Modulo 5

Data Analytics

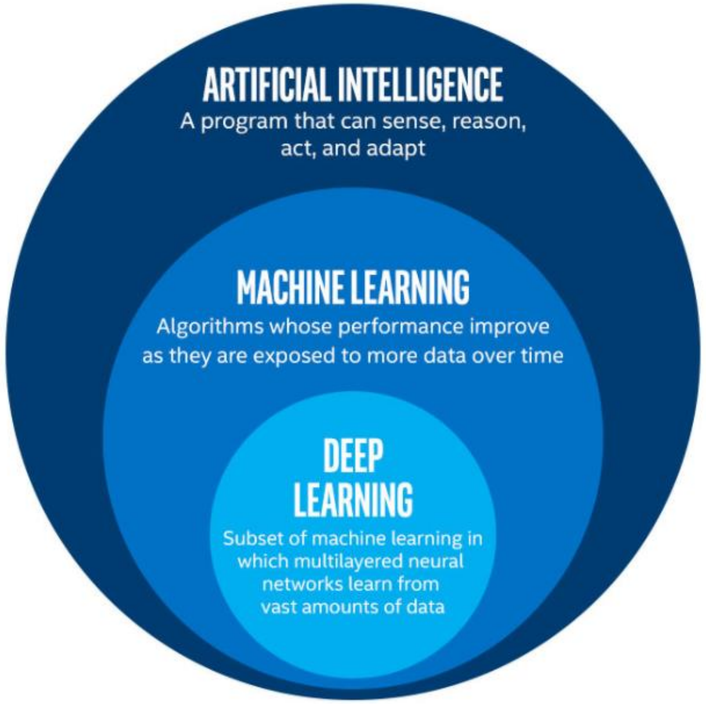
Lesson 6: Building a Machine Learning Model

Alejandro Maté
Juan Carlos Trujillo

Table of contents

WHAT IS MACHINE LEARNING?

- “**Machine learning** (**ML**) is a field of inquiry devoted to understanding and building ***methods that 'learn',*** that is, methods that ***leverage data to improve performance*** on some set of tasks”

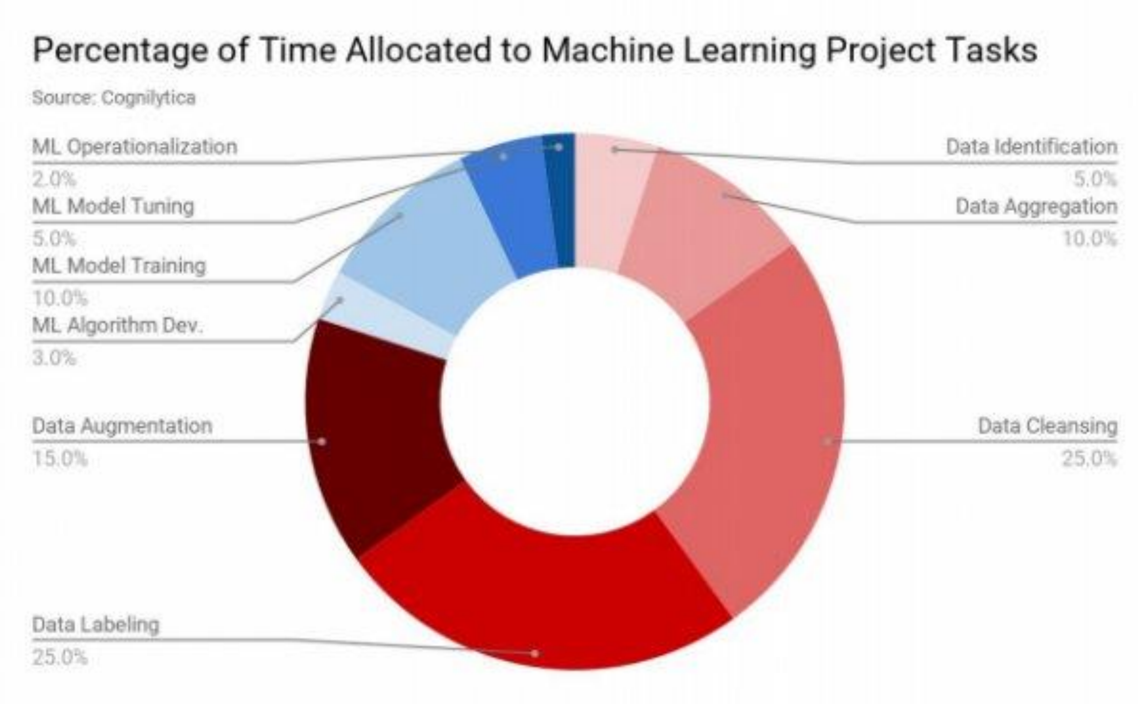


WHAT IS MACHINE LEARNING?

- Algorithms learn, **extracting knowledge** from **data**. This knowledge can take the form of:
    - Rules
    - Patterns
    - Models
- Data is **critical for ML.** Poor data quality → Poor algorithm performance
- ML rarely performs “perfectly”. We have to establish an **acceptation criteria** for the task at hand

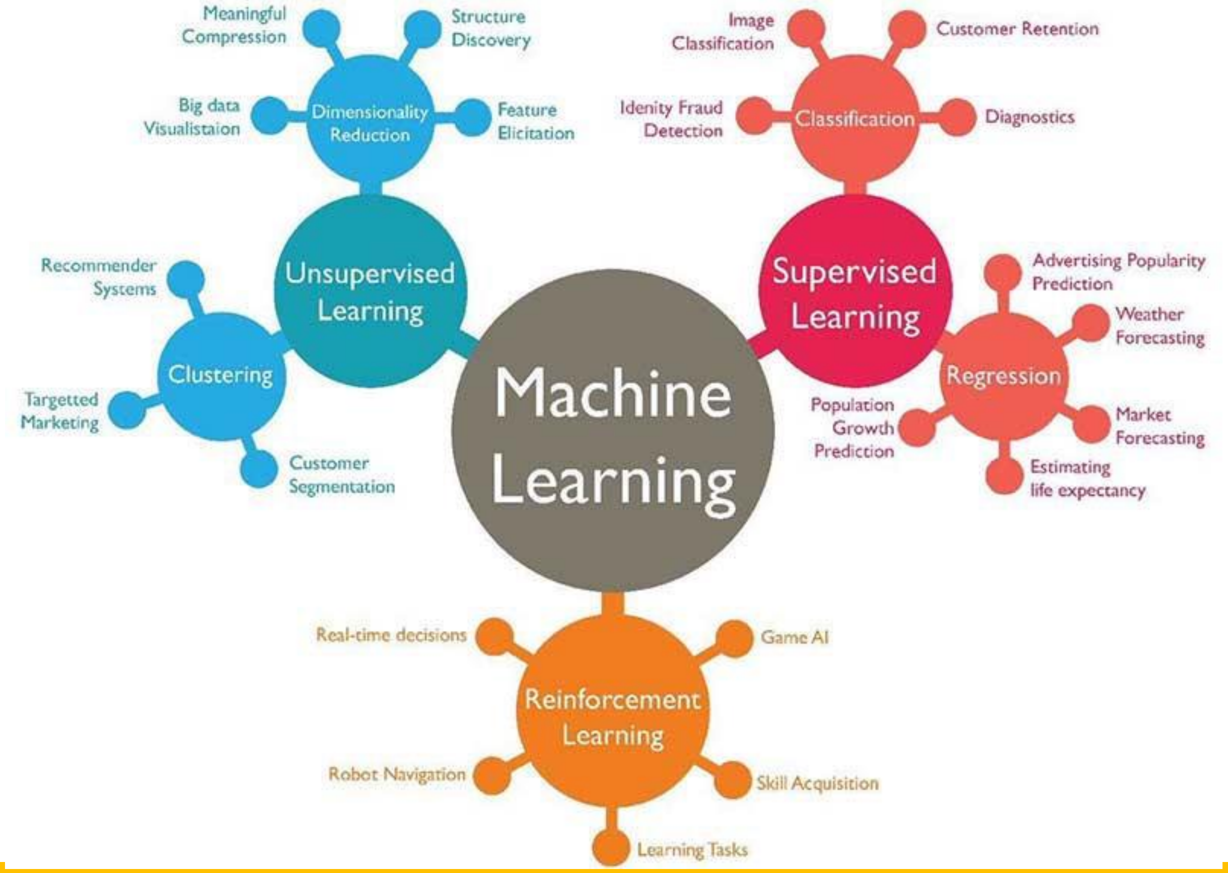
WHAT IS MACHINE LEARNING?

- Most of the time will be spent **gathering and cleansing data**



WHAT IS MACHINE LEARNING?

- Machine learning can be divided into several branches:



WHAT IS MACHINE LEARNING?

- Supervised learning:
  - Learning based on **data labels** with a clear-cut criteria
  - Divided between:
    - **Classification**: Tries to **identify/predict a class** or label associated to an instance
    - **Regression**: Tries to **identify/predict a numeric value** for an instance. A particular type of regression are Time Series, which try to predict a value for a point in time

WHAT IS MACHINE LEARNING?

- Unsupervised learning:
  - Learning based on **data characteristics** for **unlabeled data**
  - Divided between:
    - **Clustering**: Tries to group up similar instances of the data into clusters
    - **Dimensionality Reduction**: Finds relevant features in the data to simplify models and analysis

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - Identify/predict a class for an instance
    - Identifying potential/risky loan customers
    - Classifying illnesses
    - Recognizing objects in an image
  - Can be **binary** (one vs. the rest) or **multiclass**

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - Performance is based on the confusion matrix

Confusion Matrix and ROC Curve

		Predicted Class	
		No	Yes
Observed Class	No	TN	FP
	Yes	FN	TP

TN	True Negative
FP	False Positive
FN	False Negative
TP	True Positive

Model Performance

Accuracy	= (TN+TP)/(TN+FP+FN+TP)
Precision	= TP/(FP+TP)
Sensitivity	= TP/(TP+FN)
Specificity	= TN/(TN+FP)

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - **Accuracy**: most-commonly used measure
    - How many correct predictions were made
  - **Precision**: how many positives identified were really positives
  - **Sensitivity/Recall**: how many positivies were identified (out of the total)
  - **Specifity**: how many negatives were identified (out of the total)

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:

Can a model have very good accuracy and very poor performance?

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - Imagine a COVID-19 classifier that determines whether a person lives or dies. It **has a 97% accuracy**

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - Imagine a COVID-19 classifier that determines whether a person lives or dies. It **has a 97% accuracy**
  - COVID-19 has around **3% mortality rate**
  - Do you think this classifier is good?

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - Error types:
    - Usually, a false positive is called **Error Type I**
    - Meanwhile, a false negative is called **Error Type II**
  - Each error type may have **different consequences depending on the context**
    - Identifying criminals an innocent is condemned (Error Type I)
    - A patient with a tumor is considered healthy (Error Type II)

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - In order to **train** an algorithm we need **to feed it data** (including labels)
  - If we feed the algorithm **all the data** we run the risk of the algorithm "**memorizing**" it
  - We need to **split the data**
    - Training data (Usually 70%)
    - Test data (Usually 30%)

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Classification:
  - A **simple data split** may run into issues:
    - Best/worst case scenario
    - All instances of a class falling into one of the splits
  - To solve these issues a **more robust training** approach is the **K-fold cross-validation** and its variants
    - Stratified cross-validation
    - Monte-Carlo cross-validation

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Regression:
  - Regression models predict a **continuous value** by applying **coeficients** to **predicting variables**
  - Simple regressors create a **linear regression:**

$$y = \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_n p_n$$

  - Useful for estimating the **value of real estate**, **revenue**, **crop production**, etc.

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

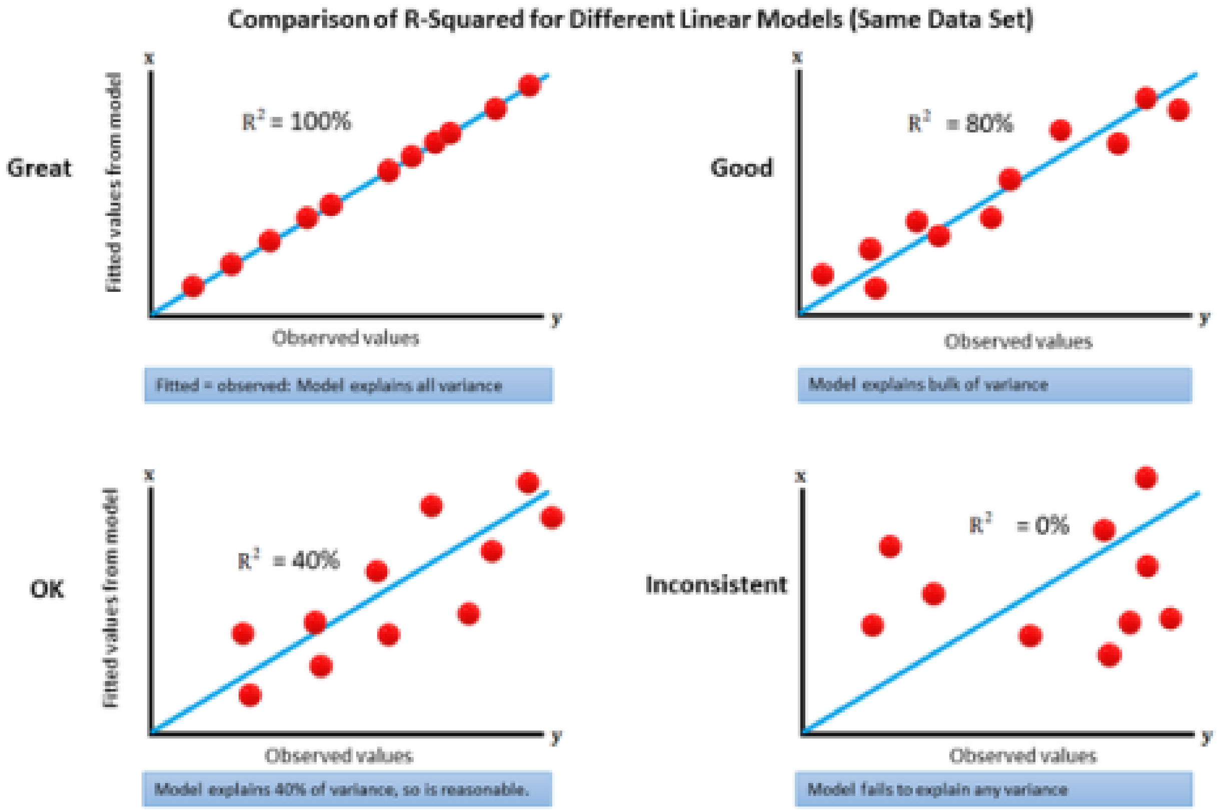
- Regression:
  - Regression has some **key differences** with temporal series:
    - It is an **interpolation** instead of extrapolation
    - It is often more interesting to **identify the values of the coefficients** than the result of the regression
    - Data **does not need to be ordered** in order to perform a regression (as opposed to Time Series)
    - It **does not account** (and therefore has problems with) for **collinearity, seasonality and autoregression**

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Regression:
  - Regression performance is **evaluated using R^2**
    - Representes **how much variability** the regression model **captures**
    - It ranges between [- ∞, 1]. If it is **equal to 1**, then the **prediction is perfect**
    - Only valid for **linear regressions**
  - There are variations of the metric to **penalize adding variables** with little information

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Regression:



APPLICATIONS OF MACHINE LEARNING FOR DATA

- Classification & Regression:
  - Due to the nature of supervised learning analysts **tend to forget**:
    - **Correlation does not imply causation**

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Clustering:
  - Clustering helps **identifying groups** in the data according to different criteria:
    - **Distance** between instances
    - Locating the **center of each group** in the data
    - Analyzing the **density of instances in the neighborhood**
    - …
  - Clustering has **many applications** like customer segmentation, recommendation engines, social network análisis, etc.

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Clustering:
  - Clustering has **no standard metrics** to determine its **performance**
  - However, there are some metrics that measure **different aspects of the result**:
    - **Silhouette coefficient**: Measures how away are samples of each cluster from the rest
    - **Calinski-Harabasz Index**: Measures the dispersion within each cluster and across clusters
    - **Rand Index**: Measures the similarity between two clusters by analyzing each pair of samples

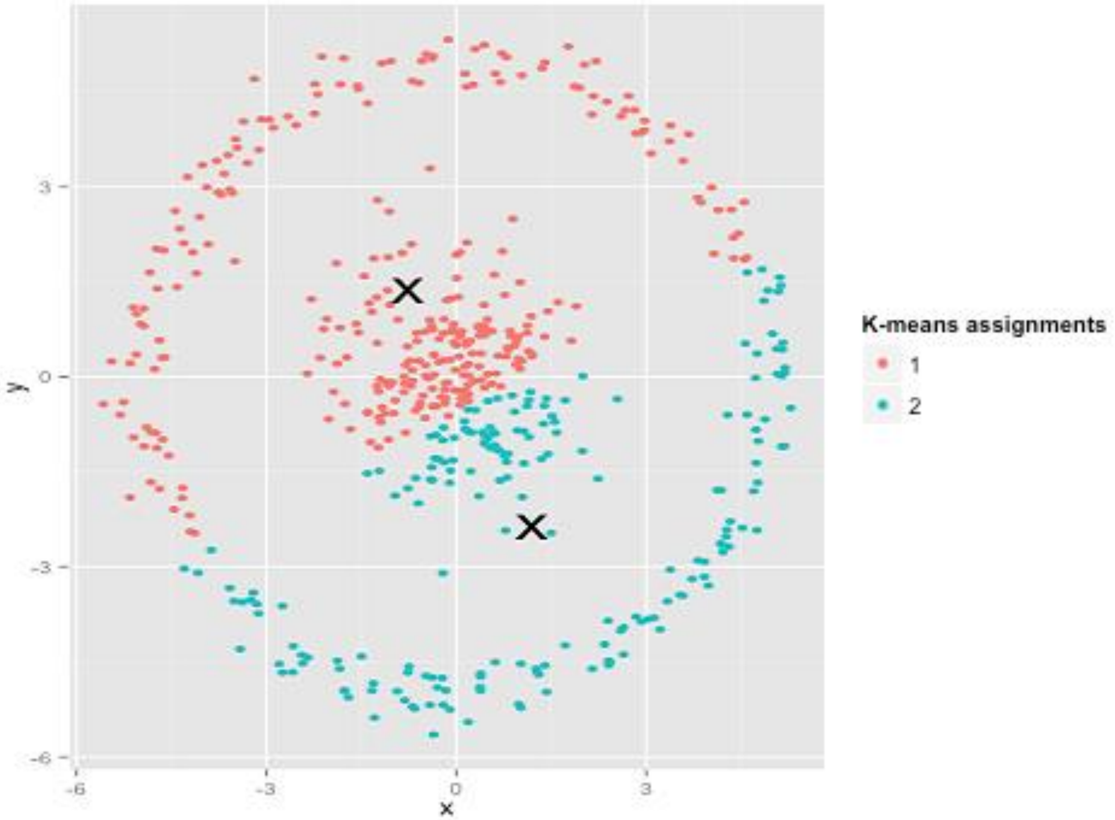
APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Clustering:
  - One of the difficulties of clustering is the fact most algorithms **require a priori number of clusters** or derived (e.g. maximum distance) parameter
  - The process is often done following a **trial and error**:
    - Test with 2,3,4 … n clusters and see the results
  - Even with the **correct number** of clusters, **things may not go as expected**

APPLICATIONS OF MACHINE LEARNING FOR DATA ANALYTICS

- Clustering:

Source: https://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means



MACHINE LEARNING IN JUPYTER Classroom guided exercise

- Using ML in Jupyter requires making use of the Scikit-Learn library
- We will start by importing the Titanic data to work with

```python
import io
import pandas as pd
import numpy as np

df_ejer1 = pd.read_csv('C:\Data\Asignaturas\Malaga\Archivos demo\Titanic.csv', sep=";")
```

```python
df_ejer1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- Afterwards we will **load the necessary libraries** for the ML algorithms

```
# Cargamos las librerias generales de sklearn
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn import metrics
```

- Many algorithms require that data is in **numeric format only** so we will transform the data before training the algorithms

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- We will **remove missing values** using .notna() functions
- Afterwards we will **remove columns** that should not be used by the algorithms in order to avoid errors

```python
# Preparamos los datos para el SVM
from sklearn import svm

classifierSVM = svm.SVC(kernel="rbf",C=1)

clfSVM = make_pipeline(StandardScaler(), classifierSVM)

# Quitamos los valores NA en Age y Embarked
X = df_ejer1[df_ejer1["Age"].notna()]
X = X[X["Embarked"].notna()]
# Recuperamos la etiqueta de clase a predecir
Y = X["Survived"]
#Reemplazamos los valores de sexo y embarque por valores numéricos
X = X.replace(to_replace="male",value=0)
X = X.replace(to_replace="female", value=1)
X = X.replace(to_replace="S", value=0)
X = X.replace(to_replace="Q", value=1)
X = X.replace(to_replace="C", value=2)
# Guardamos el vector para la regresion posterior
X = X.drop(["Name","Ticket","Cabin"],axis=1)
XReg = X
# Quitamos la etiqueta de clase
X = X.drop(["Survived"],axis=1)
```

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- Once data is ready, we **split the data** into training and test and we train our algorithm:

```
# Una vez tenemos los datos entrenamos el clasificador. Utilizamos la opción .values para pasar únicamente los valores
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
clfSVM.fit(X_train.values,Y_train.values)
```

- We can test **how well our algorithm performed** using the score() fuction:

```
# Comprobamos el accuracy del clasificador
clfSVM.score(X_test.values, Y_test.values)
```

- While **poor performance** can be improved **modifying the parameters**, most often we will have to **manipulate the data**

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- We can use our algorithm to **predict** the values of **new samples**:

```
result= clfSVM.predict([X_test.iloc[0]])
result[0]
```

- According to the *“No Free Lunch”* theorem, there is **no best algorithm for every case**, so we will have to compare multiple algorithms
- We can train a Decision Tree to compare with our SVM

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- The training steps are very similar

```
# Ahora vamos a entrenar un árbol de decisión
# Para poder pintarlo después deberemos cambiar los valores numéricos de la etiqueta de clase por cadenas
from sklearn import tree
YCat = Y.replace(to_replace=0, value="Dies")
YCat = YCat.replace(to_replace=1, value="Survives")
XTree_train, XTree_test, YTree_train, YTree_test = train_test_split(X,YCat, test_size=0.33, random_state=42)
classifierTree = tree.DecisionTreeClassifier(max_depth=3, criterion="entropy")
classifierTree.fit(XTree_train.values, YTree_train.values)
```

- Only differences are that we will use **strings for classes** and that **parameters change** from one algorithm to another

MACHINE LEARNING IN JUPYTER Classroom guided exercise

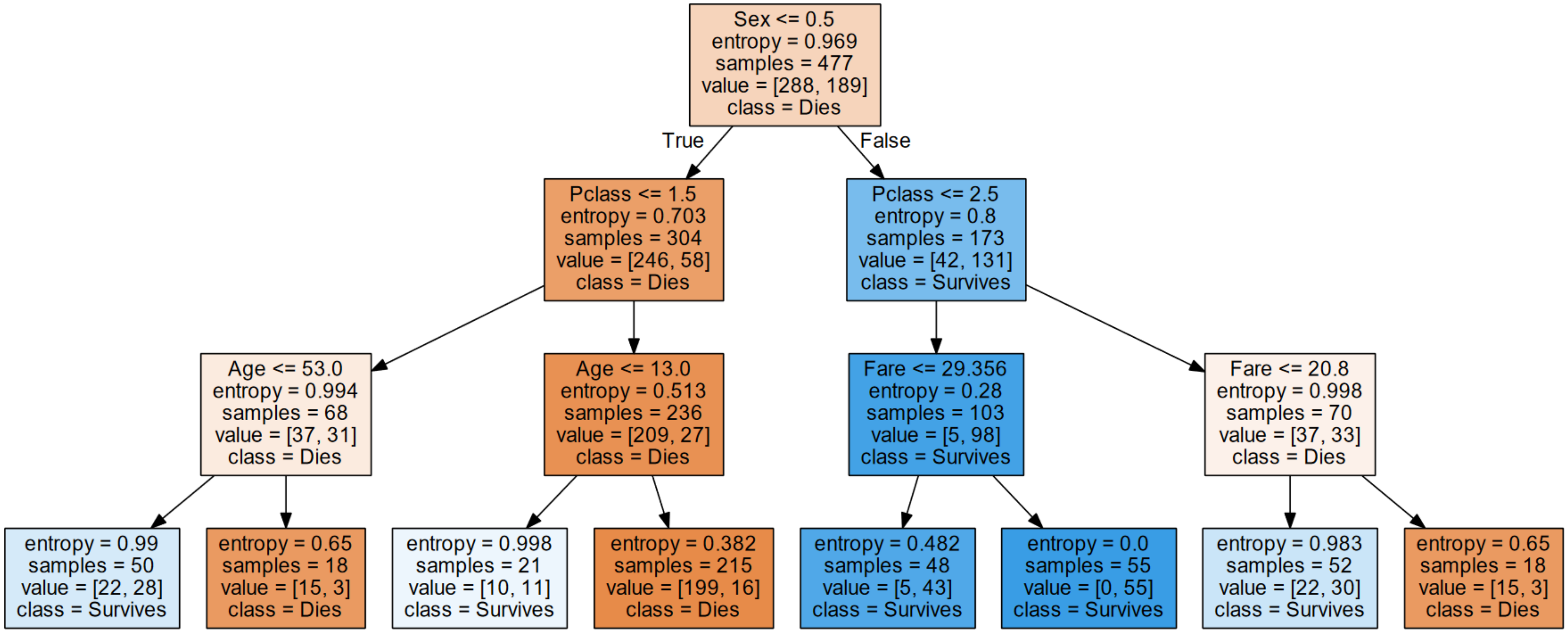
- Decision Trees have the advantage that **can be plotted**, so we can **understand** what is **the logic** behind them

```python
tree.plot_tree(classifierTree, filled=True)
```

```python
# Una vez hemos obtenido el árbol lo pintamos usando graphviz
import graphviz

dot_data = tree.export_graphviz(classifierTree, feature_names = X.columns, \
                                class_names = classifierTree.classes_, filled=True, out_file=None)
graph = graphviz.Source(dot_data)
```

MACHINE LEARNING IN JUPYTER Classroom guided exercise



MACHINE LEARNING IN JUPYTER Classroom guided exercise

- After seeing how classification works let’s try a regression
- We will try to see if we can explain **how much each passenger paid**
- First we will start training a Decision Tree for **regression**

```
# Vamos ahora a entrenar un regresor. Podemos crear uno mediante árboles de decisión
regressionTree = tree.DecisionTreeRegressor(max_depth=2)
YReg = X["Fare"]
XReg = XReg.drop("Fare",axis=1)
regressionTree.fit(XReg.values,YReg.values)
```

- We can check how well the regressor captures the **variance** using .score()

```
regressionTree.score(XReg.values,YReg.values)
```

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- As we mentioned before, the interesting part of regression is **checking the coefficients**
- We can check them by **joining the tree importance coefficients** with the **data columns**

```
regTreeCoef = pd.DataFrame(regressionTree.feature_importances_,XReg.columns, columns=['Importance'])
regTreeCoef
```

	Importance
PassengerId	0.000000
Survived	0.000000
Pclass	0.808677
Sex	0.000000
Age	0.000000
SibSp	0.021828
Parch	0.169494
Embarked	0.000000

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- As before we can **compare multiple algorithms**. The most common one is **Linear Regression**

```python
# Vamos ahora a entrenar un regresor lineal de manera muy similar
from sklearn.linear_model import LinearRegression

linearReg = LinearRegression().fit(XReg.values,YReg.values)
```

```python
# Y comparamos resultados
linearReg.score(XReg.values, YReg.values)
```

- If we check the coefficients we will see certain **similarities**

	Coefficients
PassengerId	0.000403
Survived	2.400731
Pclass	-33.495130
Sex	1.870147
Age	-0.143708
SibSp	5.778587
Parch	10.430830
Embarked	10.528990

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- It is important to note that, since we are calculating a numeric value, **normalization affects the coefficient values**

```
# Comparamos los resultados con normalización previa:
normalizedLinearReg=make_pipeline(StandardScaler(), LinearRegression())
linReg = normalizedLinearReg.fit(XReg.values,YReg.values)
linReg.score(XReg.values, YReg.values)
```

- Checking again the **coefficients** we will notice certain **changes**

	Coefficients
PassengerId	0.104200
Survived	1.178264
Pclass	-28.010853
Sex	0.899694
Age	-2.081290
SibSp	5.374307
Parch	8.903562
Embarked	8.196717

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- Finally, let's apply **Clustering** to our data
- We will use the most common Clustering algorithm: Kmeans

```
# Para terminar vamos a intentar hacer clustering de los datos
from sklearn.cluster import KMeans

numClusters = 5
XRegClusters = XReg.drop('Survived', axis=1)
```

```
kMeansClusters = KMeans(n_clusters=numClusters, random_state=42, n_init=5).fit(XRegClusters.values)
kMeansClusters.labels_
```

- We can test with different **number of clusters**
- In all cases, the result will be a list of **cluster tags** for the data passed

MACHINE LEARNING IN JUPYTER Classroom guided exercise

- Since cluster tags are **difficult to interpret**, we can use **visualizations** to better understand the results
- First, we will **join the input data with the cluster tags**

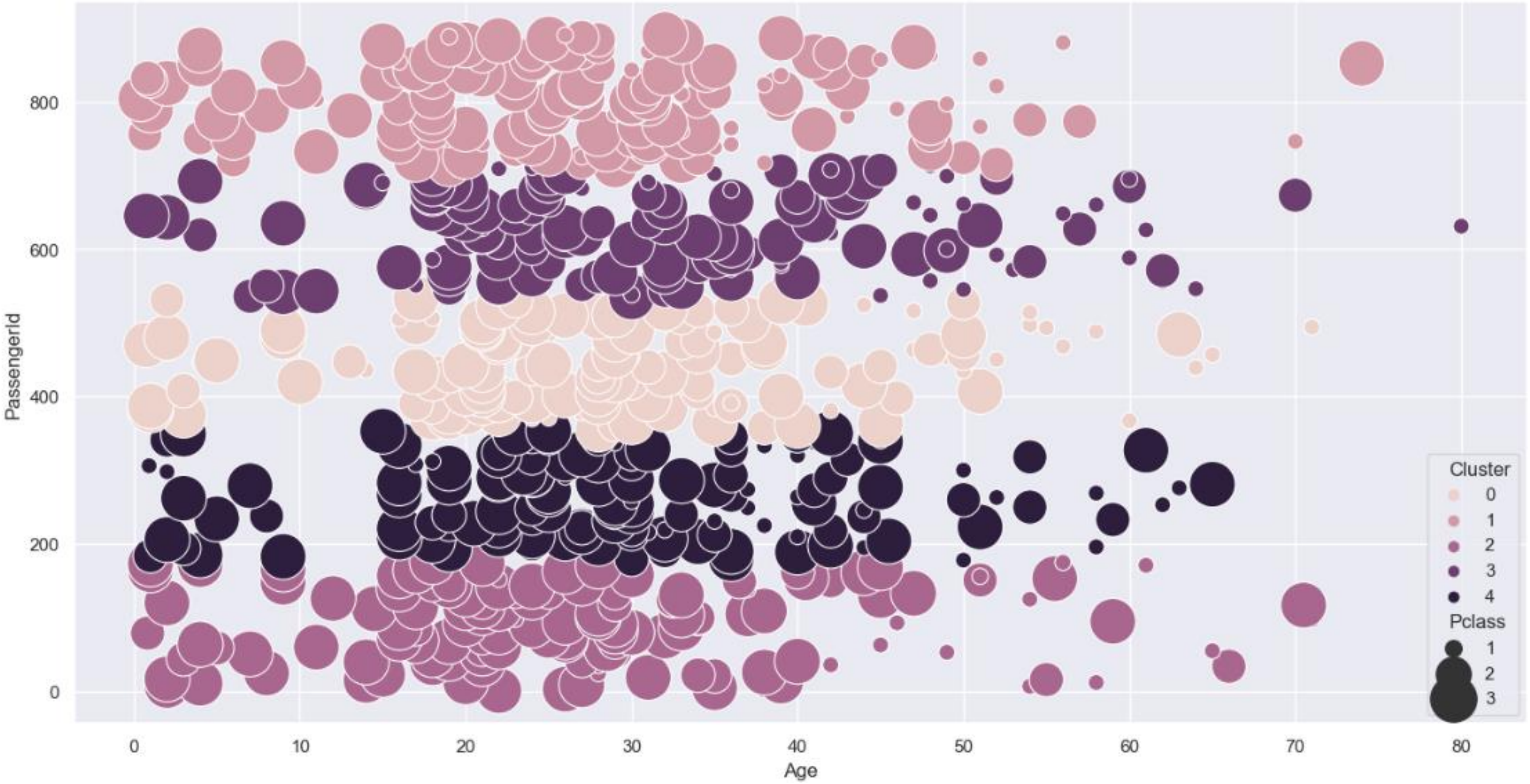
```
# Para poder interpretar los datos unimos los clusters con sus respectivos datos
DFClusters=pd.DataFrame(kMeansClusters.labels_,columns=['Cluster'])
DFClusters.index = XRegClusters.index
kMeansResult = XRegClusters.join(DFClusters)
```

- Then we will **plot it**

```
# Para comprobar la asignación de clusters de forma visual podemos utilizar las visualizaciones que vimos anteriormente
# y analizar como se correlacionan las distintas variables e instancias con los clusters
import seaborn as sns

sns.set(rc={"figure.figsize":(16, 8)})
bubbleplot = sns.scatterplot(data=kMeansResult,x='Age',y='PassengerId', \
                             size='Pclass',hue='Cluster', sizes=(100,800))
```

MACHINE LEARNING IN JUPYTER Classroom guided exercise



MACHINE LEARNING IN JUPYTER Classroom guided exercise

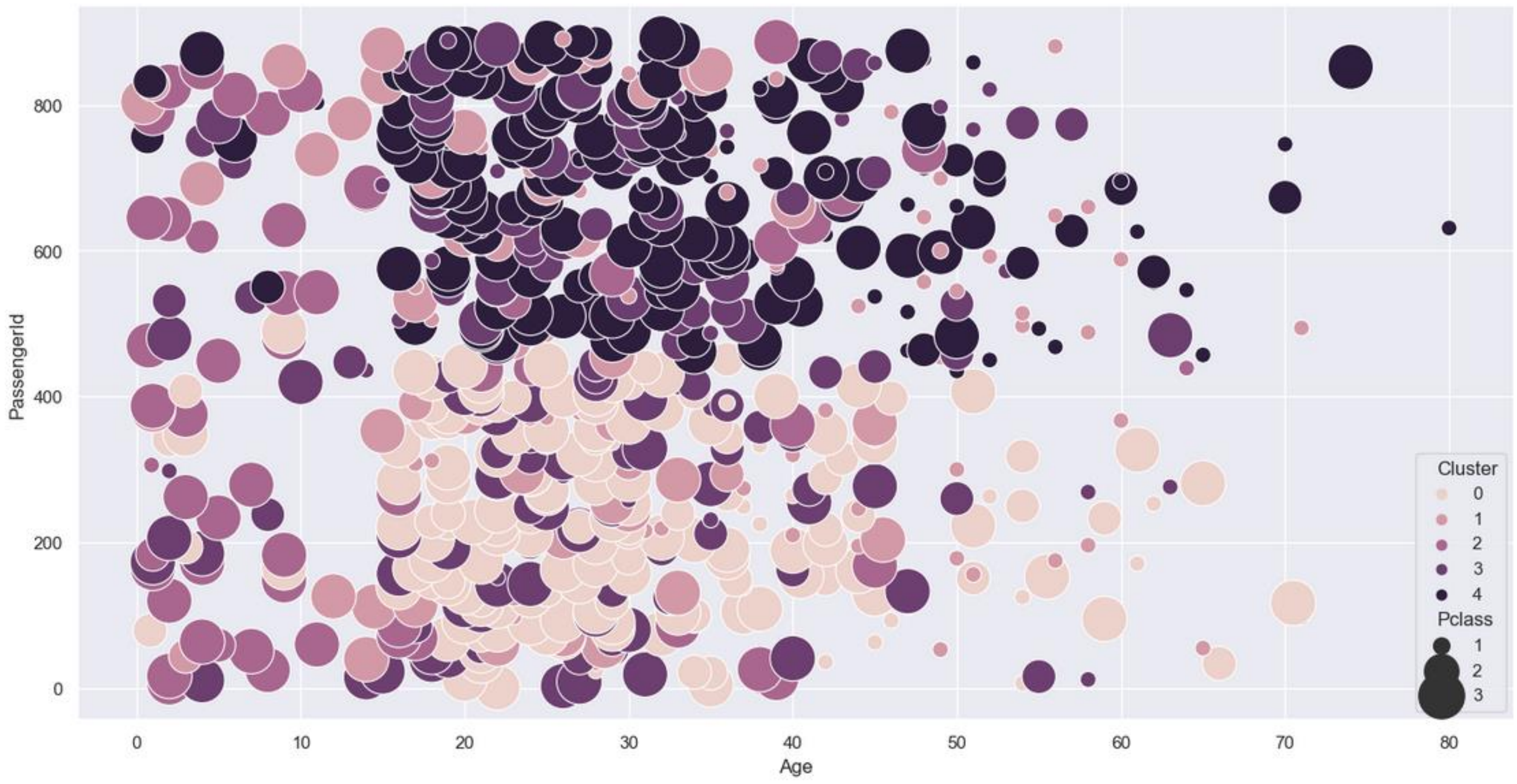
- It is important to note that, since **KMeans calculates** clusters according to **distance**, **scaling** has an **effect on cluster tags**

```
# Vamos a probar ahora escalando los datos de entrada

scaledKMeans = make_pipeline(StandardScaler(), KMeans(n_clusters=numClusters, random_state=42, n_init=5))
scaledKMeans.fit(XRegClusters.values)

DFScaledClusters=pd.DataFrame(scaledKMeans[1].labels_,columns=['Cluster'])
DFScaledClusters.index = XRegClusters.index
scaledKMeansResult = XRegClusters.join(DFScaledClusters)
scaledKMeansResult
```

MACHINE LEARNING IN JUPYTER Classroom guided exercise



MACHINE LEARNING IN JUPYTER Classroom guided exercise

- Since identifying **whether the clustering is relevant or not can be difficult** (Did we choose the right dimensions to plot?) we can **calculate coefficients** such as Silhouette:

```
from sklearn.metrics import silhouette_samples, silhouette_score
silhouette_avg = silhouette_score(XReg, scaledKMeans[1].labels_)
print("For ",numClusters," clusters, the average Silhouette score is :",silhouette_avg)
```

```
For  5  clusters, the average Silhouette score is : -0.008925394982467418
```

- A Silhouette coefficient is a value in range [-1,1]
  - -1 represents completely **overlapping** clusters
  - 1 represents perfectly **separated** clusters