Taller: Manipulación de datos y clasificación

ENCUENTRO NACIONAL DE COMPUTACIÓN 2020

Daniela Moctezuma

dmoctezuma@centrogeo.edu.mx (mailto:dmoctezuma@centrogeo.edu.mx)

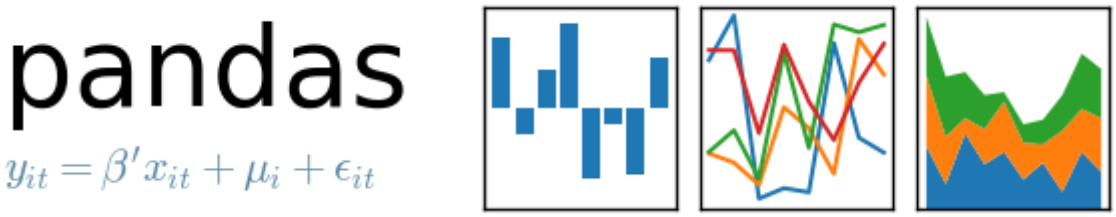
Contenido

- Manipulación de datos y clasificación
  - Pandas
  - Sklearn

instalar conda

https://docs.conda.io/en/latest/miniconda.html (https://docs.conda.io/en/latest/miniconda.html)

Pandas



Python Data Analysis Library

Pandas es una librería de open source que permite el uso fácil de estructuras de datos y herramientas de análisis utilizando el lenguaje de programación Python.

In [1]:
```
# Para usarlo primero hay que importarlo
import pandas as pd
```

In [3]:
```
#Para abrir un archivo
datos = pd.read_csv("titanic.csv")
```

In [4]:
```
datos.head()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

Scikit learn

In [11]:
```
import sklearn
```

Machine learning...

- Hasta ahora solo hemos visualizado datos, y hemos hecho un sencillo análisis exploratorio. El siguiente paso, que vamos a ejemplificar, además de datos "de juguete", en el casos del

Titanic, esto es predecir basándonos en los datos aprendidos. Para eso, necesitamos ver algunas técnicas clásicas de aprendizaje computacional como son: Regresión lineal, Árboles de decisión y Random Forest, Clustering, principalmente.

¿Qué es Machine Learning?

Existen muchas definiciones, una de las más antiguas es de Arthur Samuel (pionerio de la IA), que describe al Machine Learning como:

"Campo de estudio que porporciona a las computadoras la abilidad de aprender algo sin estar explícitamente programadas para ello"

Entiendiendo Machine Learning

En Machine learning siempre tengo ** Features ** y ** Labels **

Por ejemplo:

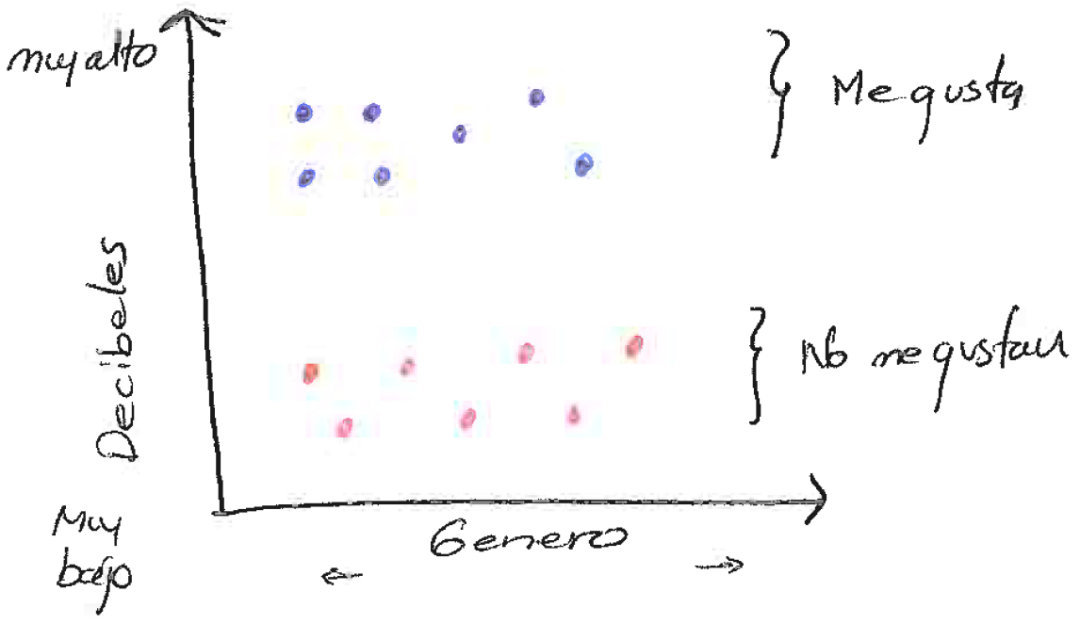
Features

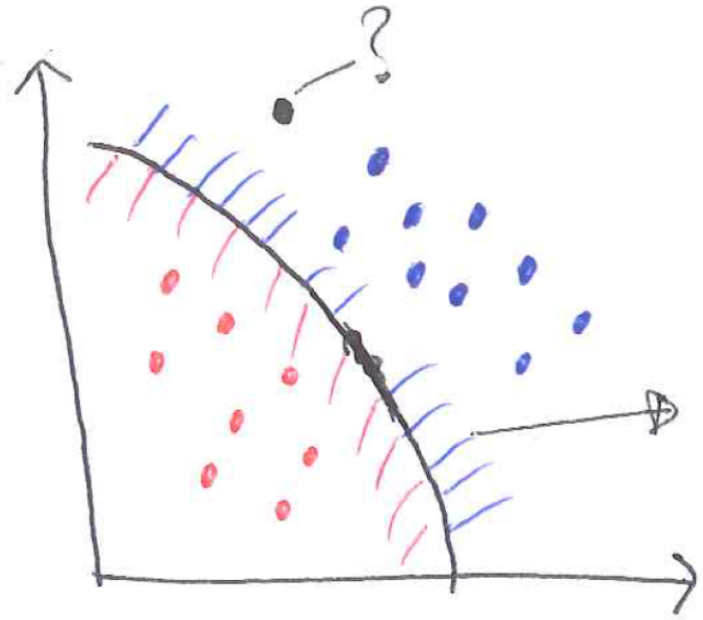
Decibeles
Género
Frecuencia
Amplitud
Artista
.
.
.

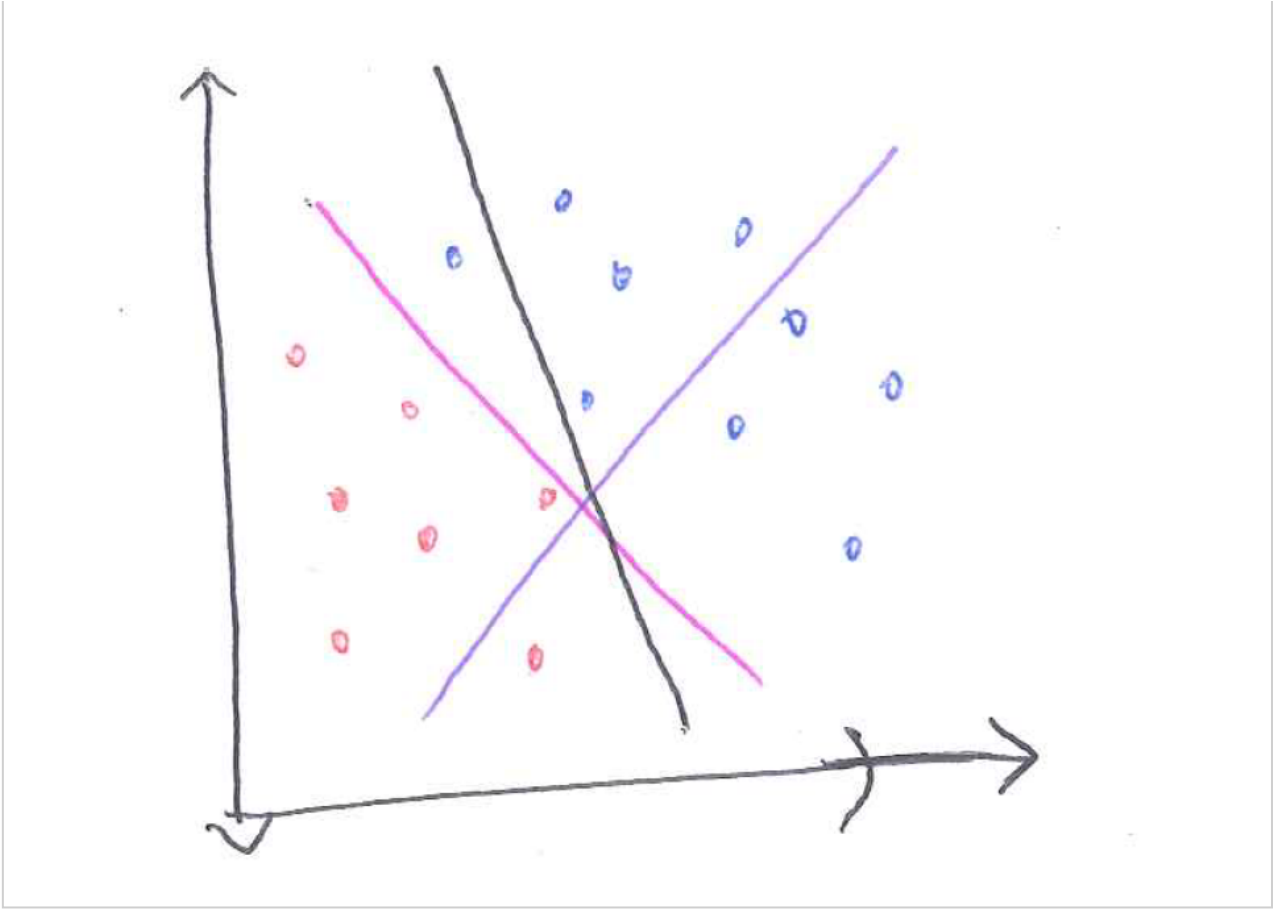
Labels

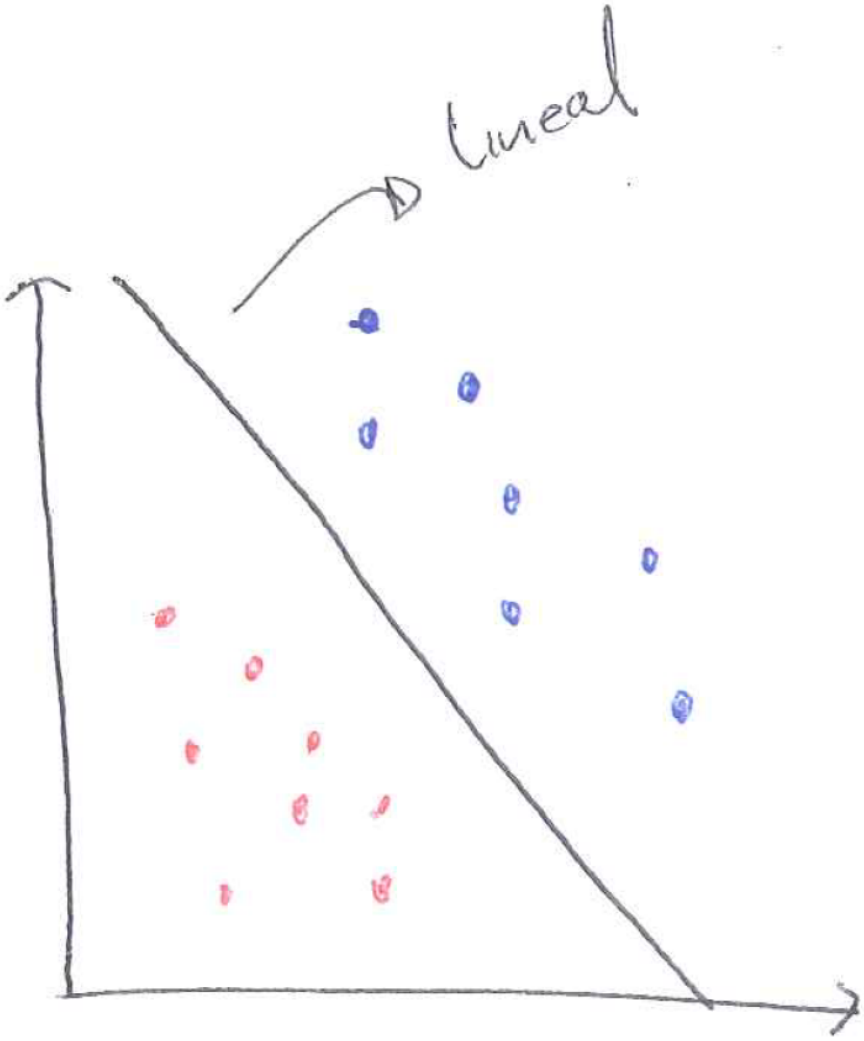
Te gusta

No te gusta


muy alto
Decibeles
Muy bajo
Genero
Me gusta
No me gustan


?





Tipos de aprendizaje

- Aprendizaje supervisado
    - Árboles de decisión
    - Random forest
    - SVM
    - ANN
- Aprendizaje no supervisado
    - Clustering (K-means)

Aprendizaje Supervisado

- Se tiene un conjunto $T = \{(x_i, y_i)\}$
- x son las variables independientes, observaciones, features, ...

- y es la variable dependiente
- objetivo $f(x) \approx y$

Medidas de rendimiento

- Recall (r): Fracción de instancias relevantes recolectadas
- Precision (p): fracción de instancias recolectadas relevantes para la consulta
- Score-$F_1 = 2\frac{p \cdot r}{p+r}$
- Accuracy

Training, validation and test sets

- Como su nombre lo indica en la parte de traning se entrena al clasificador, en la parte de validation se obtienen los parámetros adecuados para el clasificador y en test se evalúa el rendimiento real del clasificador (con datos nuevos diferentes a los utilizados en las dos fases anteriores).

Datos que usaremos en el ejemplo

In [28]:
```
titanic = pd.read_csv("titanic.csv")
```

In [6]:
```
titanic.tail()
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	E
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	

In [7]:
```
titanic.shape
```

Out[7]:
```
(891, 12)
```

In [8]:
```
titanic.columns
```

Out[8]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibS
p',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

La base de datos de TITANIC son registros con la siguiente información:

- Survival 0 = No, 1 = Yes
- pclass Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd
- Sex Female and Male
- Age Age in years
- Sibsp # of siblings
- Parch # of parents
- Ticket Ticket number
- fare fare Passenger
- Cabin Cabin number
- Embarked Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

Proceso completo

1. Importamos base de datos
2. Limpieza y transformación de los datos
3. Codificación de los datos
4. Entrenar al clasificador
5. Validación del modelo generado

Limpieza y pre-procesamiento

1. Con el objetivo de evitar el ** Overfitting ** o ** sobreentrenamiento **, se va a categorizar las edades por grupos.
2. Se cambiará el valor de Cabin a solo la letra inicial, por facilidad de manejo.
3. El precio, Fare, es un valor continuo que sería mejor que estuviera simplificado. Si ejecutamos "titanic_train.Fare.describe()" vamos a obtener la distribución de esta característica, observando esto se ubicará cada valor en su cuartil correspondiente.
4. Se simplificará el nombre solo con el prefijo (Mr. Mrs. Etc.) y el apellido, y se agregará a una nueva columna.
5. Finalmente, se borrarán las características que son consideradas, en este caso, inútiles (Ticket y Name).

1. Con el objetivo de evitar el ** Overfitting ** o ** sobreentrenamiento **, se va a categorizar las edades por grupos. ***

In [9]:
```
titanic.columns
```

Out[9]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibS
p',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [12]:
```
titanic['Fare']
```

Out[12]:
```
0       7.2500
1      71.2833
2       7.9250
3      53.1000
4       8.0500
        ...
886    13.0000
887    30.0000
888    23.4500
889    30.0000
890     7.7500
Name: Fare, Length: 891, dtype: float64
```

In [21]:
```
def simplify_ages(df):
    df.Age = df.Age.fillna(-0.5)
    bins = (-1, 0, 5, 12, 18, 25, 35, 60, 120)
    group_names = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student','Young
    categories = pd.cut(df.Age, bins, labels=group_names)
    # La función CUT de Pandas nos regresa el índice
    # del bin al que pertenece el valor x
    df.Age = categories
    return df
```

2. Se cambiará el valor de Cabin a solo la letra inicial, por facilidad de manejo.

In [13]:
```
titanic.Cabin
```

Out[13]:
```
0       NaN
1       C85
2       NaN
3      C123
4       NaN
       ...
886     NaN
887     B42
888     NaN
889    C148
890     NaN
Name: Cabin, Length: 891, dtype: object
```

In [22]:
```
def simplify_cabins(df):
    df.Cabin = df.Cabin.fillna('N')
    df.Cabin = df.Cabin.apply(lambda xx: xx[0])
    return df
```

3. El precio, Fare, es un valor continuo que sería mejor que estuviera simplificado. Si ejecutamos "titanic_train.Fare.describe()" vamos a obtener la distribución de esta característica, observando esto se ubicará cada valor en su cuartil correspondiente.

In [14]:
```
titanic.Fare.describe()
```

Out[14]:
```
count    891.000000
mean      32.204208
std       49.693429
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      512.329200
Name: Fare, dtype: float64
```

In [23]:
```
def simplify_fares(df):
    df.Fare = df.Fare.fillna(-0.5)
    bins = (-1, 0, 8, 15, 31, 1000) #titanic_train.Fare.describe()
    group_names = ['Unknown', '1_quartile', '2_quartile', '3_quartile', '4_
    categories = pd.cut(df.Fare, bins, labels=group_names)
    # Lo mismo que vimos anteriormente con la función CUT
    df.Fare = categories
    return df
```

4. Se simplificará el nombre solo con el prefijo (Mr. Mrs. Etc.) y el apellido, y se agregará a una nueva columna.

In [16]:
```
titanic.columns
```

Out[16]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibS
p',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [ ]:
```
titanic['nuevaco'] = 98
```

In [19]:
```
mensaje = "hola a todos"
mensaje.split(" ")[1]
```

Out[19]:
```
'a'
```

In [24]:
```
def format_name(df):
    df['Lname'] = df.Name.apply(lambda x: x.split(' ')[0])
    df['NamePrefix'] = df.Name.apply(lambda x: x.split(' ')[1])
    return df
```

5. Finalmente, se borrarán las características que son consideradas, en este caso, inútiles (Ticket y Name).

In [25]:
```
def drop_features(df):
    return df.drop(['Ticket', 'Name', 'Embarked'], axis=1)

#Tenemos una función que manda a llamar a las funciones
#anteriormente descritas
# para transformar el dataset
def transform_features(df):
    df = simplify_ages(df)
    df = simplify_cabins(df)
    df = simplify_fares(df)
    df = format_name(df)
    df = drop_features(df)
    return df
```

In [29]:
```
titanic.head()
```

Out[29]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [26]:
```
data = transform_features(titanic)
data.head()
```

Out[26]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Lname	Nam
0	1	0	3	male	Student	1	0	1_quartile	N	Braund,	
1	2	1	1	female	Adult	1	0	4_quartile	C	Cumings,	
2	3	1	3	female	Young Adult	0	0	1_quartile	N	Heikkinen,	
3	4	1	1	female	Young Adult	1	0	4_quartile	C	Futrelle,	
4	5	0	3	male	Young Adult	0	0	2_quartile	N	Allen,	

Un algoritmo de machine learning no trabaja con datos no-numéricos, hay que transformar o codificar los datos en caso de que no cumplan esa condición.

Codificaciones finales

- Como última parte del pre-procesamiento es normalizar las etiquetas. Scikit-learn tiene un **LabelEncoder** que convierte cada valor unico en un número, haciendo más flexibles los datos para aplicar ciertos algoritmos de aprendizaje computacional.
- El resultado de esta codificación es una tabla de números, que a ojos de los humanos se ven espantosos, pero a ojos de las máquinas se ven bien.

In [31]:
```
from sklearn import preprocessing
def encode_features(datos):
    features = ['Fare', 'Cabin', 'Age', 'Sex', 'Lname', 'NamePrefix']
    #df_combined = pd.concat([df_train[features], df_test[features]])

    for feature in features:
        le = preprocessing.LabelEncoder()
        le = le.fit(datos[feature])
        datos[feature] = le.transform(datos[feature])

    return datos

d_encode = encode_features(data)
d_encode.head()
```

Out[31]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Lname	NamePrefix
0	1	0	3	1	4	1	0	0	7	73	17
1	2	1	1	0	0	1	0	3	2	136	18
2	3	1	3	0	7	0	0	0	7	251	14
3	4	1	1	0	7	1	0	3	2	198	18
4	5	0	3	1	7	0	0	1	7	11	17

Separando en training

- Ahora vamos a aplicar un algoritmo de machine learning
- Primero hay que separar las features (X) de las labels (Y)
- En ** X_all ** se van a guardar todas las features menos la que queremos predecir (Survived)
- En ** Y_all** se van a guardar solo los valores de Survived.
- Vamos a dividir de forma aletoria los datos, vamos a utilizar una función de Scikit-learn para obtener de los datos una muestra aleatoria del N% de los datos para traning y el restante será para test.
- Para hacer un poco más rigurosa el test, se organizarán los datos en K-Fold para validar la efectividad de los algoritmos entrenados.

In [32]:
```
from sklearn.model_selection import train_test_split
X_all = d_encode.drop(['Survived', 'PassengerId'], axis=1)

#Borramos las columnas Survived y PassengerID
y_all = d_encode['Survived']

num_test = 0.20 # aquí se está especificando 20% para test
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size
```

In [35]:
```
y_train.shape
```

Out[35]: (712,)

In [36]:
```
y_test.shape
```

Out[36]:
```
(179,)
```

Proceso de clasificación

Random Forest

Random Forest es un clasificador que consiste en muchos árboles de decisión y asigna la clase más votada por todos ellos.

Random Forest

- Es un algoritmo muy utilizado, tanto para clasificación como para regresión.
- Muestra buen rendimiento para datos de alta dimensionalidad.

**¿Cómo funciona?* *

- Mezcla tres conceptos: ** Bagging, decision trees y Random subspace **

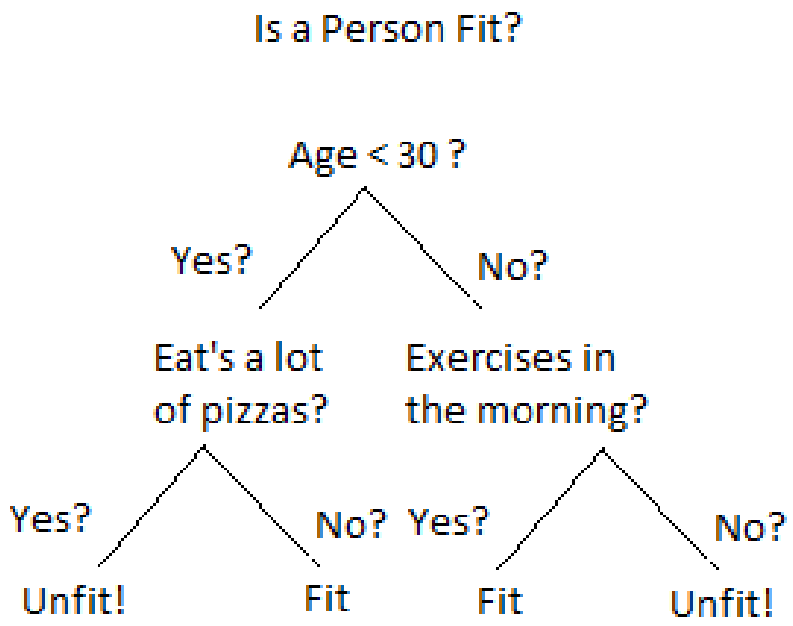
Bagging

Bootstrap Aggregation (o Bagging), es una **método de ensamble** simple y de gran ayuda.

Un método de ensamble es una técnica que combina las predicciones de múltiples algoritmos de aprendizaje automático para hacer predicciones más precisas que cualquier modelo individual haría.

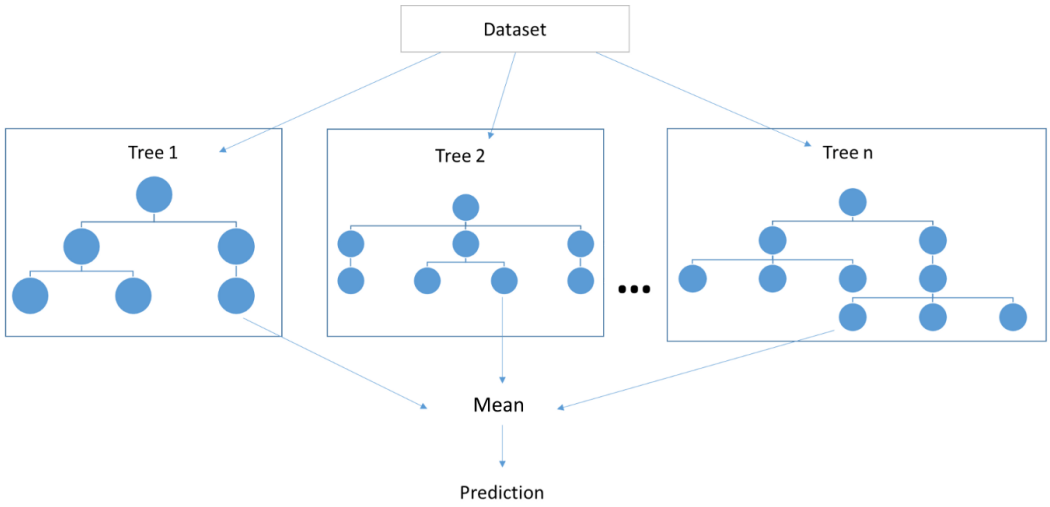
Decision trees

Los árboles de decisión es una de las estructuras más simples y útiles en aprendizaje computacional. Tienen el problema que son muy dependientes de los datos de entrenamiento, por eso son muy utilizados en combinación con el Bagging.



Random subspace

Técnica que en conjunto con Bagging nos sirven para alimentar, por medio de muestreo con reemplazo, cada uno de los clasificadores.



Random Forest

Cada árbol se construye utilizando el siguiente algoritmo:

- Sea el número de casos de entrenamiento N, y el número de variables en el clasificador sea M.
- Se nos dice el número m de las variables de entrada que se utilizarán para determinar la decisión en un nodo del árbol; m debe ser mucho menor que M.

- Elija un conjunto de entrenamiento para este árbol eligiendo n veces con reemplazo de todos los N casos de entrenamiento disponibles (es decir, tome una muestra bootstrap).
- Se utiliza el resto de los casos para estimar el error del árbol, mediante la predicción de sus clases.

** ¿Cuándo acaba?**

- Cuando todos los elementos han sido clasificados; es decir, llegaron a las hojas de los árboles.

Parámetros de Random Forest

n_estimators : integer, optional (default=10)

El número de árboles en el bosque

criterion : string, optional (default="gini")

```
Esto mide la calidad de la división.
```

max_depth : integer or None, optional (default=None)

```
El valor máximo de la profundidad del árbol (o de cada árbol)
Si el valor es None, los nodos se expanden hasta que todas las hoja
s tengan una sola clase o hasta que todas las hojas contengan menos
el mínimo especificado para dejar de hacer la división o split.
```

min_samples_split : int, float, optional (default=2)

```
Número mínimo de ejemplos para dejar de hacer un split.
```

Etc.

In [58]:
```
# Seleccionamos el tipo de clasificador, en este caso RandomForest.
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, min_samples_leaf=5)
clf.fit(X_train, y_train)
```

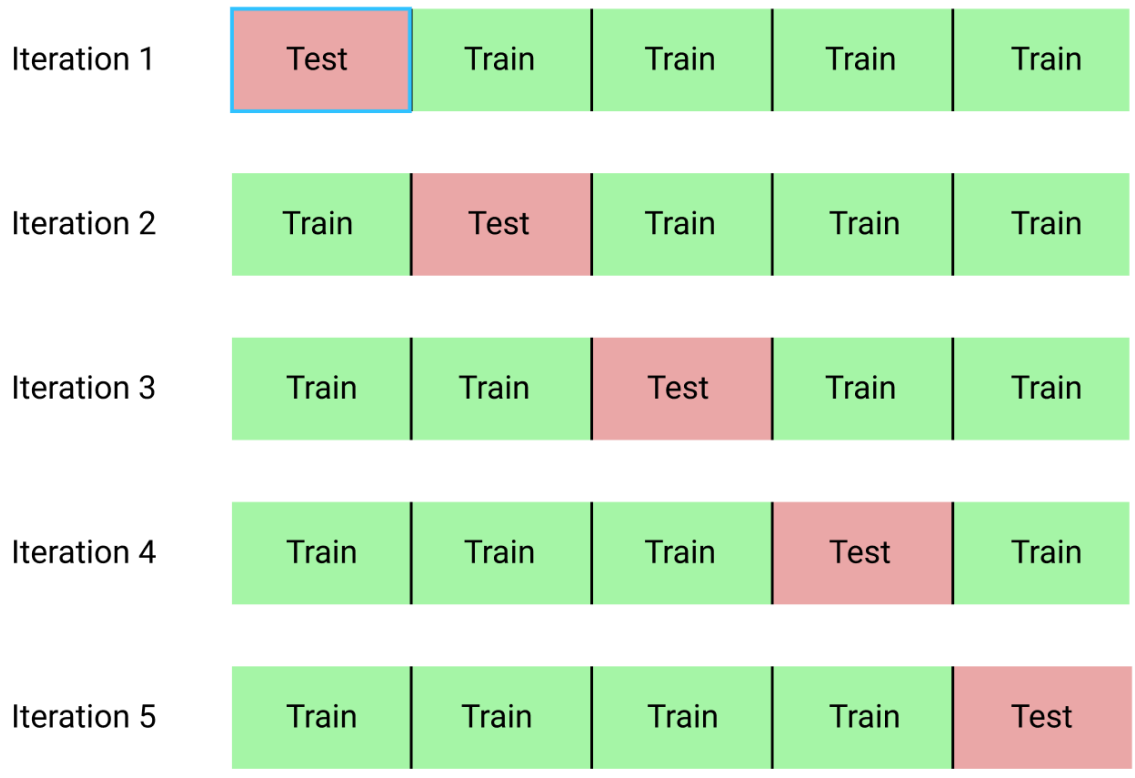
Out[58]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='au
to',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=Non
e,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [59]:
```
# Se hacen las predicciones sobre el test
from sklearn.metrics import make_scorer, accuracy_score, f1_score
predictions = clf.predict(X_test)
# se imprime el accuracy
print("Accuracy: ", accuracy_score(y_test, predictions))
#Imprimimos la f-score
print("F1-Score: ", f1_score(y_test, predictions))
```

```
Accuracy:  0.8379888268156425
F1-Score:  0.7819548872180451
```

Validando con el enfoque de KFold

- KFold ayuda a verificar, de una forma más exhaustiva, la efectividad del algoritmo. KFold divide el dataset en N partes, entonces el algoritmo de clasificación se utiliza una parte diferente como test en cada iteración.



In [51]:
```
from sklearn.model_selection import cross_val_score
scores1 = cross_val_score(clf, X_train, y_train, cv=6)
print(scores1)
```

```
[0.80672269 0.79831933 0.81512605 0.75630252 0.79661017 0.84745763]
```

In [54]:
```
import numpy as np
np.mean(scores1)
```

Out[54]: 0.8034230641409107

In [55]:
```
np.mean(scores)
```

Out[55]: 0.7455364181294861

In [50]:
```
#Podemos cambiar la métrica de evaluación con el parámetro scoring
scores = cross_val_score(clf, X_test, y_test, cv=6, scoring='f1_macro')
print(scores)
```

```
[0.74358974 0.784689   0.88446727 0.7        0.6534018  0.70707071]
```

Ahora, utilizarémos otra BD para clasificar y para poner en práctica lo aprendido hasta ahora.

BIARE

In [60]:
```
# Cargamos los datos
import pandas as pd
biare = pd.read_csv("Bienestar.csv", encoding="latin", low_memory=False)
```

In [63]:
```
biare.shape
```

Out[63]: (10654, 201)

In [67]:
```
set(biare['gr_feliz'])
```

Out[67]: {1, 2, 3, 4}

In [68]:
```
set(biare['gr_satis'])
```

Out[68]: {1, 2, 3, 4}

In [62]:
```
list(biare.columns)
```

...

Cree un modelo con Random Forest para predecir el grado de felicidad.

- Puede ser con dos, tres o más variables, puede probar diferente modelos y medir el Accuracy y la medidaf1

In [52]:
```
biare.columns
```

Out[52]:
```
Index(['anio_reg', 'trimestre', 'folio', 'hog_ent_1', 'hog_ent_2', 'id_po
bla',
       'sexo', 'edad', 'edo_conyug', 'grado_inst',
       ...
       'upm_dis', 'estrato', 'factor', 'gct_per', 'tam_hog', 'gr_edad',
       'n_inst', 'cond_act', 'gr_satis', 'gr_feliz'],
      dtype='object', length=201)
```

In [53]:
```
biare['gr_satis']
```

Out[53]:
```
0        4
1        3
2        4
3        4
4        3
        ..
10649    3
10650    4
10651    1
10652    4
10653    3
Name: gr_satis, Length: 10654, dtype: int64
```

In [54]:
```
biare['gr_feliz']
```

Out[54]:
```
0        3
1        3
2        4
3        4
4        4
        ..
10649    3
10650    4
10651    3
10652    4
10653    4
Name: gr_feliz, Length: 10654, dtype: int64
```

In [ ]:

Ahora otro algoritmo de clasificación, pero ahora no-supervisado

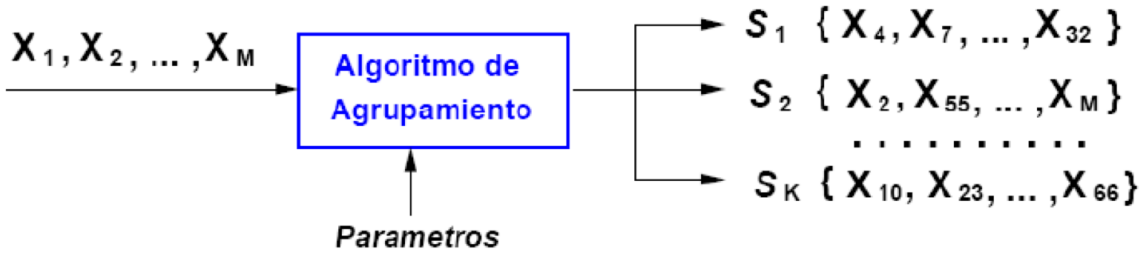
Clustering

Cluster

- Conjunto de valores que tienen algo en común, y se agrupan en función de determinado rasgo.

Algoritmos de agrupamiento

- Tienen como objetivo devolver al usuario una serie de puntos que en cierto modo representan al resto de puntos iniciales por su posición representativa con respecto al total.



k-Means

- Aprendizaje no supevisado
- Técnica de agrupamiento con diversos parámetros
  - Número de clusters
  - Criterio de paro
  - Valores iniciales (semillas, normalmente se deja con selección aleatoria)

k-Means

1. Toma como parámetro inicial el número de k, que es el número de clusters a generar
2. Selecciona k elementos de forma aleatoria, estos elementos son los centroides de cada cluster
3. A cada objeto (diferente de los centroides) se le asigna el cluster al que se parece más, esto en base a la distancia entre el objeto y el centroide (o media del cluster).
4. Se calcula el nuevo valor del centroide
5. Se itera del paso 3 al 4 hasta que no haya cambios en los valores de los centroides u otro criterio de paro.

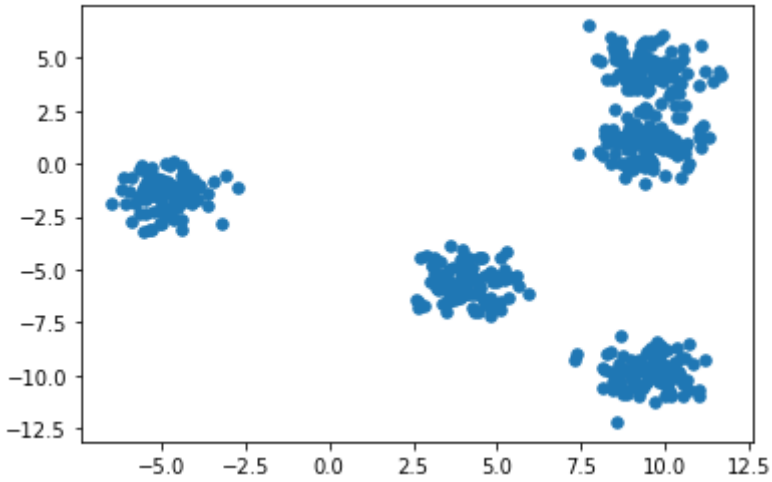
Medidas de similaridad

- Normalmente se utiliza una medida de similaridad basada en el error cuadrático o la distancia Euclidiana
- Distancia Euclideana

$$\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

In [32]:
```
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [86]:
```
#Aquí creamos datos sintéticos para observar en un plot sus muestras y sus
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=500, centers=5, cluster_std=.8, random_sta
plt.scatter(X[:, 0], X[:, 1], s=30);
```



In [70]:
```
X.shape
```

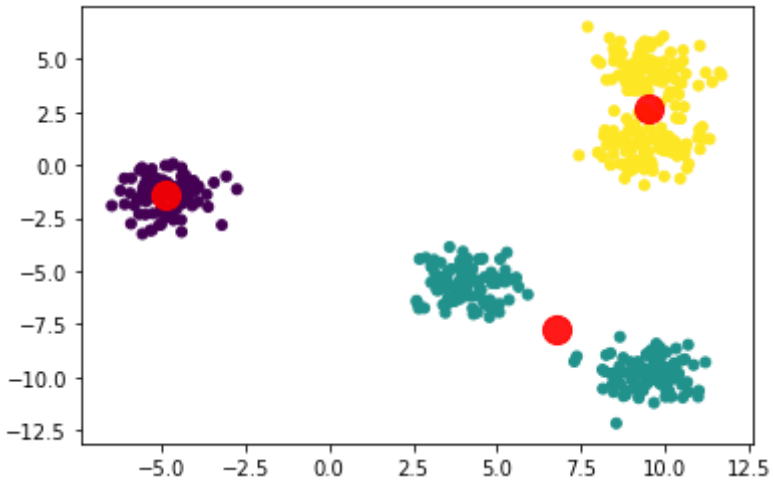
Out[70]:
```
(500, 2)
```

In [91]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

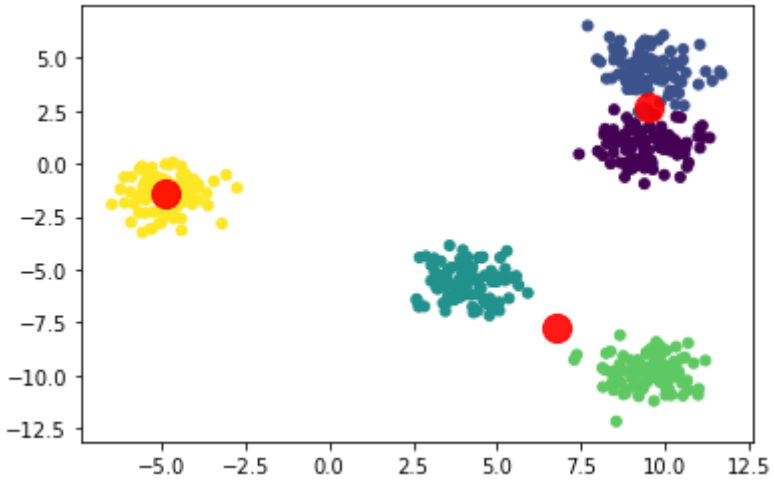
In [90]:
```
accuracy_score(y_true, y_kmeans)
```

Out[90]:
```
0.2
```

In [92]:
```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=25, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.9);
```



In [93]:
```
plt.scatter(X[:, 0], X[:, 1], c=y_true, s=25, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.9);
```



In [38]:
```
model = KMeans(n_clusters=3)
model.fit(X)
predictedY = model.predict(X)
```

In [37]:
```
print(model.labels_)
```

...

In [80]:
```
accuracy_score(y_kmeans, predictedY)
```

Out[80]: 0.25

In [41]:
```
#Base de datos de IRIS
iris = datasets.load_iris()
iris.data
iris.target
x = pd.DataFrame(iris.data, columns=['Sepal Length', 'Sepal Width', 'Petal
y = pd.DataFrame(iris.target, columns=['Target'])
```

In [74]:
```
model = KMeans(n_clusters=3)
model.fit(x)
# The fudge to reorder the cluster ids.
pre = np.choose(model.labels_, [1, 0, 2]).astype(np.int64)
accuracy_score(y['Target'], pre)
```

Out[74]:
```
0.8933333333333333
```

Ejercicio: Ahora aplique el algoritmo de aprendizaje computacional no supervisado (KMeans) a la base de datos de Kobi o del Titanic, mida el rendimiento, modifique algunos parámetros del algoritmo y repita el experimento, haga al menos tres experimentos diferentes.