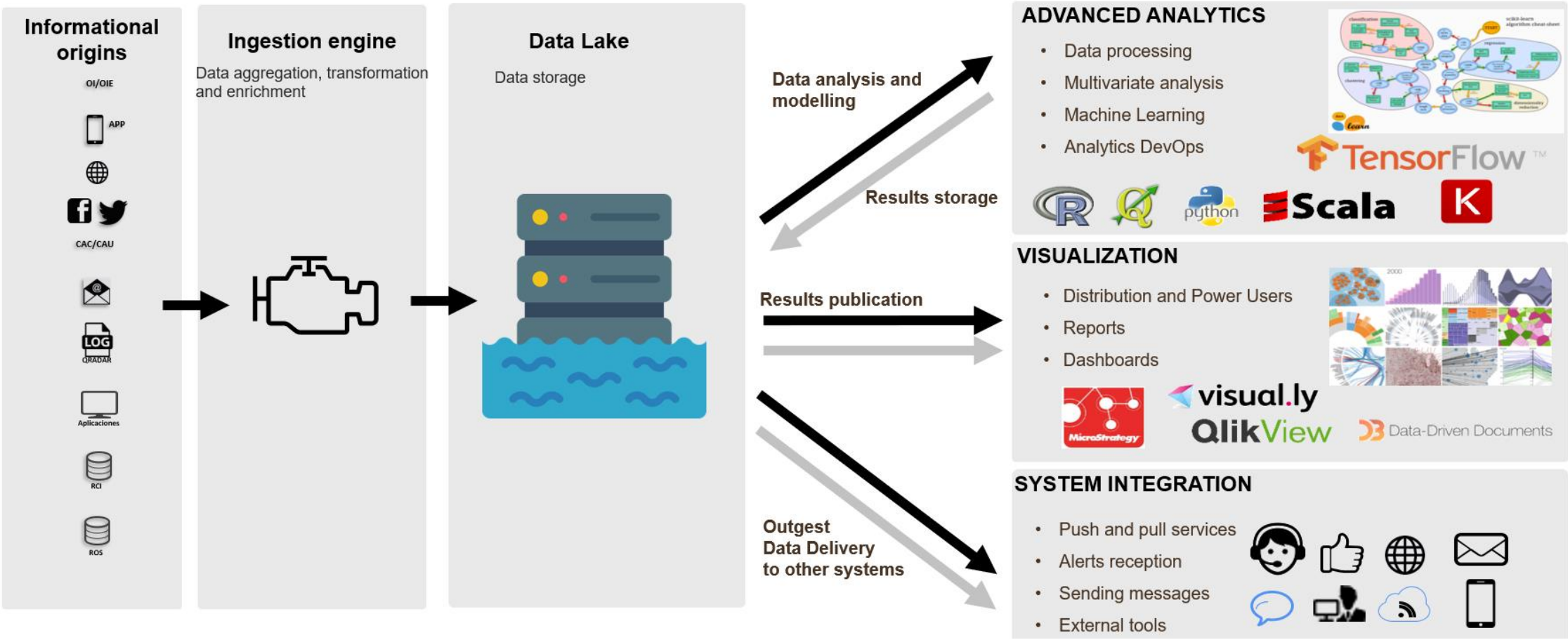
Analytics and Exploitation for users

ANALYTICAL RELATIONAL DB
It allows interacting with relational data of an aggregated nature and analysis requirements with high concurrency, e.g. as follow-up dashboards

DISPLAY AND DISCOVERY
Provides tools to visualize the data of the data lake and to exploit and discover the data

DATALAB / WORKSPACE - ADVANCED ANALYTICS
Analytical work environment for the bank's Data Scientist users, allowing the application of advanced analytical techniques such as modeling through Machine Learning algorithms

Big data platform: Data analysis and exploitation



Analytics, exploitation and modeling of lake data

The user can exploit the Data Lake data with a variety of tools according to their needs, from making simple queries, building complex Machine Learning models to making software packages

Web browser-based tools

- *User-friendly tools*
- These types of interfaces have improved graphical capabilities
- Theyallow easy generation and execution of codes by the user

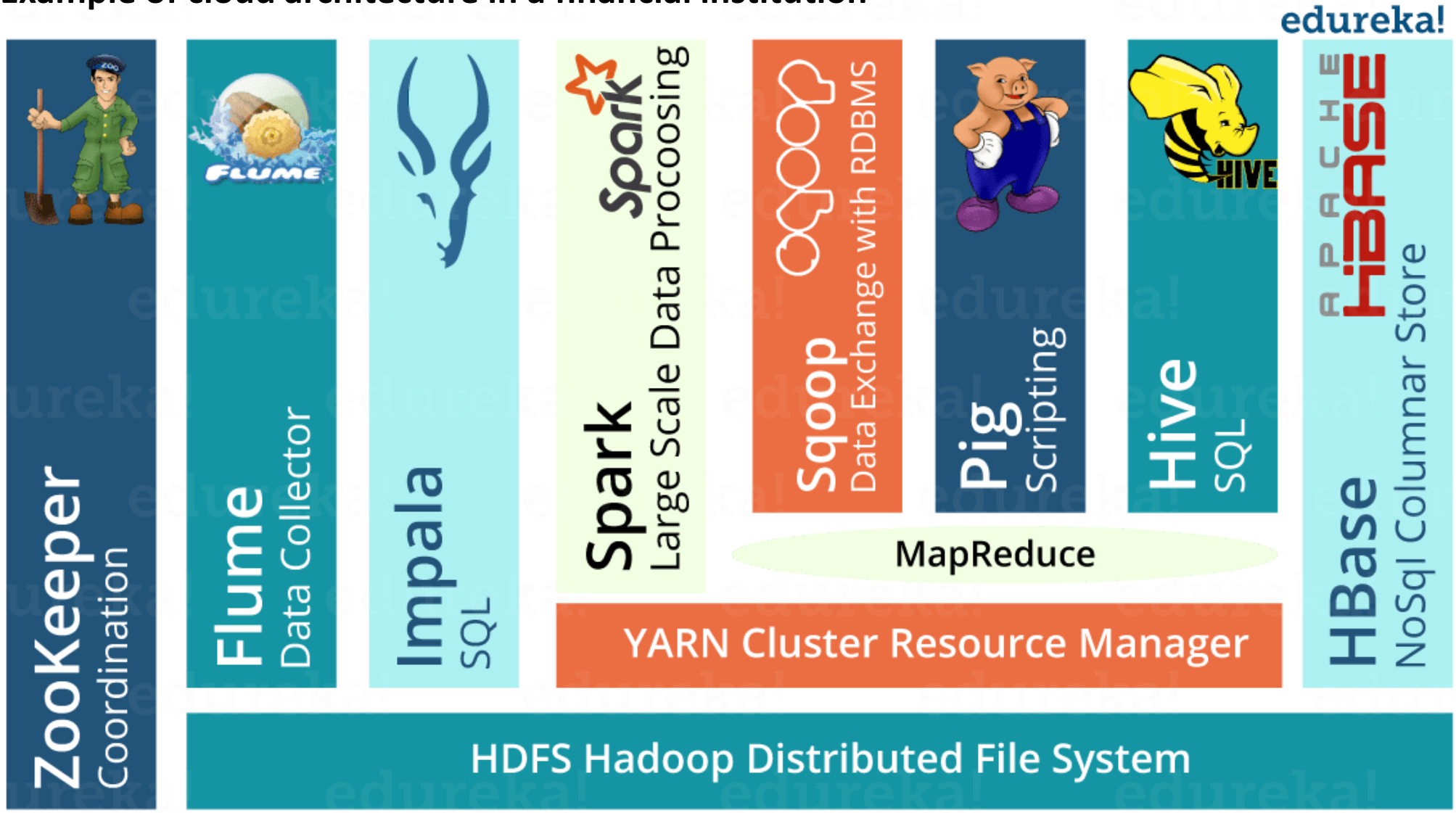
Local tools

- User-friendly tools for users with a high level of programming
- For these tools it is not necessary to use the web browser, but they are used from programs installed on the user's personal computer.

Primary tools

- Most basic form of interaction with the machine for users with knowledge of linux systems and programming
- It lacks comforts for its use, but in return offers a high freedom for the user in all senses

Example of cloud architecture in a financial institution



Python interaction with distributed datasets: PySpark

Python For Data Science *Cheat Sheet*

PySpark - RDD Basics

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

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

`>>> sc.version`	Retrieve SparkContext version
`>>> sc.pythonVer`	Retrieve Python version
`>>> sc.master`	Master URL to connect to
`>>> str(sc.sparkHome)`	Path where Spark is installed on worker nodes
`>>> str(sc.sparkUser())`	Retrieve name of the Spark User running SparkContext
`>>> sc.appName`	Return application name
`>>> sc.applicationId`	Retrieve application ID
`>>> sc.defaultParallelism`	Return default level of parallelism
`>>> sc.defaultMinPartitions`	Default minimum number of partitions for RDDs

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

`>>> rdd.getNumPartitions()`	List the number of partitions
`>>> rdd.count()` `3`	Count RDD instances
`>>> rdd.countByKey()` `defaultdict(<type 'int'>,{'a':2,'b':1})`	Count RDD instances by key
`>>> rdd.countByValue()` `defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})`	Count RDD instances by value
`>>> rdd.collectAsMap()` `{'a': 2,'b': 2}`	Return (key,value) pairs as a dictionary
`>>> rdd3.sum()` `4950`	Sum of RDD elements
`>>> sc.parallelize([]).isEmpty()` `True`	Check whether RDD is empty

Summary

`>>> rdd3.max()` `99`	Maximum value of RDD elements
`>>> rdd3.min()` `0`	Minimum value of RDD elements
`>>> rdd3.mean()` `49.5`	Mean value of RDD elements
`>>> rdd3.stdev()` `28.866070047722118`	Standard deviation of RDD elements
`>>> rdd3.variance()` `833.25`	Compute variance of RDD elements
`>>> rdd3.histogram(3)` `([0,33,66,99],[33,33,34])`	Compute histogram by bins
`>>> rdd3.stats()`	Summary statistics (count, mean, stdev, max & min)

Applying Functions

`>>> rdd.map(lambda x: x+(x[1],x[0]))` `.collect()` `[('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]`	Apply a function to each RDD element
`>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))` `>>> rdd5.collect()` `['a',7,7,'a','a',2,2,'a','b',2,2,'b']`	Apply a function to each RDD element and flatten the result
`>>> rdd4.flatMapValues(lambda x: x)` `.collect()` `[('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]`	Apply a flatMap function to each (key,value) pair of `rdd4` without changing the keys

Selecting Data

Getting

`>>> rdd.collect()` `[('a', 7), ('a', 2), ('b', 2)]`	Return a list with all RDD elements
`>>> rdd.take(2)` `[('a', 7), ('a', 2)]`	Take first 2 RDD elements
`>>> rdd.first()` `('a', 7)`	Take first RDD element
`>>> rdd.top(2)` `[('b', 2), ('a', 7)]`	Take top 2 RDD elements

Sampling

`>>> rdd3.sample(False, 0.15, 81).collect()` `[3,4,27,31,40,41,42,43,60,76,79,80,86,97]`	Return sampled subset of `rdd3`

Filtering

`>>> rdd.filter(lambda x: "a" in x)` `.collect()` `[('a',7),('a',2)]`	Filter the RDD
`>>> rdd5.distinct().collect()` `['a',2,'b',7]`	Return distinct RDD values
`>>> rdd.keys().collect()` `['a', 'a', 'b']`	Return (key,value) RDD's keys

Iterating

`>>> def g(x): print(x)` `>>> rdd.foreach(g)` `('a', 7)` `('b', 2)` `('a', 2)`	Apply a function to all RDD elements

Reshaping Data

Reducing

`>>> rdd.reduceByKey(lambda x,y : x+y)` `.collect()` `[('a',9),('b',2)]`	Merge the rdd values for each key
`>>> rdd.reduce(lambda a, b: a + b)` `('a',7,'a',2,'b',2)`	Merge the rdd values

Grouping by

`>>> rdd3.groupBy(lambda x: x % 2)` `.mapValues(list)` `.collect()`	Return RDD of grouped values
`>>> rdd.groupByKey()` `.mapValues(list)` `.collect()` `[('a',[7,2]),('b',[2])]`	Group rdd by key

Aggregating

`>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))` `>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))` `>>> rdd3.aggregate((0,0),seqOp,combOp)` `(4950,100)`	Aggregate RDD elements of each partition and then the results
`>>> rdd.aggregateByKey((0,0),seqop,combop)` `.collect()` `[('a',(9,2)), ('b',(2,1))]`	Aggregate values of each RDD key
`>>> rdd3.fold(0,add)` `4950`	Aggregate the elements of each partition, and then the results
`>>> rdd.foldByKey(0, add)` `.collect()` `[('a',9),('b',2)]`	Merge the values for each key
`>>> rdd3.keyBy(lambda x: x+x)` `.collect()`	Create tuples of RDD elements by applying a function

Mathematical Operations

`>>> rdd.subtract(rdd2)` `.collect()` `[('b',2),('a',7)]`	Return each rdd value not contained in rdd2
`>>> rdd2.subtractByKey(rdd)` `.collect()` `[('d', 1)]`	Return each (key,value) pair of rdd2 with no matching key in rdd
`>>> rdd.cartesian(rdd2).collect()`	Return the Cartesian product of rdd and rdd2

Sort

`>>> rdd2.sortBy(lambda x: x[1])` `.collect()` `[('d',1),('b',1),('a',2)]`	Sort RDD by given function
`>>> rdd2.sortByKey()` `.collect()` `[('a',2),('b',1),('d',1)]`	Sort (key, value) RDD by key

Repartitioning

`>>> rdd.repartition(4)`	New RDD with 4 partitions
`>>> rdd.coalesce(1)`	Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                          'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

DataCamp
Learn Python for Data Science Interactively

Interlude on servers Operating Systems

Many new software provided by IT companies is based on Linux Operative System (opensource).

Linux computers use the language **BASH** as tool of user interaction.

WINDOWS KERNEL
VERSUS
LINUX KERNEL

WINDOWS KERNEL	LINUX KERNEL
A commercial kernel of Windows operating system developed by Microsoft	An open source Unix-like computer operating system kernel
Developed by Microsoft	Developed by Linux Torvalds
There is no access to the source code	There is full access to the source code
Has hybrid architecture	Has monolithic architecture
Uses Access Control List (ACL) for file access control	Uses traditional Unix permissions and POSIX ACL for file access control
Includes a GUI stack in the kernel	GUI stack is in the user-space
Supports multiple users and sessions but depends on the versions and editions	Supports 100 % multi-user environment
Maintains a registry to store configurations	Maintains the configurations in files
Has different mechanisms for different devices	Every device is a file

Visit www.PEDIAA.com

Interlude on servers Operating Systems: BASH basics

Bash scripting cheatsheet

Example

```
#!/usr/bin/env bash

NAME="John"
echo "Hello $NAME!"
```

Conditional execution

```
git commit && git push
git commit || echo "Commit failed"
```

Conditionals

```
if [ -z "$string" ]; then
  echo "String is empty"
elsif [ -n "$string" ]; then
  echo "String is not empty"
fi
```

See: Conditionals

Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

Functions

```
get_name() {
  echo "John"
}

echo "You are $(get_name)"
```

See: Functions

Brace expansion

```
echo {A,B}.js
```

{A,B}	Same as A B

String quotes

```
NAME="John"
echo "Hi $NAME"  #=> Hi John
echo 'Hi $NAME'  #=> Hi $NAME
```

Shell execution

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"
# Same
```

See Command substitution

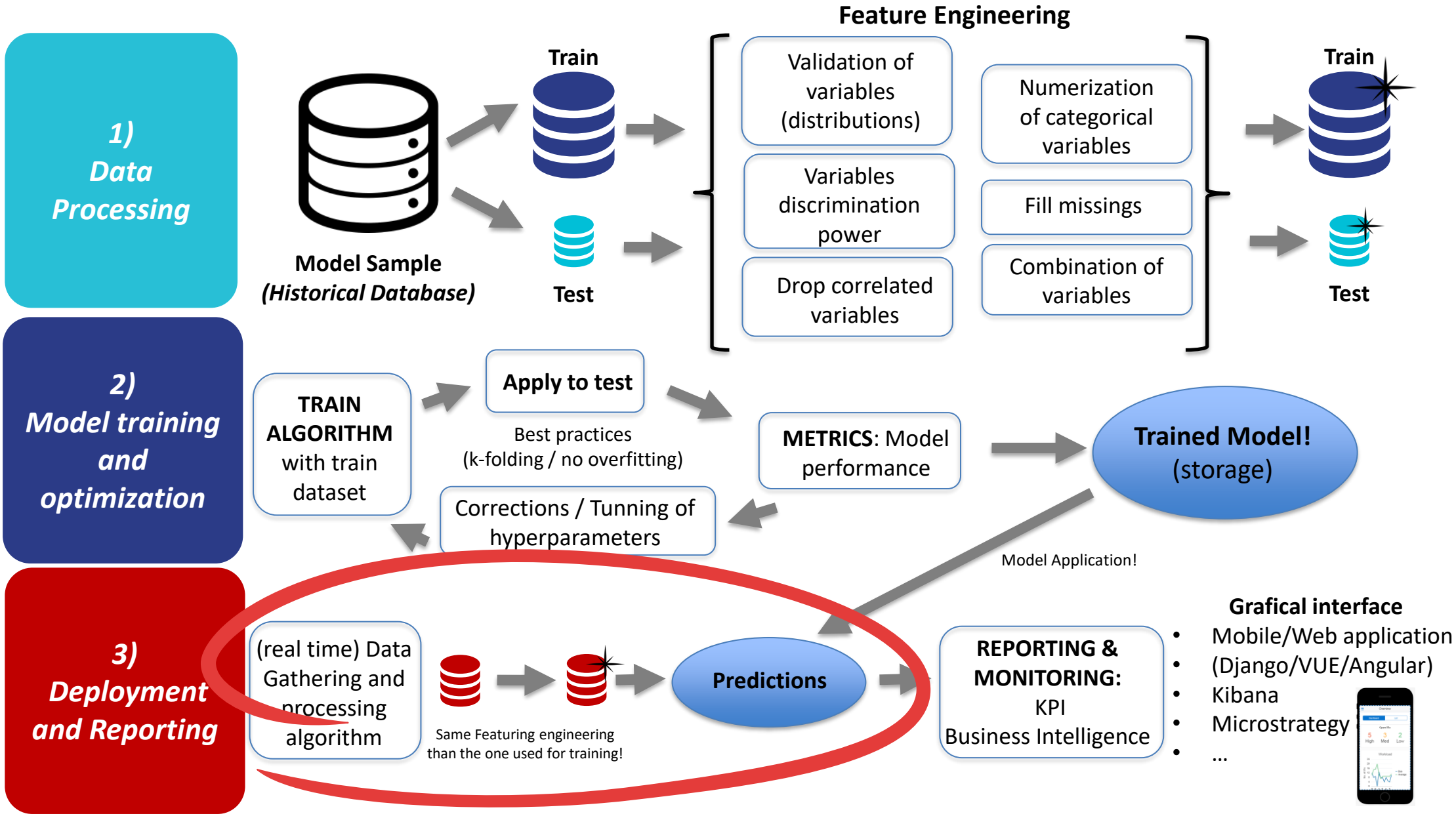
Strict mode

```
set -euo pipefail
IFS=$'\n\t'
```

See: Unofficial bash strict mode

Basics on Model Deployment

General flow of a (large) modeling project of supervised learning



Software deployment

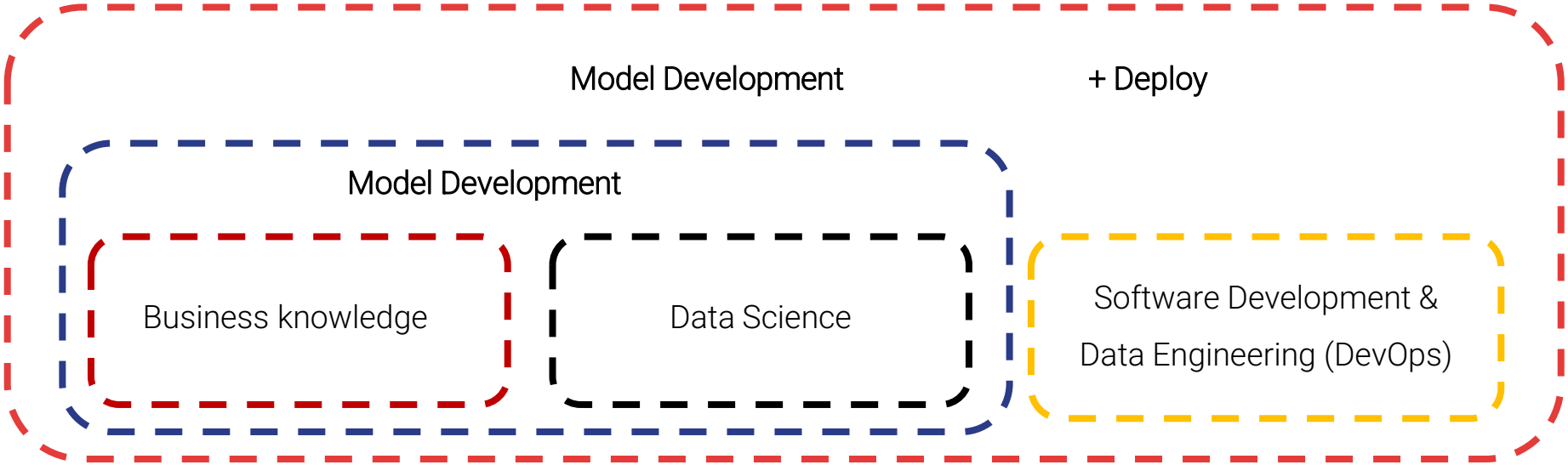
From Wikipedia, the free encyclopedia

Software deployment is all of the activities that make a software system available for use.[1]

The general deployment process consists of several interrelated activities with possible transitions between them. These activities can occur at the producer side or at the consumer side or both. Because every software system is unique, the precise processes or procedures within each activity can hardly be defined. Therefore, "deployment" should be interpreted as a *general process* that has to be customized according to specific requirements or characteristics.[2]

The deployment process must be done with great attention to detail

The operationalization (deployment) of models refers to preparing the built models for (recurring) use, in some cases it is required that the model be used automatically.



Model deployment subjected to the final use

'Manual' Models

Models that are **executed ad-hoc** without any rigorous time restriction

Normally, these types of models are designed to predict a magnitude of a given stock for a long period of time, assuming that there will be no significant changes in the model variables. For example, predicting the sale price of a portfolio of properties at the beginning of the quarter where its appraisal value remains unchanged for a long time.

Automatic Models

They are models that **run automatically on a computer system**, therefore subject to time requirements and strict computational quotas.

This type of model is **executed autonomously when receiving an external stimulus (Trigger)**, for example periodically at a certain time. This type of model usually has to deal with a significant population instability of its variables, therefore its robustness is an important factor. Ideally they should be accompanied by a retraining schedule and an unforeseen contingency plan.

Operativización de modelos lineales de scoring

Scoring models are traditionally translated into SQL code (ORACLE or SAP systems)

The implementation codes are made by IT services and not by the people who designed the model. The main steps are:

1. **Divide each of the variables** into the same categories as those used in the construction of the model with a set of “if” and “case when”.
2. **Assign each category** the corresponding score provided by the model.
3. **Add up all the scores to get the final score** for each client

Deploy of automatic models

The deployment of autonomous models is normally subject to strong external conditions. Yet the central premise is that the data on which the model will act must have gone through exactly the same processing that the training data underwent. The final product (model + software) ends up being a balance between different factors:

Expectations of the result by the client

Complexity of the model algorithm

Available computational ressources

Save type of source data

Available tools

- *IT structure (servers)*
- *Available programming languages*

Feature Engineering Complexity

Time available to build the complete software

Connection between servers and databases

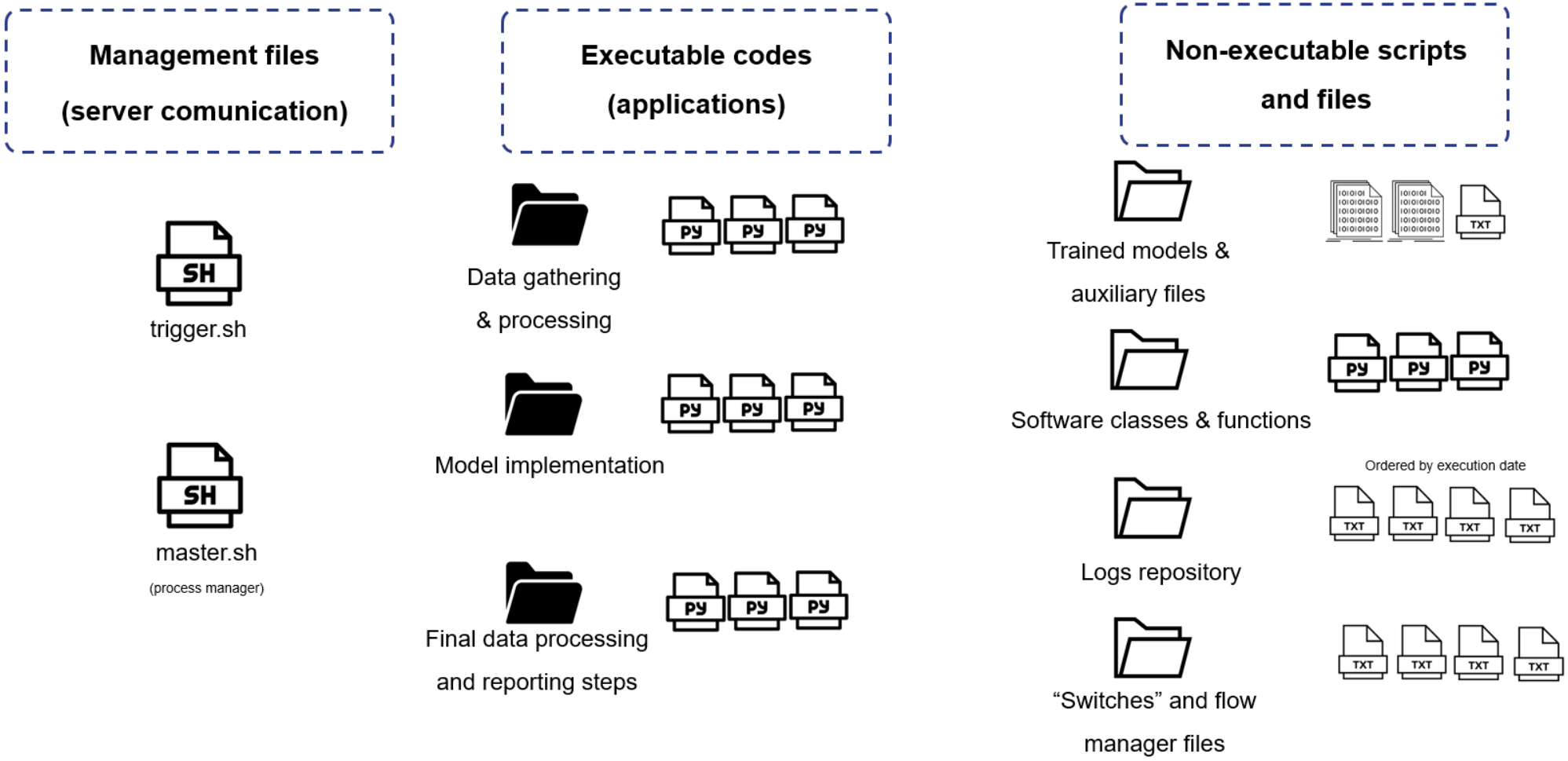
Servers saturation

Available execution time

Source data update rate

Deploy of automatic models: Software elements

Ideal structure that a directory should have with a **complete software for deploying Python models in a generalist Linux server scenario**



Model deployment in distributed systems

A Trigger is a mechanism that initiates the main execution of the software through external algorithms of the "switch" class. They are usually implemented in the IT infrastructure where the model is running. Among others, there are 3 main types of Triggers:

Time switch

Time-based Trigger

This type of Trigger is activated by the calendar of the server system where the models are operative.

File switch

Files-based Trigger

This type of Trigger is activated by the generation or update of files in a specific directory

"Power-on" switch

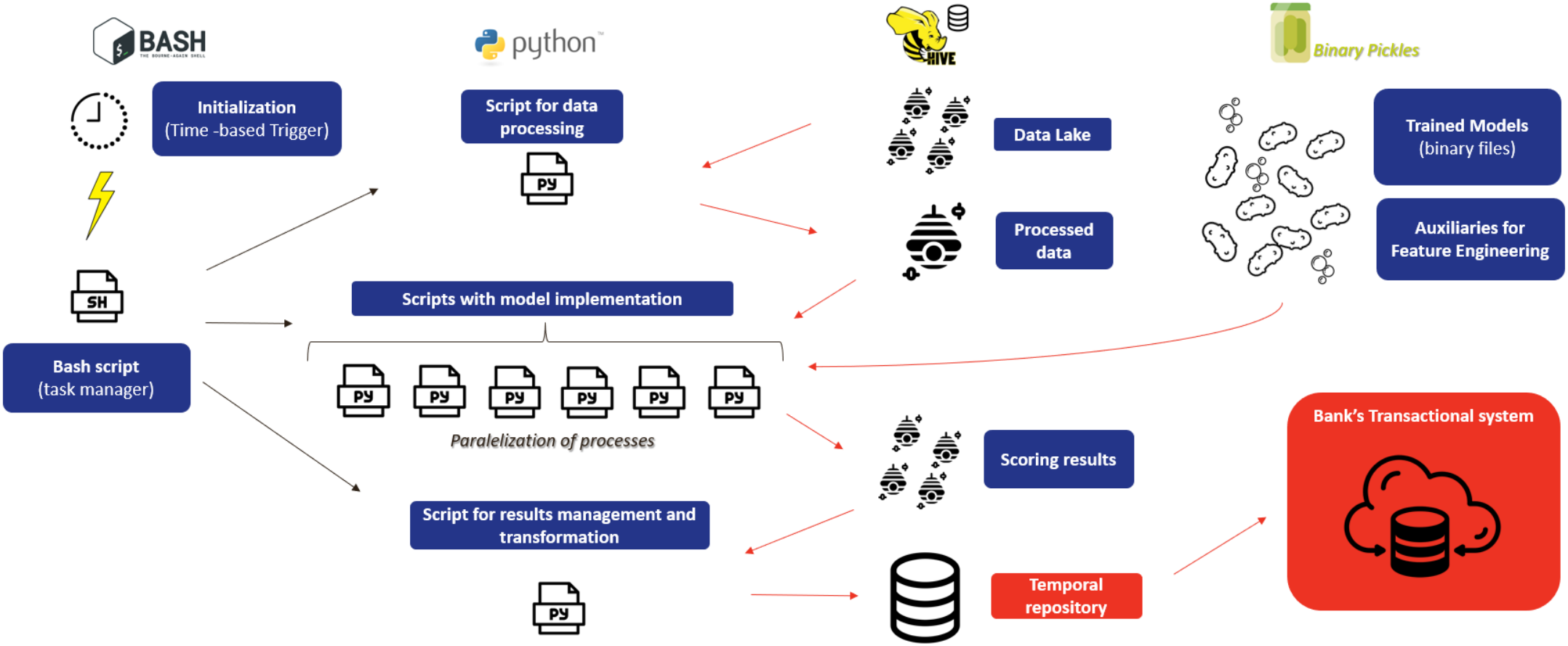
Boot-based Trigger

This type of Trigger is activated when a specific machine is running, along with its other intrinsic initialization processes

Example of a time trigger file in python

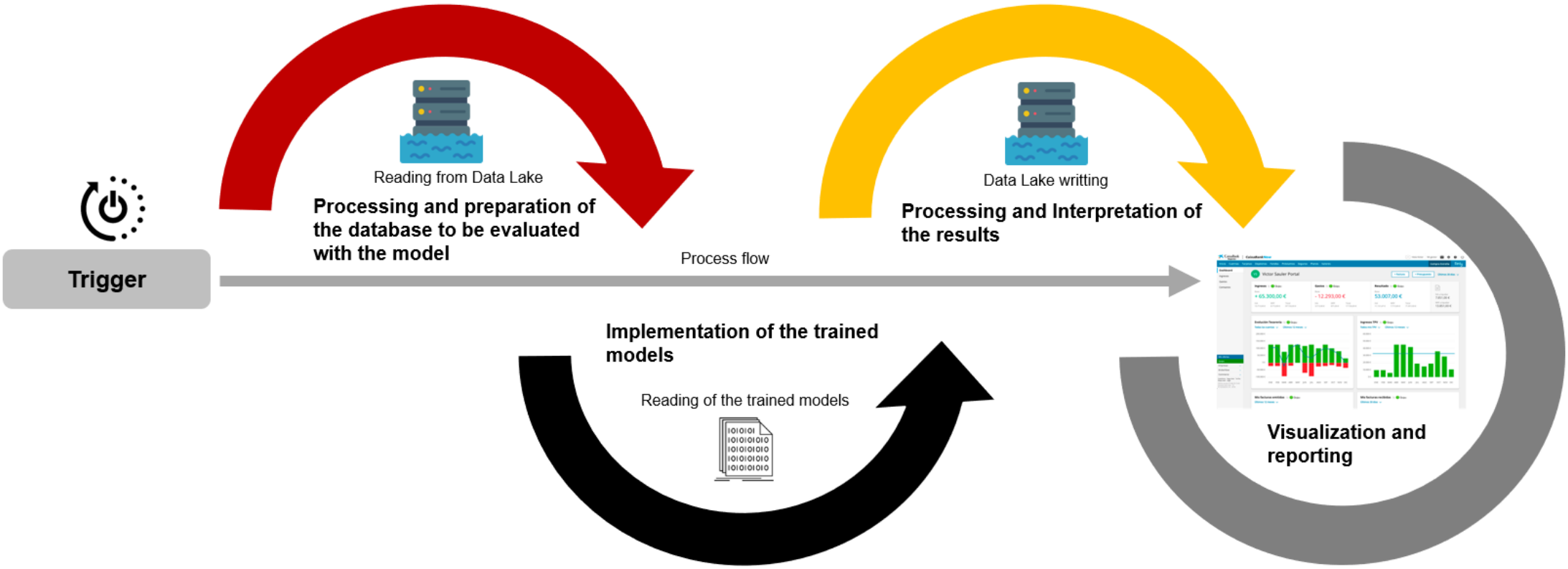
```python
waitingtime      =180
hour_ref         =21
minute_ref       =30
switch_files     =["switch_autocura.txt",\
                  "switch_alto_riesgo.txt"]
print("+ Waiting 5 min for the parallel codes to start processing")
print("   --> Sleeping for "+str(300)+" seconds")
time.sleep(300)
goon = False
trie=0
while goon==False:
    print("------------------ ATTEMPT "+str(trie)+" on "+str(datetime.date.today())+" ------------------------")
    print(datetime.datetime.now())
    switch="on"
    for file in switch_files:
        _switchfile=open(file, "r")
        _switch    =_switchfile.read()
        print("     -> '"+_switch+"' is what is written in "+file)
        _switchfile.close()
        if _switch=="off": switch="off"
    print("  - Status: switch is "+str(switch)+" at "+str(int(datetime.datetime.now().hour))+":"+str(int(datetime.datetime.now().minute)))
    if switch=="on" or (int(datetime.datetime.now().hour)>=hour_ref and int(datetime.datetime.now().minute)>=minute_ref):
        print("");print(">>> Stop wating and writing results to host");print("")
        goon=True
    else:
        print("  ==> Conditions not fulfilled -> sleeping for "+str(waitingtime)+" seconds")
        time.sleep(waitingtime)
    trie+=1
```

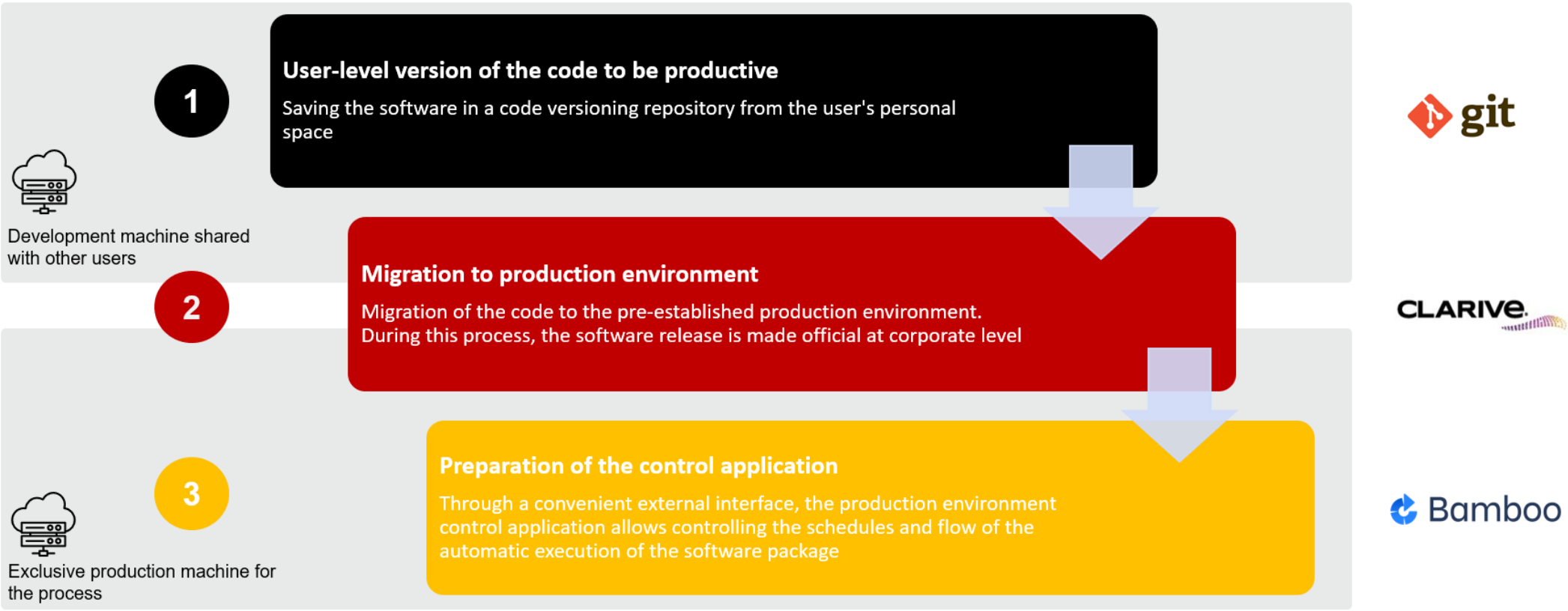
Model deployment in distributed systems



Deploy of models

The Big Data platform allows you to build custom software packages that perform specific tasks in a remote and automated way. These tasks include the implementation of Machine Learning models that the user has developed and whose results can later be sent to an external tool.



‘Productivization’ of models

Once the software package that contains the operationalization of the models has been tested by the user in their personal space, the algorithm's productivity process can be carried out. Productivization consists of the corporate release of the software package, being officially integrated into the bank’s systems.