Introduction to Machine Learning

Elisabetta Ronchieri
INFN CNAF
Big Data Course, 9-12 December 2019, INFN CNAF

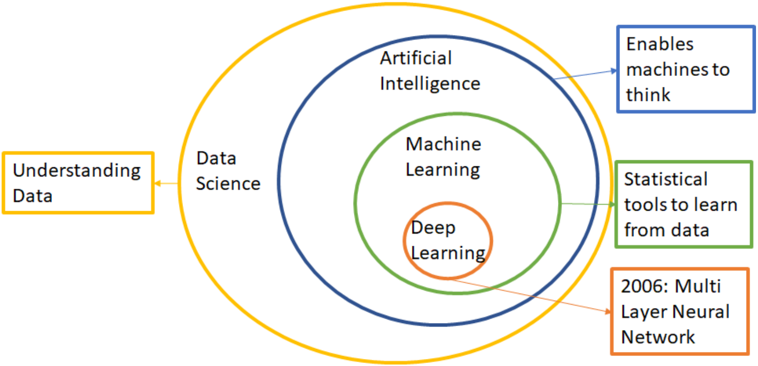
12 December 2019

Outline

- What is Machine Learning (ML)?
- Data Preparation
- Feature Engineering
- Data Modeling
- Performance Measure
- Performance Improvement

What is Machine Learning (ML)?

Where does ML fit in Data Science?



Do we need ML for Big Data?

The sizes and complexities of datasets, we have to deal with, have increased tremendously.

Data represents **a new class of economic assets**, like currency or gold.

This might be true, but it is only true if we are able to extract value from data.

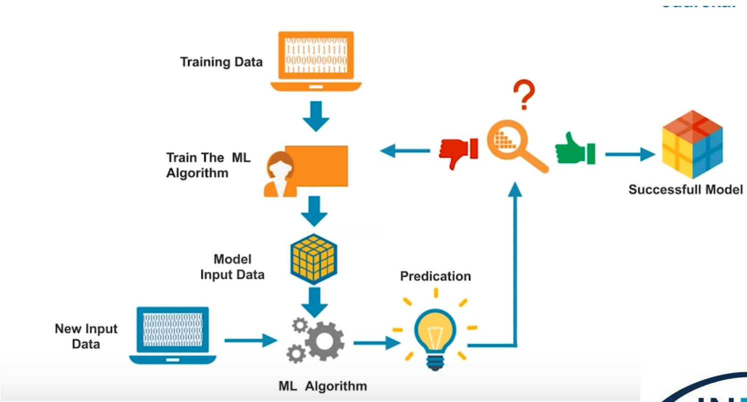
Storing data is not enough.

Extracting value from data requires us to do very deep data analysis.

ML has **a fundamental role in data analysis** to extract value from data.

What is ML?

ML is a set of methods that provides machines the ability to **learn** automatically and **improve from experience** E with respect to some **class of tasks** T and **performance measure** P without being explicitly programmed.

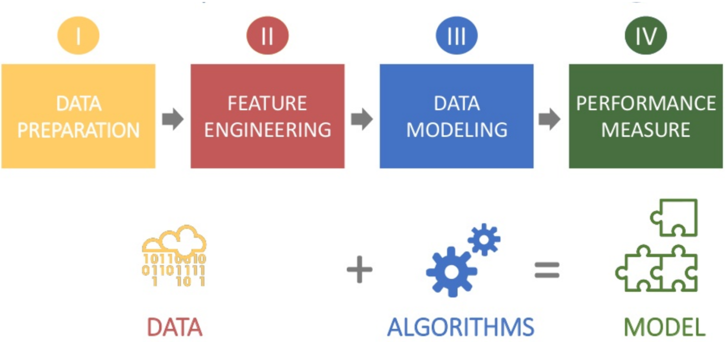


What is a ML model?

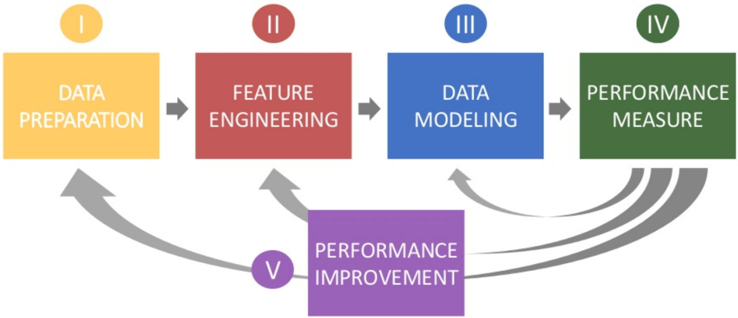
A **ML model** intends to determine **the optimal structure in a dataset** to achieve **an assigned task**.

It results from **learning algorithms** applied on **a training dataset**.

Steps to build a ML model



This is a **highly** iterative process, to repeat until your model reaches **a satisfying performance**.



What tools are available? A subset

Tools	Platform	Cost	Written in Language	Algorithms or Features
Scikit Learn	Linux, Mac OS, Windows	Free	Python, Cython, C, C++	Classification, Regression, Clustering, Preprocessing, Model Selection, Dimensionality reduction, ...
PyTorch	Linux, Mac OS, Windows	Free	Python, C++, CUDA	Autograd Module, Optim Module, NN Module
TensorFlow	Linux, Mac OS, Windows	Free	Python, C++, CUDA	Provides a library for dataflow programming
Weka	Linux, Mac OS, Windows	Free	Java	Data preparation, Classification, Regression, Clustering, Visualization, Association rules mining, ...
RStudio	Linux, Mac OS, Windows	Free	R	Data preparation, Classification, Regression, Clustering, Visualization, Association rules mining , Preprocessing, Model Selection, ...

https://www.softwaretestinghelp.com/machine-learning-tools/

What tools we will use? A subset

Tools	Description
Jupyter	a popular interface for ML
pandas	the reference module to **efficiently manipulate rows of data** in Python
scikitlearn	**the reference module for ML** in Python
numpy, scipy, matplotlib	more convenient Python modules for **data computations** and **data visualization**

https://www.softwaretestinghelp.com/machine-learning-tools/

For the hands on: https://notebooks.cloud.cnaf.infn.it:8888

What you will learn?

Blocks	
I Data Preparation	How can you **import your raw data**? What are the **most common data cleaning methods**?
II Feature Engineering	How do you turn raw data into relevant data? How can you **make the difference between useful and useless data** in a huge dataset?
III Data Modeling	What are the **different types of machine learning algorithms**? Which one should you **choose to build your model**?
IV Performance Measure	What is **the right method to assess the performance** of your model? Which **indicator** should we use?
V Performance Improvement	What are the **reasons why your ML is not performing well**? What are the **most common techniques to improve its performance**

Data Preparation

Preparing data can be done in 3 steps:

1. **query your data**
2. **clean your data** (e.g. deal with **missing values**, remove **outliers**)
3. **format your data**

You can query your data using **pandas**.

E.g. Connect to a local database

```
import pandas as pd

sql_query = """
SELECT * FROM table
LIMIT 100000
"""

df = pd.read_sql(sql_query, connection_to_database)
```

- If you have lots of data, you will find useful to **start working on a subset of your dataset**.
- You will be able to iterate quickly since computations will be fast if there is not too much data.

Query your data

You can query your data using **pandas**.

E.g. Load a local csv file

```python
import pandas as pd

df = pd.read_csv("dataset.csv")
sb.df = df[:100000]
```

- You can import the whole file and work on a subset.

You will obtain a **dataframe** with your raw data.

$$X = \left|\begin{array}{cccccc} & cat1 & cat2 & count1 & count2 & ... \\ \hline 0 & A & B & 0.7234 & 0.7825 & ... \\ 1 & A & C & 0.2689 & 0.8671 & ... \\ 2 & B & B & 0.5721 & 0.3926 & ... \\ 3 & A & B & 0.6345 & 0.4267 & ... \\ ... & & & & & ... \end{array}\right|$$

A **dataframe** is a common data format that will enable you to efficiently work on large volumes of data.

Usually, there are **missing values** in the dataset

- Some **columns will certainly contain missing values**, often as NaN, empty strings, ...
- NaN represents a value that is undefined or unrepresentable
- You **will not be able** to use algorithms with NaN values.
- These cause problems for many ML algorithms.

Need to solve somehow.

Compute **ratio** R_m as the fraction between the number of missing values and the total number of values.

- If R_m is high, you might want to **remove the whole column**.
  - remove all records with NULLs
- If R_m is reasonably low, to **avoid losing data**, you can impute the **mean**, the **median** or the **most frequent value** in place of the missing values.
  - use a default value
  - estimate a replacement value

E.g. Replace missing values with the mean value

```python
from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing_values='NaN', strategy='mean')
```

Some **columns will certainly contain outliers**, i.e. a value that lies at an **abnormal distance** from other values in your sample.

$Players$	$Scores$
$Player_1$	30
$Player_2$	29
$Player_3$	28
$Player_4$	30
$Player_5$	29
$Player_6$	14

- As you can see, all other players scored 28+ except Player$_6$ who scored 14.
- This is showing a mistake or the variance in your data and indicating that Player$_6$ is performing very bad.

Outliers are likely to mislead the ML model, so you will have to **remove them**.

- **Just remove them**: if they are the result of a mistake, then you can remove them.
- **Remove them arbitrarily**: for each of your column, you might guess arbitrarily thresholds above which your data do not make sense.
- **Use robust method**: several methods rely on **robust estimators** (e.g. median) to remove outliers from an analysis.

E.g. Remove outliers

```python
import numpy as np
#Detecting outliers in a Gaussian distributed dataset
from sklearn.covariance import EllipticEnvelope
true_cov = np.array([[.8, .3], [.3, .4]])
X = np.random.RandomState(0).multivariate_normal(mean
    =[0, 0], cov=true_cov, size=500)
cov = EllipticEnvelope(random_state=0).fit(X)
# predict returns 1 for an inlier and -1 for an
    outlier
```

Format data

- You will have to **modify your data** so that they fit constraints of algorithms.
- The **most common transformation** is the **encoding of categorical variables**.

E.g. Sex is a dummy variable, you can add a column for each sex type.

Sex	PrimaryClass		Sex	PrimaryClass		Female	Male	PrimaryClass
Female	3		0	3		1	0	3
Male	2		1	2		0	1	2
Female	2		0	2		1	0	2
Male	1	→	1	1	→	0	1	1
Male	4		1	4		0	1	4
Female	5		0	5		1	0	5

- Strings are converted in numbers.

Format data

E.g. Create dataset

```
import pandas as pd
data = {'sex':['male','female','female','female','male'],
        'pclass':[3,1,3,1,3]}
X = pd.DataFrame(data)
```

E.g. Encode sex variable

```
from sklearn.preprocessing import LabelEncoder
number=LabelEncoder()
X['sex']=number.fit_transform(X['sex'].astype('str'))
```

Feature Engineering

A **feature** is an **individual measurable property** of a **phenomenon** being observed.

The **number of features** is called the **dimension**.

E.g. predict the price of an apartment in Bologna.

Features (individual measurable properties)	Label (phenomenon observed)
size: 45 sqm location: Bologna Floor: 2 Elevator: No #rooms: 2 ...	210 Keuro

What is feature engineering?

Feature engineering is the process of **transforming raw data into relevant features**:

- **informative** providing useful data for your model to correctly predict the label
- **discriminative** helping your model make differences among your training examples
- **non-redundant** avoiding to say the same thing than another feature

The relevant features allow to **improve model performance** on unseen data.

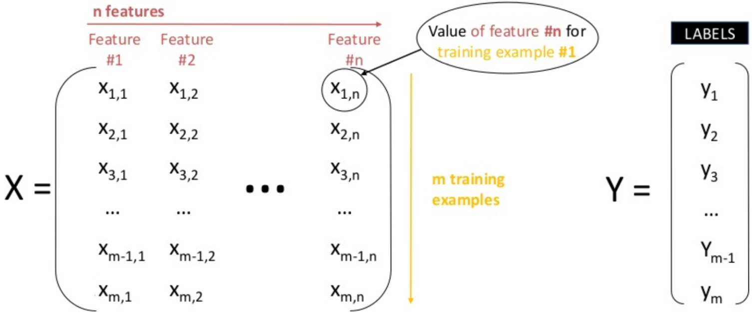
What is feature engineering?

E.g. predict the price of an apartment in Bologna

Is the feature ...	YES	NO
Informative?	size in square meters	the name of your neighbor
Discriminative?	size in square meters	simple or double window?
Non-redundant	size in square meters	size in square inches

- **Size in square meters** is **a good feature to predict** the price of an apartment.

After feature engineering, your dataset will be a **big matrix** of **numerical values**.



What is feature engineering?

Feature engineering usually includes

1. feature construction
2. feature transformation
3. dimension reduction (i.e. feature selection and feature extraction)

Feature construction means turning raw data into **informative features** that **best represent the underlying problem** and that the **algorithm can understand**.

E.g. decompose a Date Time

Same raw data	Different problems	Different features
2019-12-11 16:06:00	Predict how much hungry someone is	Hours elapsed since last meal: 3
2019-12-11 16:06:00	Predict the likelihood of a burglary	Night: 0

In this case, you will need **all the domain expertise**.

Feature Transformation

Feature transformation is the **process of transforming feature** into a new one with a specific function.

Name	Transformation	Reason
scaling	$X_{new} = \frac{X_{old} - \mu}{\sigma}$	**Many algorithms need feature scaling** for faster computations and relevant results (e.g. in dimension reduction)
log	$X_{new} = \log(X_{old})$	

Feature Transformation

E.g. Scaling Transformation

```
from sklearn import preprocessing
X_scaled = preprocessing.scale(X)
```

E.g. Log Transformation

```
import numpy as np
X_log = np.log(X)
```

Dimension reduction

Dimension reduction is the process of **reducing the number of features** used to build the model, with the goal of keeping only **informative, discriminative and non-redundant** features.

The main benefits are:

- Faster computations
- less storage space required
- increased model performance
- data visualization (in case of 2D or 3D dimension)

Feature selection is the process of **selecting the most relevant features** among your existing features.

- We will remove features that are **non informative, non discriminative and redundant**.
- Identifying the most relevant features will help you **get a better general understanding** of the phenomenon you are trying to predict.

Remove non informative features

- **Method**: e.g. Recursive Feature Elimination (RFE)
- **Principle**: We eliminate a single feature **in turn**, run the model each time and note impact on the performance of the model.
- **Note: The lower the impact on model performance, the less informative the feature is and viceversa**.

Remove non discriminative features

- **Method**: e.g. Variance threshold filter
- **Principle**: We remove any feature whose values are close across all the different training examples (i.e. low variance).
- **Note:** A feature that **always says the same thing won't help your model**.

Remove redundant features

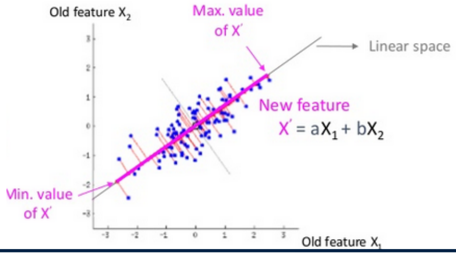
- **Method**: e.g. High correlation filter
- **Principle**: We remove features that are similar or highly correlated with other feature(s).
- **Note:** Your model does **not need** the **same information twice**.
- You can detect **correlated features** computing the Pearson product-moment correlation coefficient matrix.

Feature extraction starts from an initial set of measured data and **automatically** builds **derived features** that are more relevant.

- The new features given by the algorithms are **difficult to interpret**.

- **Method**: e.g. Principal Component Analysis (PCA)
- **Principle**: It makes an orthogonal **projection on a linear space** to determine new features, that are a linear combination of the old ones.
- **Note**: The final features **minimize the loss of information** and **maximize their relevance**.

E.g. Reduction of 2 features into a single one.



Feature engineering

- **Feature engineering** is the process of transforming raw data into **informative, discriminative and non-redundant** features.
- Feature engineering is a very important part to build a performing ML model.

Methods	Description
Feature construction	**Turn raw data into relevant features** that best represent the underlying problem
Feature transformation	Transform features so that they **fit some algorithms constraints**
Dimension reduction	**Eliminate less relevant features** by selecting them or by extracting new ones automatically

Benefits: Faster computations, less storage space required, increased model performance and data visualiza

Data Modeling

You are going to train a model on your data using a **learning algorithm**.

Types of ML: A subset

Supervised learning:

- the model makes predictions or takes decisions based on past data
- the **training set** contains **labels** (output or target)
- e.g. the price of an apartment with well-known characteristics

Unsupervised learning

- the model is able to **identify** patterns, anomalies and relationships in the input data
- the **training set** contains **no labels**, but **only features**

Types of Algorithms: A subset

Supervised learning

- Regression with linear regression
- Cost function and gradient descent
- Classification with random forest

Unsupervised learning

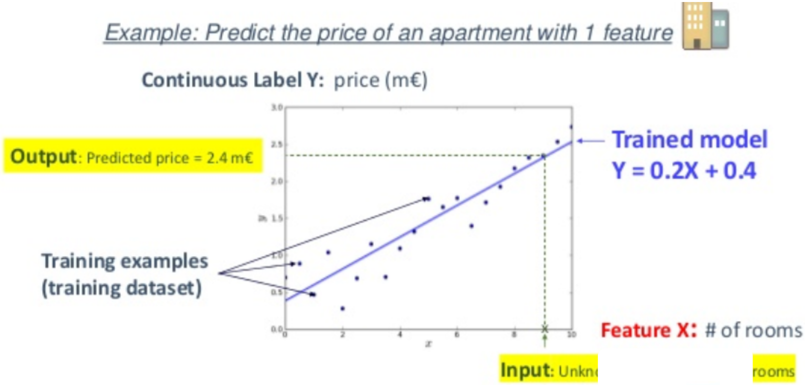
- Clustering with K-means

They are used to build two different kind of models:

1. **Regression**, when the label to predict is a **continuous value**
   - In case of price prediction of an apartment, the value is the price value
2. **Classification**, when the label to predict is a **discrete value**

Regression with Linear Regression

- The output of a linear regression on some data.



Regression with Linear Regression

- By using a Linear Regression it is **assumed** there is a **linear** relationship between your features X and the labels Y.

For all i = 1, ..., m:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \dots + \beta_n x_{i,n} + \varepsilon_i$$

Variable representing **random error (noise)** in the data, assumed to follow a standard normal distribution.

which gives, in matrix form:

$$Y = X\beta + \varepsilon \quad \text{where} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_m \end{pmatrix} \quad \varepsilon = \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \dots \\ \varepsilon_m \end{pmatrix}$$

Cost function and Gradient descent

- The basic Linear Regression method is **called Ordinary Least Squares** and will try to **minimize** the **cost (or loss) function**, representing the difference between your prediction and the true labels.

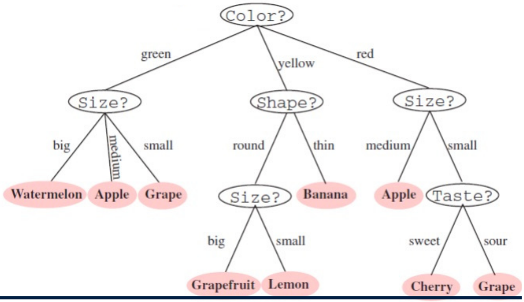
$$J(\beta) = ||Y - X\beta||_2^2 = \sum_{i=0}^{m} (Y_i - X_i\beta)^2$$ (where X_i is the feature vector of the i-th training example, and Y_i the corresponding label)

- Cost functions are often minimized using an algorithm called **Gradient descent**.
- Gradient descent is an **iterative optimization algorithm that will look step by step for** β values where the gradient equals to 0, thus **finding a local minimum of the cost function**.

Decision tree algorithm

E.g. Determine what fruit is an apple on the base of the color.

- color, size, shape and taste are features.
- The selected feature is the one maximizing the **purity** in each output subgroups after the split.
- The decision can rely on two indicators, such as **gini impurity** and **information gain**.
- The **depth of a decision tree** is the length of the longest path from a root to a leaf.

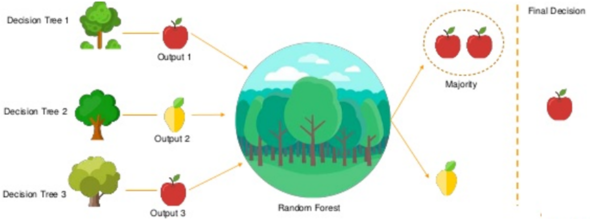


Problems with Decision Tree algorithm

- Imagine you add in your dataset some green lemons and not yet bananas and tomatoes.
- Now we have **many green fruits to classify**.
- Color is not the attribute with the most information gain anymore, so it will not be the splitting attribute in the root node, the structure of the tree is going to change drastically.
- **Decision trees are very sensitive to changes in training examples**.
- If there is a non informative features that happens to provide good information gain, decision tree will wrongly use it as a splitting attribute.
- **Decision trees are very sensitive to changes in the features**.
- **Decision trees are week learners**.

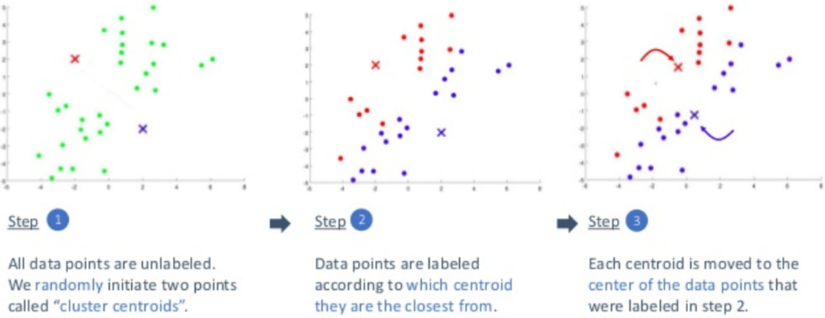
Classification with Random Forest

- **Random Forests** algorithm is using randomness at 2 levels, that is in **data selection** and in **attribute selection**.
- It **relies on the Law of Large Numbers** to discard error in the data.
- This method constructs multiple Decision Trees during training phase and makes decision at the end.

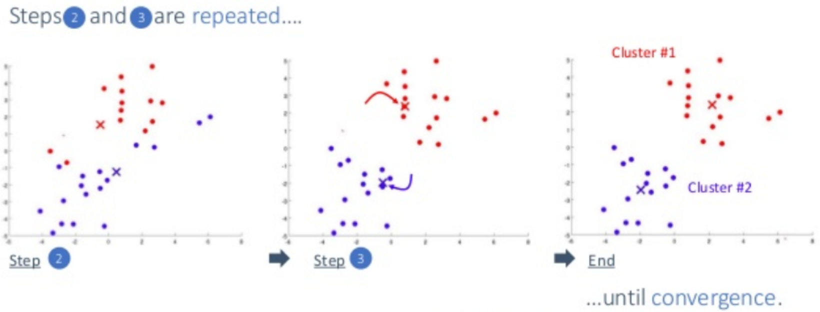


'Random Forest Algorithm - Random Forest Explained, Random Forest In Machine Learning' by Simplilearn

Clustering with K-means



Clustering with K-means



E.g. Linear Regression

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
model = regr.fit(X, y)
```

E.g. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
model = clf.fit(X, y)
```

E.g. K-means

```
from sklearn.cluster import KMeans
# desired number of clusters
kmeans = KMeans(n_clusters=2)
model = kmeans.fit(X, y)
```

What are the best algorithms

- Choosing the right algorithm is one tricky part in ML.
- How your model performs will also depend on your ability to tune them.
- Building a performing ML model depends on the right assumptions about your data and choosing the right learning algorithm for these assumptions.

Performance Measure

Assessing your model performance is a 2-step process:

1. Use your model to **predict the labels** in your dataset

```
Y_predicted = model.predict(X_test)
```

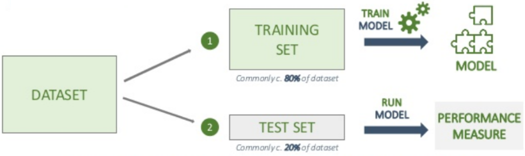
2. Use some indicator to **compare the predicted values with real values**

```
comparison = some_indicator(Y, Y_predicted)
```

1. Predicting your dataset labels
   - training set and test set
   - cross-validation
2. Choosing the right performance indicator
   - Classification
   - Regression

Easy approach

- You never **train** your model and **test** its performance on the same dataset.
  - The **performance measure** would be **deeply biased**.
- The dataset is split in two parts: 80% for the training set and 20% for the test set.



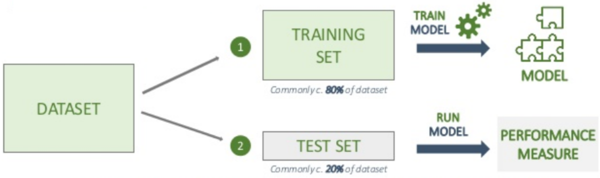
- The test set enables to test our model on **unseen data**.

Training set and test set

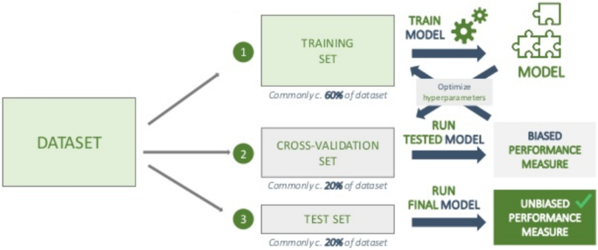
```python
from sklearn.model_selection import train_test_split
# test size is 20%
X_train, X_test, y_train, y_test = train_test_split(X
    , y, test_size=0.2)
```

Easy approach problem

- This method can be used to **test different algorithms** or to **tune hyper parameter values**.
- The **performance measure** will be **biased**, because it **depends on the data in the test set**, therefore you will have to use **cross-validation**.



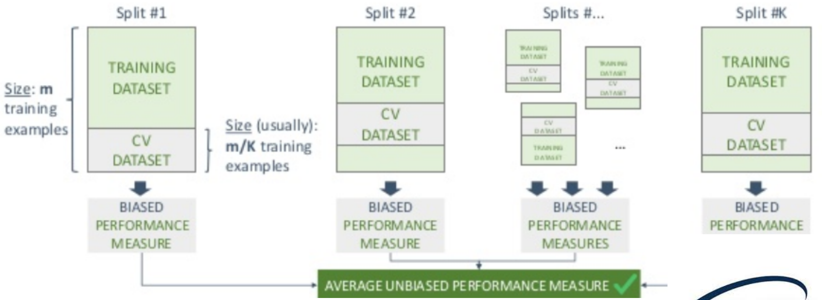
Conservative approach



- You **lose 20% of the data** to train your algorithm.
- With few data, you might prefer **Kfold cross-validation**.

Kfold cross-validation

- **Kfold cross-validation** consists in **repeating the training/cross-validation splitting process** K times to come up with an **average unbiased performance measure**.



E.g. Kfold cross-validation with K=10

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, scoring=
    indicator, cv=10)
```

E.g. Tune hyper parameters with GridSearchCV

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
regr = LinearRegression()
params = {'fit_intercept': [True, False]}
regr = GridSearchCV(regr, params, cv=10)
regr.fit(X, y)
```

Choosing the right indicator: Regression

Examples of two commonly used indicators:

1. MSE is the average of the square of the difference between the true values and the predicted values
2. R^2 is a statistical measure that represents the goodness of fit of a model.

y_i is the **true label** for the *i-th* example in the test set $\hat{y}_i$ is the **predicted label** for the *i-th* example in the test set

$\bar{y}$ is the **average** of the label values in the test set

	Mean squared error	Coefficient of determination (R^2)
Formula	$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$	$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \bar{y})^2}$
Pros	Easy to understand	Absolute value Very roughly, a model with $R^2 > 0.6$ is getting good (1 being the best), $R^2 < 0.6$ is not so good
Cons	Relative value You need the scale of your labels to interpret MSE	Difficult to explain

Choosing the right indicator: Classification

$$\text{Accuracy} = \frac{\text{Number of correctly predicted labels}}{\text{Total number of labels in test set}}$$

Accuracy is **very easy to understand but often too simple** to correctly interpret the performance of your model

Confusion matrix

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

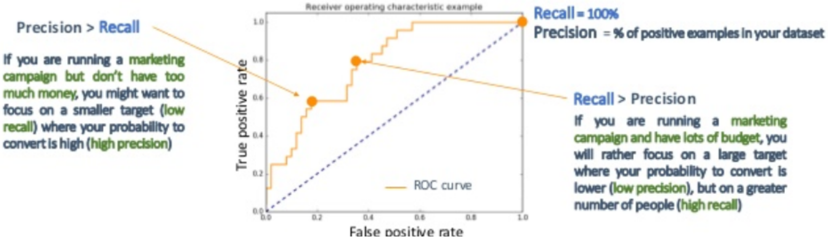
Precision is a % expressing the **precision with which the positive values where recalled** by your model.

		PREDICTED LABELS	
		POSITIVE	NEGATIVE
ACTUAL LABELS	POSITIVE	✓ TRUE POSITIVE (TP)	✗ FALSE NEGATIVE (FN)
	NEGATIVE	✗ FALSE POSITIVE (FP)	✓ TRUE NEGATIVE (TN)

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

Recall is a % expressing the **capacity of your model to recall positive values**.

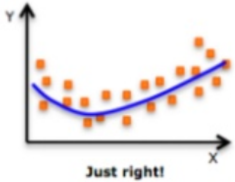
ROC curve visualizes how the TP and FP rates evolve according to different discrimant thresholds of your model.



Performance Improvement

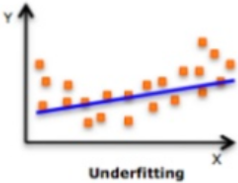
- Reasons for underperformance
    1. underfitting
    2. overfitting
- Solutions to increase performance

- A **performing model** will fit the data in a way that it generalizes well to new inputs.

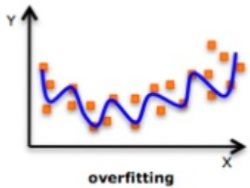


- The model should **reproduce the underlying data structure** but **leave aside random noise** in the data.
- A model would not generalize and not perform correctly for: **underfitting** and **overfitting**.

- Underfitting happens when your **model is too simple** to reproduce the underlying data structure.
- A model is said to have **high bias**.
- Performance on training set and test set are bad.



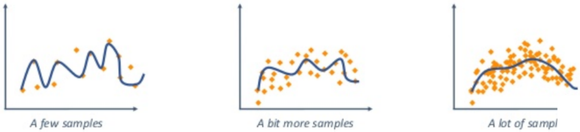
- Overfitting happens when your **model is too complex** to reproduce the underlying data structure.
- The model **captures the random noise** in the data.
- A model is said to have **high variance**.
- Performance on training set is very good but it is bad on test set.



Issue of the model	Act on data	Act on algorithm
Overfitting	More training examples Less Features	Simpler algorithms Regularization Bagging
Underfitting	More Features	More complex algorithms Boosting

To **avoid overfitting** with **more training examples**

- **Different algorithms can perform similarly** for a given problem **as the amount of training examples increases**.
- The more training examples there are, **the more complex** it is for an algorithm to fit the noise in the data.
- The fitted model will be **less sensitive to noise** and will better generalize.



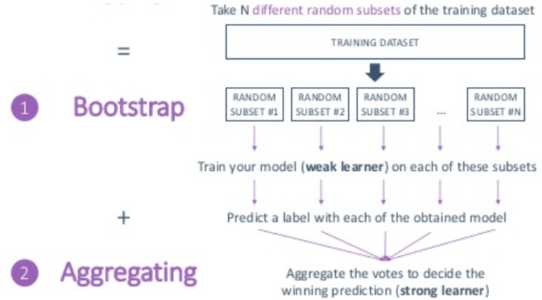
To **avoid overfitting** with **less features**

- Some features might contain **more noise than informative data** for your model.
- This happens when the features are **non-informative or correlated** with other features.
- **Removing them** allows your model not to take noise into account.

To **avoid underfitting** with **more features**

- Your model can be underfitting because you did not give it enough **informative features**.

To **avoid overfitting** with **Bagging algorithm**



Note

You can also apply bagging on your set of features.

Did you notice ?

Random Forest is simply bagging applied on decision tree classifiers (week learners).

To **avoid underfitting** with **Boosting**

Bagging

Aggregating equally the results of weak learners built independently on random samples to create a strong learner.

Strong learner

Weak learner weight, = 1 for all

$$\underbrace{D(x)}_{\text{Strong learner}} = \sum_j \alpha_j d_j(x)$$

Weak learners (usually decision trees) built **independently**

Boosting

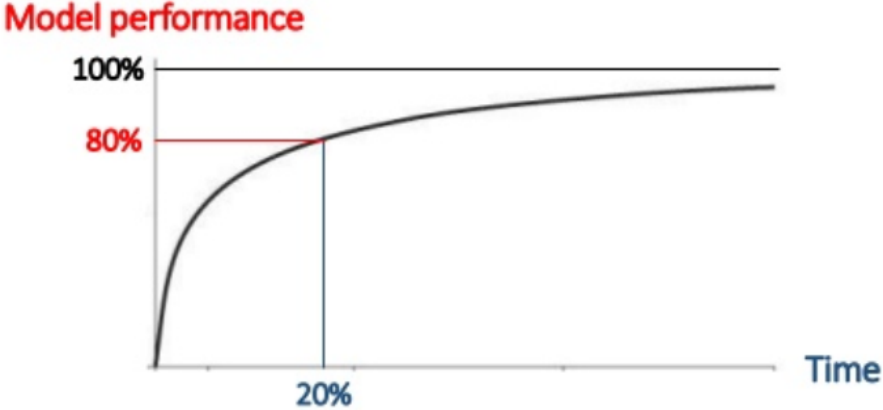
Combining differently the results of weak learners built sequentially on the whole dataset to create a strong learner.

Strong learner

Different weight for each weak learner d_j

$$D(x) = \sum_j \alpha_j d_j(x)$$

Weak learners (usually decision trees) built **sequentially**


Model performance
100%
80%
Time
20%

Building a ML model

Blocks	
I Data Preparation	Detailed the most common data cleaning actions to perform on **raw data**, including **removing outliers** and dealing with **missing values** and **categorical variables**.
II Feature Engineering	Detailed how to **turn raw data into individual measurable properties (features)** that will help your model complete its task. Features have to be as **informative, discriminative and non-redundant** as possible.
III Data Modeling	Detailed **supervised or unsupervised** ML algorithm. Their complexity vary but how they correctly model your data depends on your assumptions.
IV Performance Measure	Detailed **relevant indicators** that you understand and measure on unseen test data.
V Performance Improvement	Detailed **a model can underperform due to underfitting and overfitting**. Many solutions exist, such as **regularization for overfitting** and **boosting for underfitting**

Have fun!

Thanks to Charles Vestur who shared his presentation 'Building a performing Machine Learning model from A to Z'.

Thanks to thank Barbara Martelli, Alessandro Costantini and Doina Cristina Duma for their technical support.

Please contact me if you are interesting in collaborating on unsupervised problems.