# Supervised Learning

## Data Science & Business Analytics

Lesson 2

**Tiago Cabo**

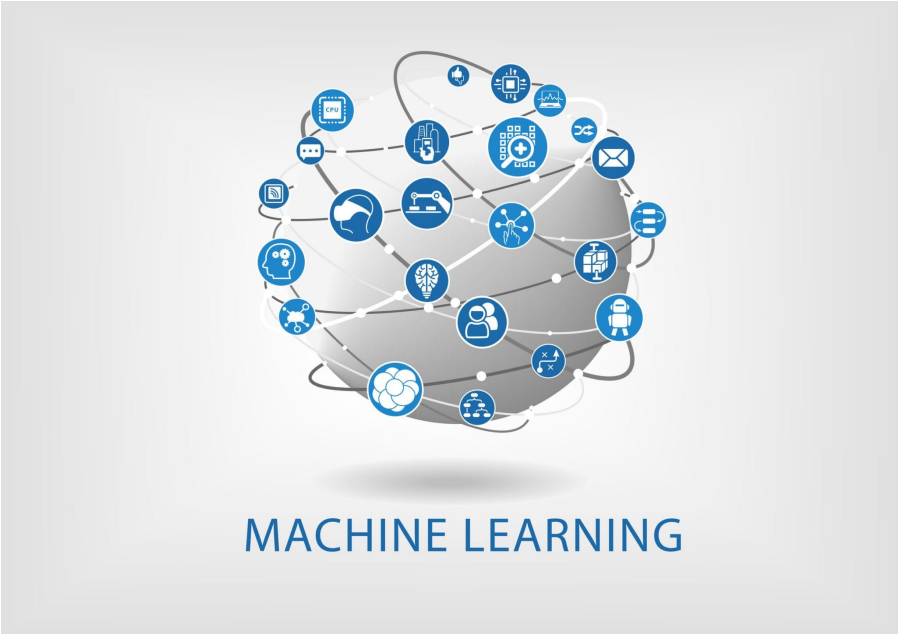
# Lesson Plan

- Some ML history
- Introduction to ML projects (MLOPS)
- Types of Machine Learning
- Supervised Learning Algorithms
- Classification metrics
- Bias/variance trade-off
- Exercises

# ML History

According the IBM “Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. ”

In my opinion, both ML or AI are terms used for the same. However, industry tends to call DL when the algorithms use some kind of neural network.



# ML Applications

**Retail:** Market basket analysis, Churn prediction

**Finance:** Credit scoring, Fraud Detection

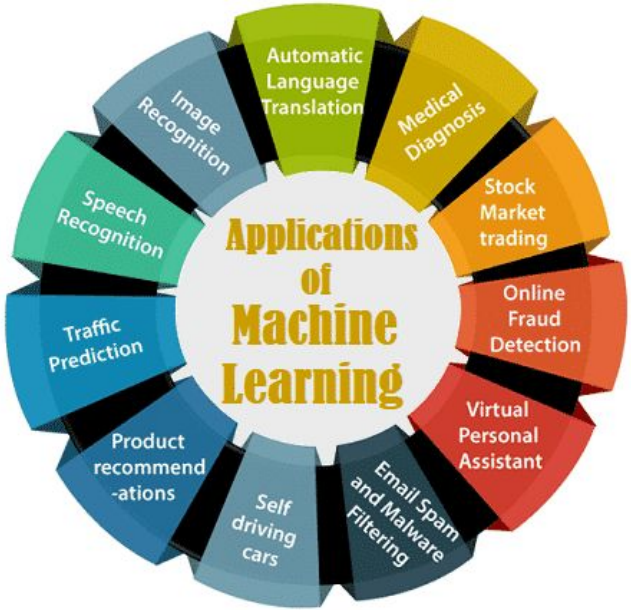
**Manufacturing:** Control, robotics

**Medicine:** Medical diagnosis

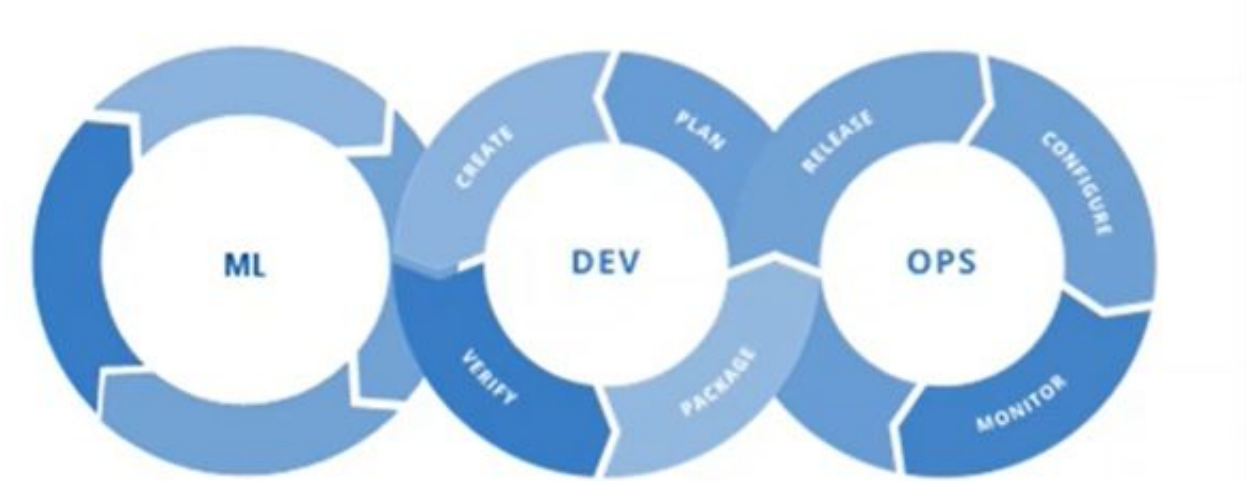
**Telecommunications:** Spam filters, intrusion detection

**Bioinformatics:** Motifs, alignment

**Web:** Ads, Search engines



# MLOPS




| Experiment | Develop | Operate |
|---|---|---|
| Data Acquisition | Modeling + Testing | Continuous Delivery |
| Business Understanding | Continuous Integration | Data Feedback Loop |
| Initial Modeling | Continuous Deployment | System + Model Monitoring |

# Focus on modeling

Most Machine Learning projects consist on two separate parts:

- Train Pipeline
  - Batch pipeline: Model is trained frequently: once a week is most common
    - Ex:  Models to predict images, pretrained recommendations
  - Event driven: Model is retrained when a new or a defined amount of labels is collected
    - Predictive maintenance algorithms
- Predict pipeline
  - Batch pipeline: Consists on calculating the results in predefined schedules i.e. once a day, once a week, …
    - Pre-processed recommendations
  - Event driven (real time): Predict are done when some event is triggered.
    - Ex:  Personalized recommendations

# Types of Learning Algorithms

- **Supervised Learning**

To train we need to provide the real labels into the models.

- **Unsupervised Learning**

The model learns without labels. This can be done by similarity.

- **Semi-supervised Learning**

A mix of supervised and unsupervised. Commonly used when the labels are few.

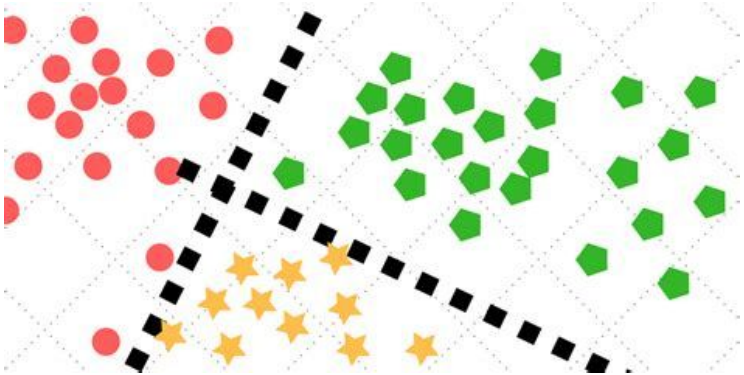
- **Reinforcement Learning**

The models starts with bad predictions, and then corrects itself given the response

# Supervised Learning

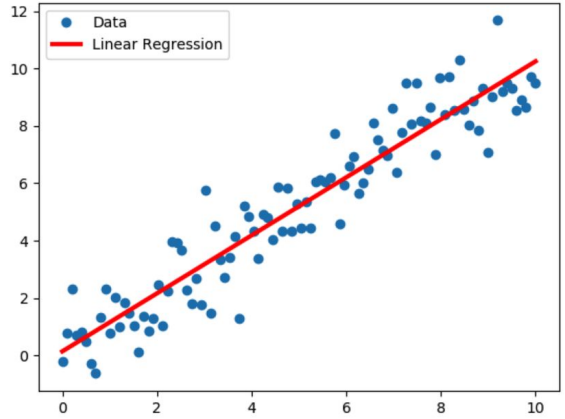
- **Classification**

Applied in discrete data.



- **Regression**

Applied to continuous data. The decision is made given the position of the data compared with the regression line

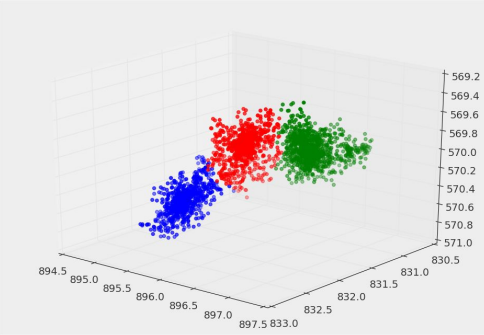


# Unsupervised Learning

**Clustering algorithms**

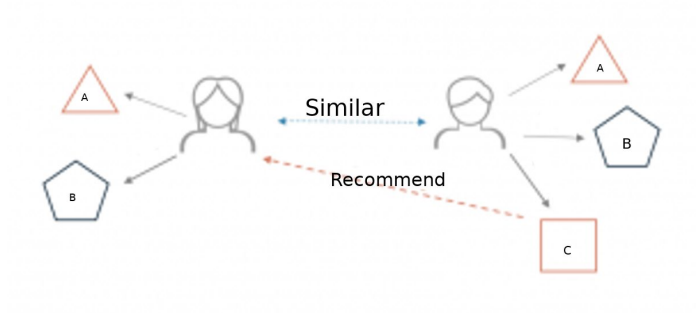
Examples:

- K-means clustering
- Gaussian Mixture Models (GMMs)
- DBSCAN




- **Association**

Association rules allow you to establish associations amongst data objects inside large databases. This unsupervised technique is about discovering interesting relationships between variables in large databases. For example, people that buy a new home most likely to buy new furniture.



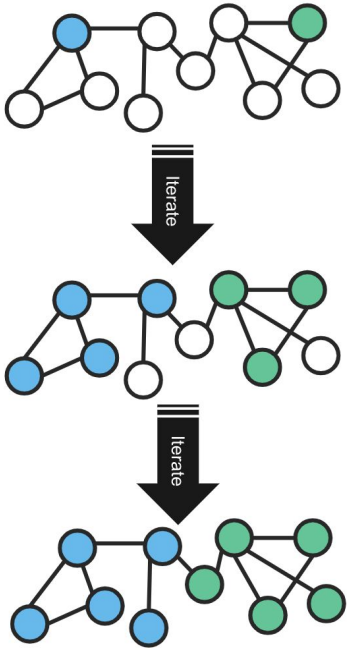
# Semi-Supervised Learning

**Main applications:**

- Social Networks
- Anomaly detection

This type of models allows very nice prediction in use cases were the labels are very few. Allowing a quicker time to market.

**Label Propagation**
Community Detection



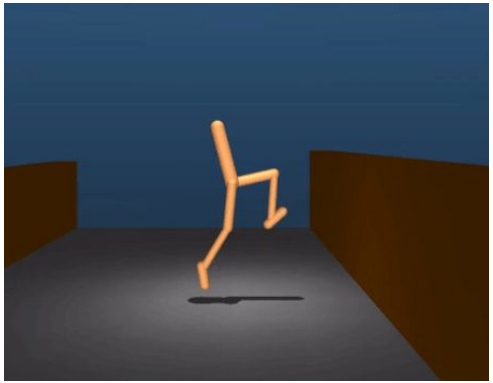

Nodes adopt labels based on neighbors to infer clusters.

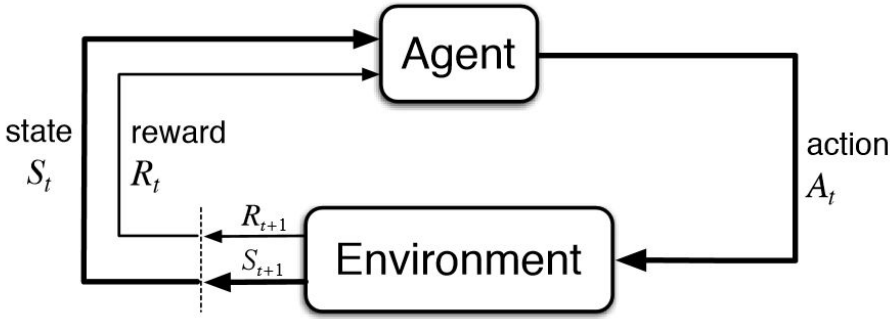
# Reinforcement Learning

**Main Algorithms:**

- Deep neural networks
- Genetic Algorithms

The main learning mechanism is called reward function. The purpose behind is to mimic the way humans learn.



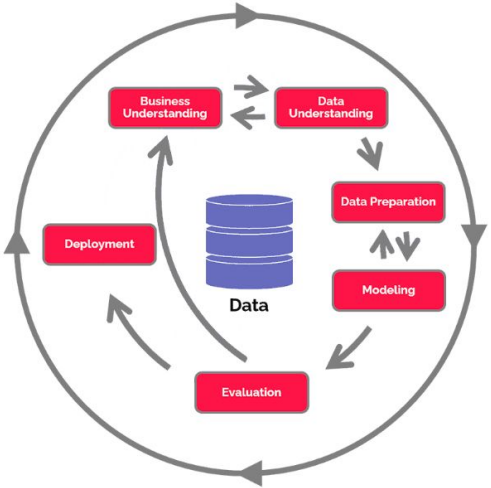


# CRISP-DM

Another reference from the MLOPS slide, is what is called CRISP-DM data processing framework. This is an attempt of standardizing the steps in machine learning projects. In practise, this is a little fuzzy, but at least the terminology is somewhat consistent across the industry. Please check this reference for more info: https://www.datascience-pm.com/crisp-dm-2/

The main steps are:

- Data acquisition
- Data preprocessing
- Feature Engineering
- Feature Selection
- Model Learning
- Model Evaluation
- Model deployment
- Model Monitoring



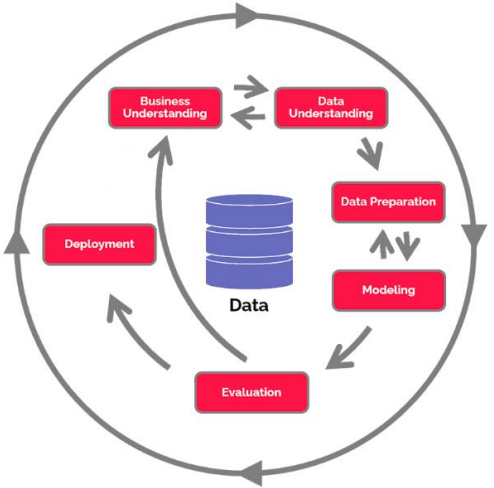
# CRISP-DM

Another reference from the MLOPS slide, is what is called CRISP-DM data processing framework. This is an attempt of standardizing the steps in machine learning projects. In practise, this is a little fuzzy, but at least the terminology is somewhat consistent across the industry. Please check this reference for more info: https://www.datascience-pm.com/crisp-dm-2/

The main steps are:

- ~~Data acquisition~~ Not scope of this course
- Data preprocessing
- Feature Engineering
- Feature Selection
- Model Learning
- Model Evaluation
- ~~Model deployment~~
- ~~Model Monitoring~~

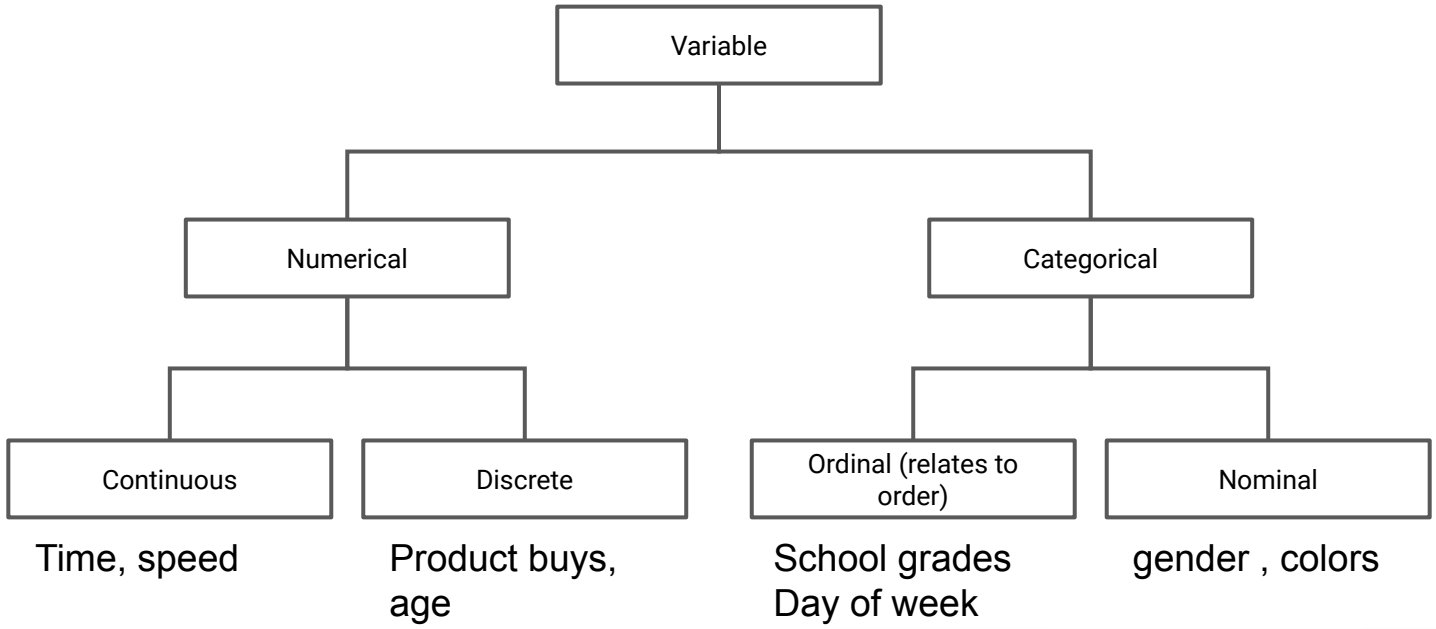
We will talk this briefly, but not that in depth



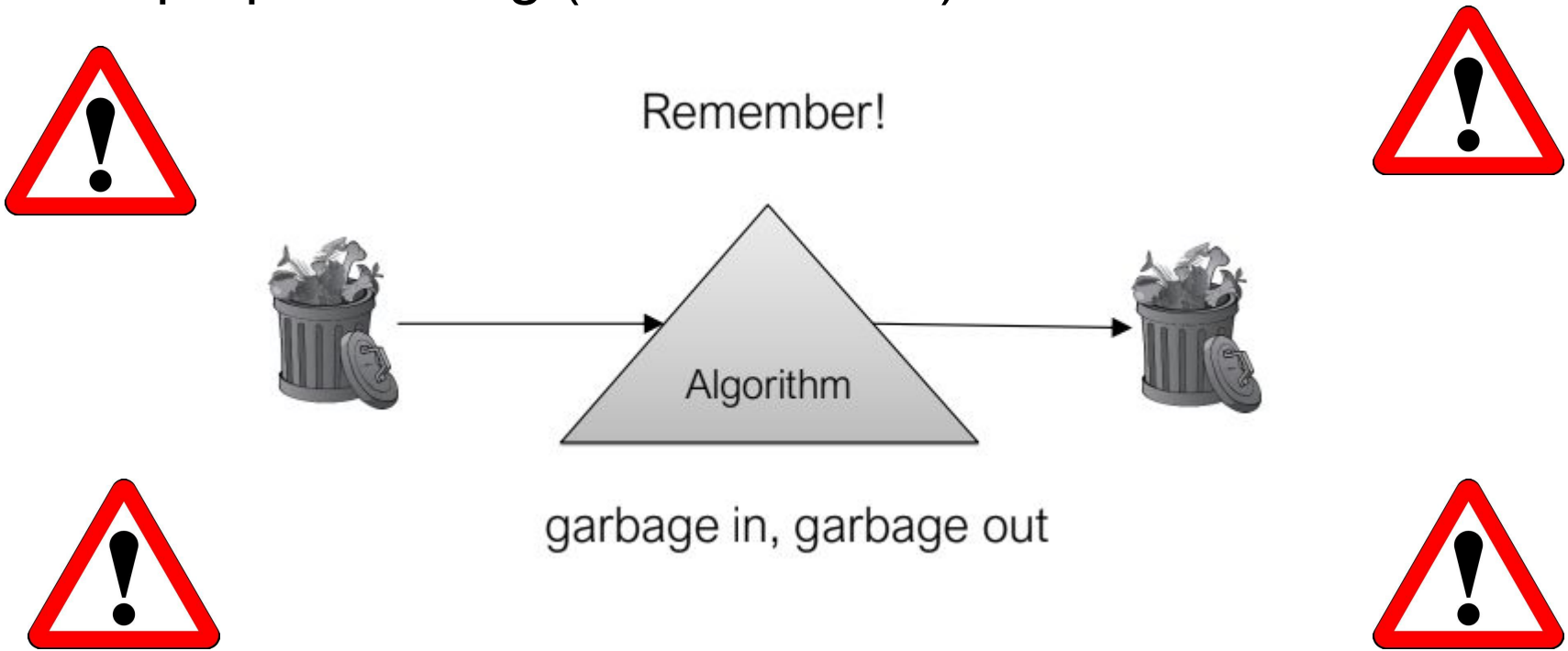
# Let’s talk about Supervised Learning

# Let’s talk about data preprocessing

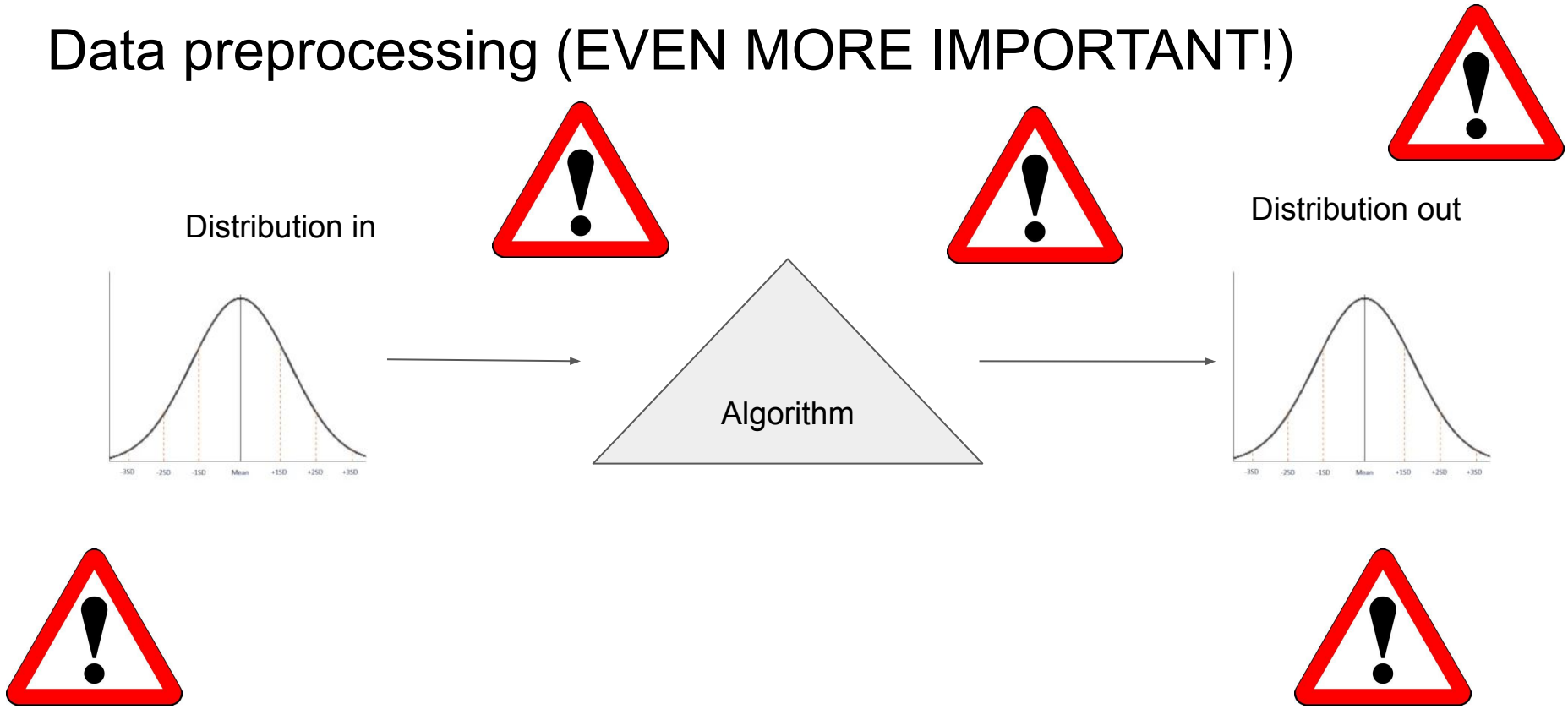
# Data Types (Remember)



# Data preprocessing (IMPORTANT)



# Data preprocessing (EVEN MORE IMPORTANT!)



# Common Data preprocessing

- Data Cleaning:
  - Fill missing values
    - Fill with average
    - Fill with zeros
  - Remove outliers
- Data integration
  - Join datasets and ensure consistency
  - Remove correlated data
- Data transformation
  - Normalization
  - Encode categorical data
  - Group numerical data into bins

# Example

https://colab.research.google.com/drive/1hCXzwphIpaR8zbMluEeNcMOaYgdctiaZ?usp=sharing

# Supervised learning models

- Linear Discriminant Analysis (LDA)
- K-Nearest Neighbors (KNN)
- Naïve Bayes (NB)
- Logistic Regression
- Artificial Neural Networks (ANN)
- Support Vector Machines (SVM)
- Decision Trees (DT)
- Random Forests (RF)
- Gradient Boosting Machine (GBM)
- Light Gradient Boosting Machine (LightGBM)
- eXtreme Gradient Boosting (XGBoost)
- CatBoost

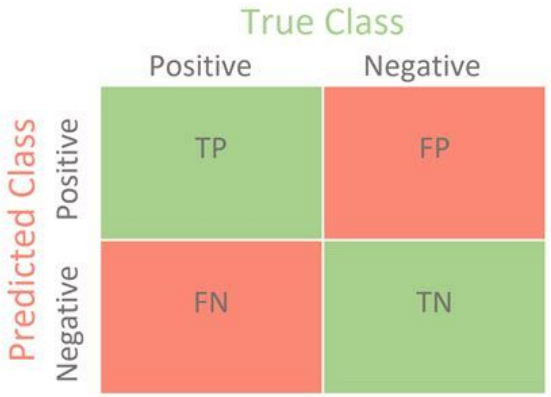
Scikit-learn bible https://scikit-learn.org/stable/auto_examples/index.html#classification

# How to evaluate models

- Performance metrics:
  - Classification
    - Confusion Matrix, Accuracy, AUROC, AUPRC, Precision/Recall, F1, log loss,
  - Regression
    - MAE, MAD, MSE, RMSE, MAPE, MASE, RRSE, R2,
- Practical concerns
  - Time to train
  - Time to predict
  - Time to develop
- Robustness
  - How handles mistakes
- Scalability
  - Ability to handle large amounts of data
- Interpretability
  - Ability to understand why the model predicted any result

# Performance metrics

One of the most important metric is the confusion matrix



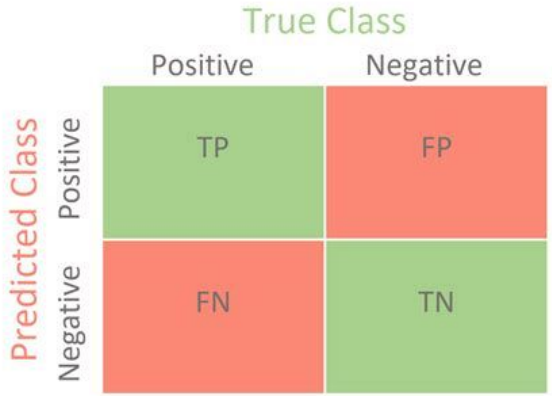

Given this table, there are different formulas that can be used given the use case. VERY IMPORTANT!!!!!!!!!

Example, in the below picture we see an app that detects cancer in an automation fashion. What you think it is the most important value: TP or FN?

# Metrics: Accuracy

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

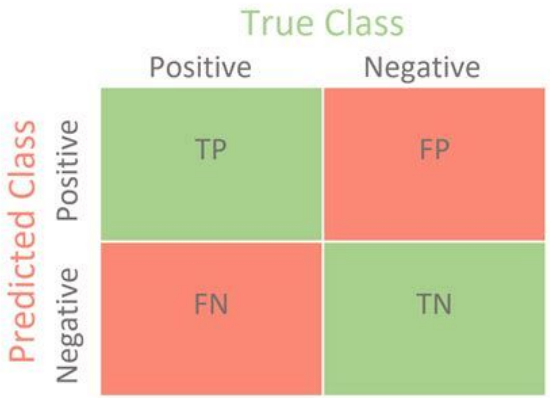
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$



# Metrics: Precision

What proportion of positive identifications was actually correct?

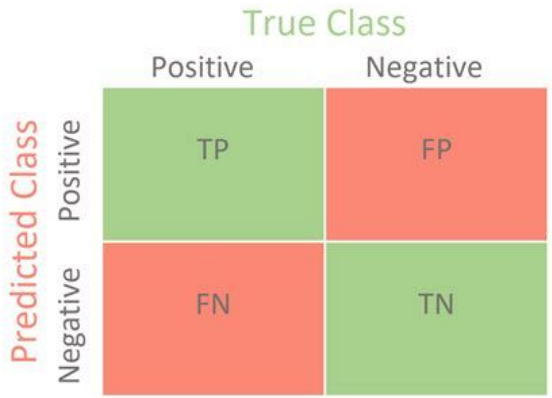
$$\text{Precision} = \frac{TP}{TP + FP}$$



# Metrics: Recall

What proportion of actual positives was identified correctly?

$$\text{Recall} = \frac{TP}{TP + FN}$$



# Precision vs recall

Given the formulas for the precision and recall, do you think it is possible to improve both at same time?

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Ideally what we want is a diagonal matrix, which is not possible. So given the problem we might optimized for a particular metric. This is called “Precision-Recall Tradeoff” . We can tweak a model to increase precision at a cost of a lower recall, or on the other hand increase recall at the cost of lower precision.

# Metrics: F1

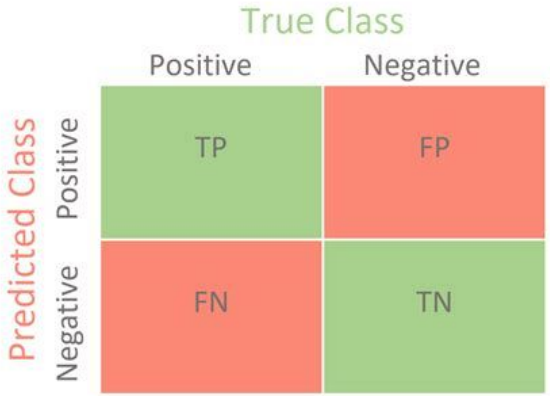
F-measure - combines Precision and Recall and allows for easier comparison of two or more algorithms.

Most common is to use $\beta = 1$

$$F = \frac{(\beta^2+1)PR}{(\beta^2 P + R)}$$

$$F1 = \frac{2\,PR}{P+R}$$

More info:
https://towardsdatascience.com/the-f1-score-bec2bbc38aa6



# Metrics: AUC i.e Area under the curve

To calculate AUC we need ROC curve. An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate (same as recall)
- False Positive Rate

$$TPR = \frac{TP}{TP + FN}$$

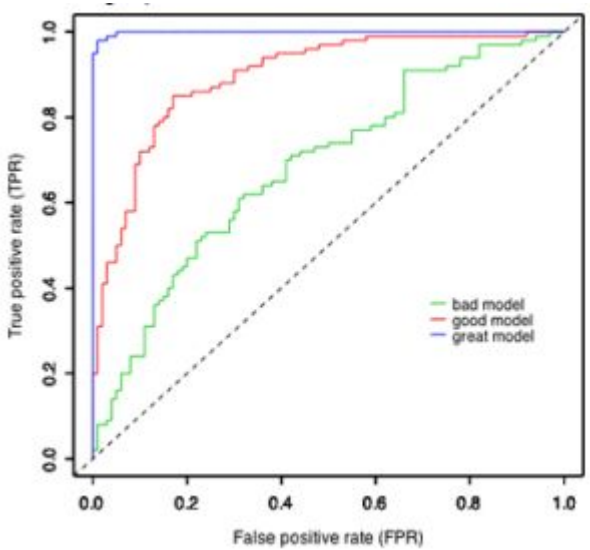
$$FPR = \frac{FP}{FP + TN}$$

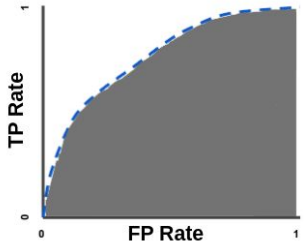
The area is computed by calculating the integral of the curve. As below. Compute using

```
from sklearn.metrics import roc_curve, auc
```

fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=results, pos_label=1)

AUC = auc(fpr, tpr)



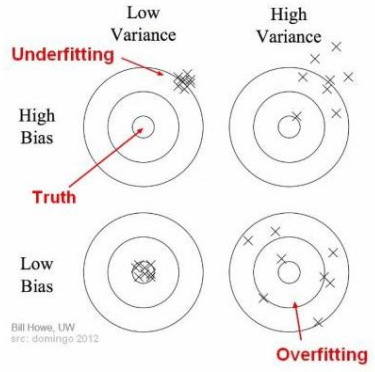



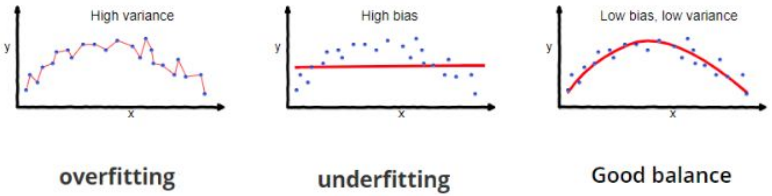
# Understanding the Bias-Variance Tradeoff

This is similar to the precision vs recall trade off. In this case, this compares how close we are from the predicted labels. We may face two different phenomenon:

- Overfitting
  - This happens when either we have a very complex model or we did not use new data to evaluate the model. So, when we start applying the model to new data we end up with huge errors. Which leads to generalization problemas
- Underfitting
  - This happens when the model is to simple for the data, or we have few labels to train and we end up with a huge train error.

Info:
https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229



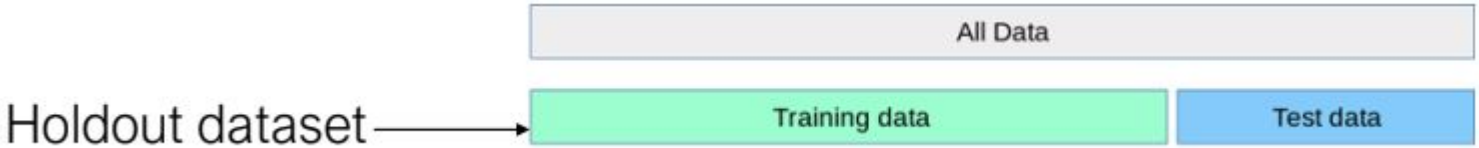



# Techniques to prevent overfitting (not model related)

The most basic technique is to separate the dataset in :

- Training data
- Test data

This is crucial because the test data allow to test the model with unseen data in order to prevent overfitting problems. The split ratio can and must be adjusted depending on the problem. For normal balanced datasets a 70% split is commonly used.



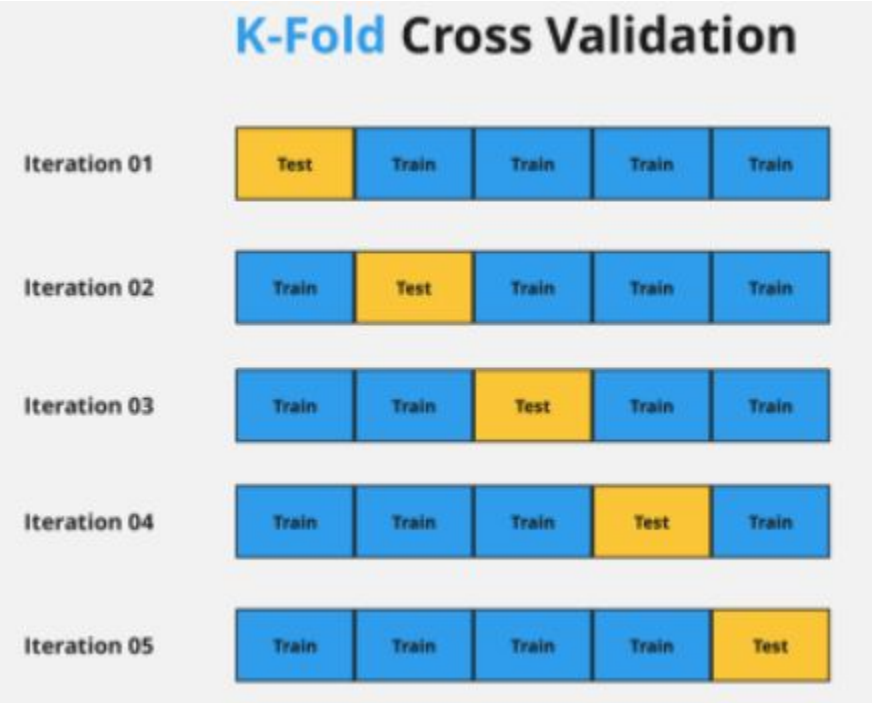

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,  y,  test_size = 0.2, random_state = 1)
```

# Techniques to prevent overfitting (not model related)

K-fold validation is another popular technique that involves splitting the data in several groups. Training the model with each dataset configuration. In the end of the iterations the metric are averaged in order to compute the final result.

The downside of this is that if the model takes a lot of time to train, the k-fold process may take days or weeks to finish.

```
from sklearn.model_selection import KFold
k = 5
kf = KFold(n_splits=k, random_state=None)
kf.split(X)
```



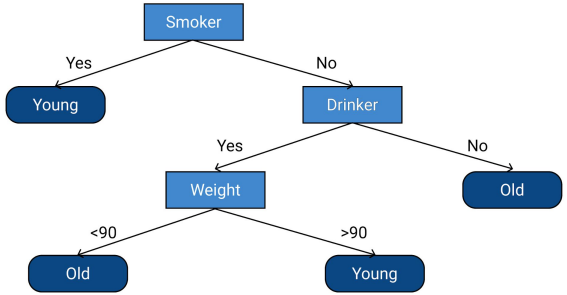
# Decision-tree

Video Time!

https://www.youtube.com/watch?v=ZVR2Way4nwQ&ab_channel=NormalizedNerd

In case you want to know, how the algorithm is implemented from scratch please check

https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python



# Decision tree parameters

| **Parameter** | **Meaning** |
|---|---|
| *criterion* | Metric to decide between splits |
| *max_depth* | This parameter decides the maximum depth of the tree. |
| *min_samples_split* | This parameter provides the minimum number of samples required to split an internal node. |
| max_features | It gives the model the number of features to be considered when looking for the best split. |

# Decision tree Attributes

| **Attributes** | **Meaning** |
|---|---|
| *feature_importances_* | This attribute will return the feature importance. |
| *classes_* | It represents the classes labels i.e. the single output problem, or a list of arrays of class labels i.e. multi-output problem. |
| *max_features_* | It represents the deduced value of max_features parameter. |
| *n_classes_* | It represents the number of classes i.e. the single output problem, or a list of number of classes for every output i.e. multi-output problem. |

# Implementation example

https://bit.ly/3xX4hvU

# How to optimize? - Grid Search

- from sklearn.model_selection import GridSearchCV
- parameters = {'max_depth':MAX_DEPTH, 'min_samples_split':MIN_SAMPLE_SPLIT, "max_features":MAX_FEATURES}
- clf = GridSearchCV(model, parameters, cv=5, scoring="accuracy")
- clf.fit(X, y)
- clf.best_params_
- clf.best_score_

# Let’s apply this in the titanic dataset

1. Load titanic dataset
2. Remove ["Name", "Ticket", "Cabin"] from the first solution
3. EDA
4. Fill missing data
5. Encode categorical variables
6. Split dataset with a 70% 30% ratio
7. Train DT model + evaluation
8. Interpret results
9. Apply to test dataset
10. Generate results to submit to kaggle
11. Check kaggle results
12. Let’s try to improve using k-fold validation
13. Let’s perform GridSearch
14. Find best parameters, best model, show feature importances,
15. Retrain with just the most important features