Scikit Learn

Objetivos:

- Conocer en qué consiste el **Aprendizaje Automático** o **Machine Learning (M.L)**

- Obtener una vista panorámica sobre todo el contexto que rodea a M.L: Manejo de los datos, métodos y algoritmos de aprendizaje.

- Introducirse, de forma particular, en la librerías de M.L, **Scikit-Learn** para **Python 3**, realizando una práctica sobre un conjunto de datos de ejemplo.

Segunda Parte:

Librerías de Scikit Learn con Python 3

Scikit-Learn

Contenidos :

- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn.

- Es probablemente la librería más útil para Machine Learning en Python
- Es de código abierto y es reutilizable en varios contextos, fomentando el uso académico y comercial
- Proporciona una gama de algoritmos de aprendizaje supervisados y no supervisados en Python
- Está construida sobre Numpy, SciPy y Matplotlib
- Una de las principales razones detrás del uso de herramientas de código abierto es la gran comunidad que tiene, lo mismo sucede para scikit-learn. Hay más de 30 colaboradores mejorando esta librería.

Scikit-Learn. Instalación

Para instalar Scikit-Learn :

```
$ sudo pip3 install scikit-learn
```

Para comprobar la versión de Scikit-Learn que tenéis instalada:

```
>>> import sklearn
>>> sklearn.__version__
'0.21.3‘
```

La última versión de Scikit-learn es de Julio de 2019. **scikit-learn 0.21.3**

Desde la versión 0.21 se requiere tener Python 3.5 or superior.

Scikit-Learn.

Algunos componentes de Scikit-Learn:

La librería de scikit-learn viene cargada con muchas funciones, entre otras:

- **Algoritmos de aprendizaje supervisados.**

  La gran mayoría de los algoritmos de M.L sobre aprendizaje supervisado forman parte de scikit-learn, desde los modelos lineales generalizados, como lo es regresión lineal, las máquina de vectores de soporte (SVM), árboles de decisión y hasta los métodos bayesianos

  Ej: Módulo **sklearn.svm**

- **Validación cruzada**

  Existen varios métodos para verificar la precisión de los modelos supervisados y esta librería cuenta con las instrucciones necesarios para poder implementar estos métodos sin mucho esfuerzo. Ej. Módulo **sklearn.model_selection**

- **Algoritmos de aprendizaje no supervisados.**

  Cuenta con una gran variedad de algoritmos disponibles para aprendizaje no supervisado, comenzando por la agrupación, el análisis factorial, el análisis de componentes principales y las redes neuronales no supervisadas.

  Ej: módulo **sklearn.cluster**

Scikit-Learn.

Algunos componentes de Scikit-Learn:

- **Varios conjuntos de datos o dataset**

  Pone a disposición varios dataset que son muy útiles para practicar tus conocimientos de Machine Learning con Python, implementando esta librería.

  Ej. Módulo **sklearn.datasets**

- **Extracción, selección de características. Métricas**

  Esta librería es muy útil para extraer características de imágenes o texto, así como también para identificar atributos significativos, a partir de los cuales crear modelos supervisados. Ej: Módulo **sklearn.metrics**

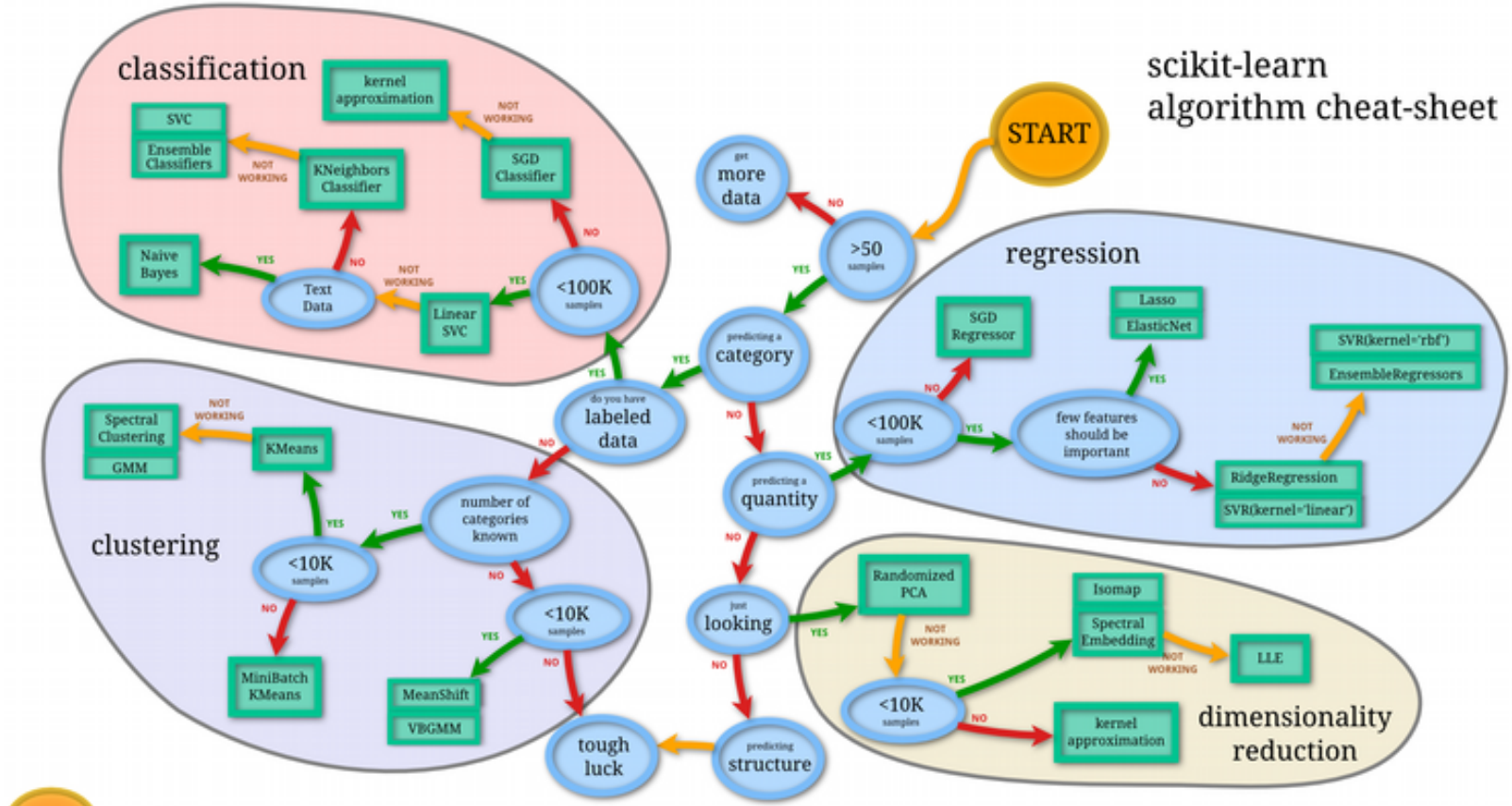
- **Etc…**

El conjunto completo de la API lo podemos ver en la documentación:

https://scikit-learn.org/dev/modules/classes.html#

Scikit-Learn.

ELECCIÓN DEL ESTIMADOR ADECUADO PARA SCIKIT-LEARN:



Back

https://scikit-learn.org/stable/tutorial/machine_learning_map/

Scikit-Learn

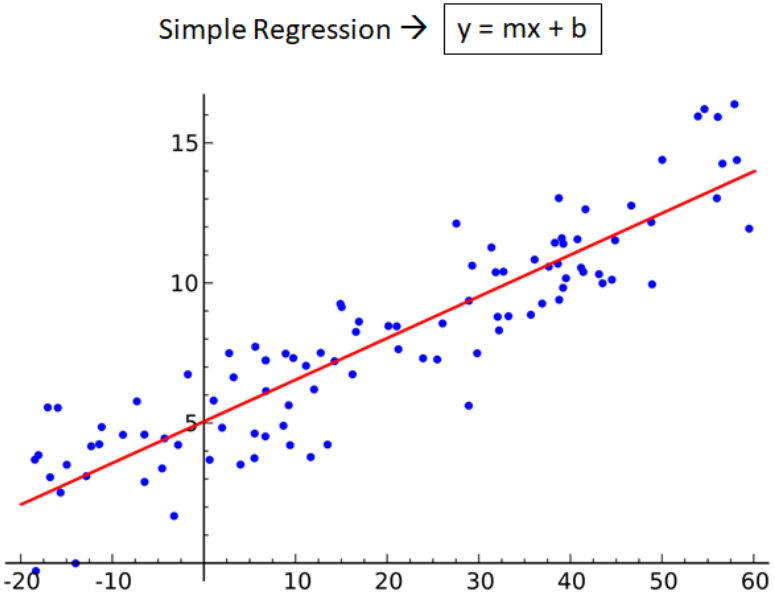
Contenidos :

- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn. Regresión lineal Simple

- Es una forma estadística de establecer una relación de dependencia entre unas variables continuas cuantitativas:
  - Variables independientes ‘x’ (características): es una variable que se manipula para determinar el valor de una variable dependiente. Simplemente, son las características que queremos usar para predecir algún valor dado de y.
  - Variable dependiente ‘y’ (objetivo): la variable dependiente depende de los valores de la variable independiente. Es la variable de respuesta, o lo que queremos predecir (target).

Este algoritmo consiste en hallar una línea recta que mejor se ajuste al conjunto de datos.
Una vez obtenida esta línea recta, seremos capaces de hacer predicciones hipotéticas sobre cuál será el valor de “y” dado “X”.



Scikit-Learn. Regresión lineal Simple

Ecuación de la recta: **y = ax + b**

y – es la variable dependiente o la variable a predecir.

x – es la variable independiente o la variable que usamos para hacer una predicción.

a – es la pendiente o el valor que debe ser determinado, se le conoce como coeficiente.

b – es la constante que debe ser determinada, cuando x es igual a 0, entonces y = b.

¿Cómo funciona el algoritmo?

El objetivo es minimizar la distancia entre todos los datos y nuestra línea (el error) para determinar la mejor línea. Existen muchos métodos para cumplir con este objetivo. El método de los ***mínimos cuadrados,*** que buscar minimizar la distancia vertical de todos los puntos a la línea recta. Para ello:

- **Utiliza el error al cuadrado que tiene buenas propiedades matemáticas, por lo que es más fácil diferencias y calcular el descenso del gradiente.**
- **Es fácil de analizar y computacionalmente más rápido, es decir, puede aplicarse rápidamente a conjuntos de datos que tienen miles de características.**
- **La interpretación es mucho más fácil que otras técnicas de regresión**

$$a = \frac{\sum(xi - xmedia)(yi - ymedia)}{\sum(xi - xmedia)^2}$$

$$b = ymedia - a(xmedia)$$

Scikit-Learn. Regresión lineal Simple

Es una técnica de naturaleza paramétrica porque hace ciertas suposiciones basadas en el conjunto de datos. El rendimiento de un modelo de regresión está condicionado al cumplimiento de estas suposiciones, tanto en la regresión simple como múltiple:

- Linealidad. Debe existir una relación lineal entre las variables dependientes e independientes. Por lineal, significa que un cambio en la variable dependiente por un cambio de unidad en la variable independiente es constante.
- Eliminar el ruido. La regresión lineal asume que sus variables de entrada y salida no son ruidosas. Podríamos usar operaciones de limpieza de datos que nos permitan exponer mejor y aclarar la señal en los datos.
- Eliminar la colinealidad. La regresión lineal se ajustará demasiado a los datos cuando tenga variables de entrada altamente correlacionadas. Considera calcular correlaciones por pares para sus datos de entrada y eliminar los más correlacionados
- La variable dependiente y los términos de error deben tener una distribución normal.

Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Tal y como comentamos, en las características de Sckit-Learn se dispone de varios dataset, muy básicos, con los que puedes practicar tus conocimientos de Machine Learning, no son muchos, pero cubre los principales tipos de algoritmos como regresión y clasificación. Concretamente, *"El precio de las casas de Boston"* ha sido utilizado en mucho artículos sobre regresión, tal y como indica en su propia documentación y lo mejor es que no tenemos que descargarlo de ninguna fuente de datos como por ejemplo https://www.kaggle.com

Para ver la documentación de Sckit-Learn sobre los conjunto de datos disponibles, podemos ir a :

https://scikit-learn.org/stable/datasets/index.html#datasets

Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Nuestro objetivo es desarrollar un modelo para predecir el precio de las casas en Boston de acuerdo al número de habitaciones que cuenta la vivienda.

Es aconsejable seguir una rutina de pasos, que va a ser muy parecida cada vez que procesemos un conjunto de datos

1º Importar las librerías necesarias.

La librería Scikit-Learn proporciona una función llamada **LinearRegression** para hacer justamente este trabajo, y de esta forma ahorrarnos todos los cálculos matemáticos. **LinearRegression** pertenece al módulo **Linear_model** de sklearn, y crea objetos del tipo LinearRegression para trabajar con ellos.

```
# Importación de las librerías necesarias

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
```

De sklearn importamos datasets, que es donde se guardan todos los conjuntos de datos, y linear_model , ya que vamos a trabajar con Regresión Lineal.

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

2º Cargar los datos.

Seguidamente importamos el famoso dataset de los precios de las casa en Boston, y lo tendremos disponible para nosotros. Para ello haremos uso de la función **.load_boston( ),** del módulo **datasets**

```
#Importamos los datos de la librería de boston
boston = datasets.load_boston()
print(boston)
print()
```

Utilizamos la función .load_boston( )
Y comprobamos lo que nos sale por pantalla.

Hay mucha información de golpe !!

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

3º Comprobar la información que contiene el conjunto de datos

Entramos en un proceso de compresión de los datos, en el cual analizaremos la información con la que vamos a trabajar. Para ello ejecutaremos el método **.keys()** de nuestro objeto, el conjunto de datos **boston**

```
#comprobar la información que contiene el dataset
print('Información en el dataset:')
print(boston.keys())
print()
```

```
Información en el dataset:
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

Como podemos observar, nos devuelve la siguiente información: los datos **"data"**, seguido de **"target"** que es la columna con las etiquetas o respuestas, posteriormente **"feature_names"** que son los nombres de cada una de las columnas de datos y **DESCR** que es la descripción completa del dataset.

Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

4º Aprovechamos para echar un vistazo a la descripción del dataset:

.DESCR es una cadena de texto que podemos imprimir para analizar su contenido

```
#Verificar las características del dataset
print('Características del dataset:')
print(boston.DESCR)
```

```
.. _boston_dataset:
Boston house prices dataset
---------------------------
**Data Set Characteristics:**
    :Number of Instances: 506
:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14)
is usually the target
.............
..........
```

El dataset cuenta con 506 datos y 13 atributos, la columna 14 es el target y es la media del valor de las viviendas. También tenemos la descripción del significado de cada columnas o atributo. Un dato importante es, que nos avisa de que no existe ningún valor perdido, en consecuencia, nuestros datos están completos y no necesitaremos realizar mucho preprocesamiento de los mismos.

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

5º Verificar la cantidad de datos del dataset

.data es un **numpy.ndarray** que contiene el grueso de los datos.

Así que podemos usar lo que ya sabemos de numpy y utilizar **.shape** que nos devuelve la tupla: *(filas, columnas)*

```
#Verifico la cantidad de datos que hay en los dataset
print('Cantidad de datos:')
print(boston.data.shape)
print()
```

```
Cantidad de datos:
(506, 13)
```

Es una información redundante a la que nos a dado la descripción anterior, pero no siempre el conjunto de datos, tiene una descripción que nos dé esa información, por eso, es una buena práctica acostumbrase a una rutina de comprobaciones sistemáticas, cada vez que nos enfrentamos al conjunto de datos..

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

6º Verificar también las columnas o atributos

.feature_names es un **numpy.ndarray** que contiene los nombres de las columnas.

Así que procedemos a mostrar con print el contenido del array

```
#Verificar también las columnas o atributos
print('Nombres de los atributos:')
print(boston.feature_names)
```

```
Nombres columnas:
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO' 'B' 'LSTAT']
```

De nuevo, se trata de una información que ya nos había dado la descripción del dataset, pero no siempre el conjunto de datos nos da esa información, por lo que seremos sistemáticos, con las comprobaciones.

Aunque tenemos mucha información, en nuestro análisis solamente vamos a tomar la columna correspondiente al número de habitaciones de la casa ("**RM**")

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

7º Preparar los datos para REGRESIÓN LINEAL SIMPLE

Solamente se requiere una variable independiente, que en nuestro caso será la correspondiente al número de habitaciones (**RM**), pero también se puede tomar, por ejemplo, la columna correspondiente a los años de construcción de la vivienda.

Recuperamos un array de Numpy de **.data[ ]** la posición 5, concretamente.

```
#Nos quedamos con la columna 5 del dataset
X = boston.data[ : , np.newaxis , 5]
```

```
.....
    [6.12 ],
    [6.976],
    [6.794],
    [6.03 ]])
```

Seleccionamos nuestra variable objetivo o dependiente, que será el precio de la vivienda

```
y = boston.target
```

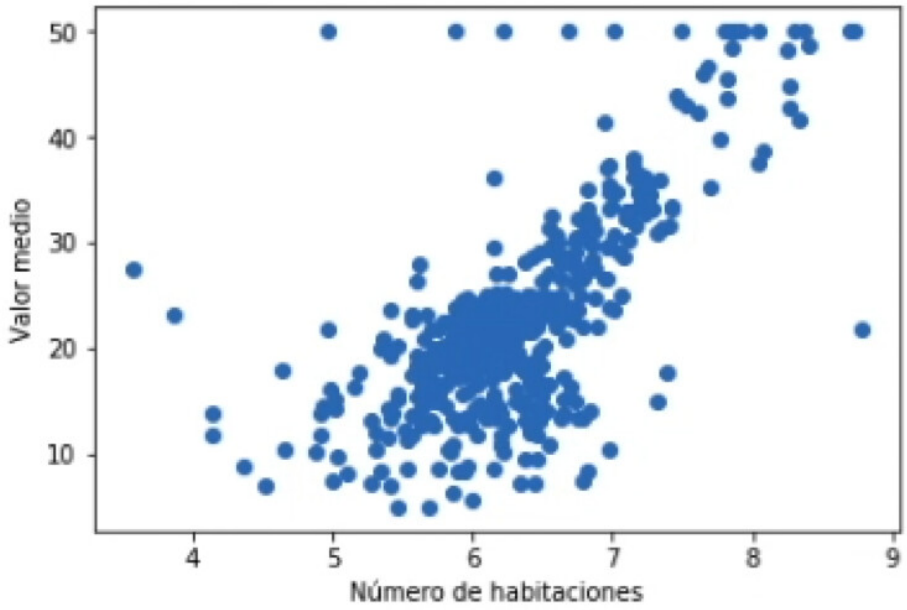
Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

8º Plotearemos los datos utilizando una gráfica de dispersión para ver la distribución de los mismos

```
#Mostrar el gráfico de dispersión de los datos
plt.scatter(X, y)
plt.xlabel('Número de habitaciones')
plt.ylabel('Valor medio')
plt.show()
```

A simple vista ya podemos deducir que el modelo de Regresión Lineal que vamos a construir tendrá un error muy alto.



Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

9º Comenzamos con la REGRESIÓN LINEAL SIMPLE.

Antes de nada, debemos separar los datos en entrenamiento y prueba por lo que importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn.**

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
from    sklearn.model_selection    import    train_test_split

#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos

X_train,  X_test,  y_train,  y_test  =  train_test_split( X,  y, test_size = 0.2 )
```

Invocaremos la creación de nuestro objeto para regresión lineal, usando la función **.LinearRegression()** , que ya habíamos importado anteriormente.

```
#Defino el algoritmo a utilizar
lr = linear_model.LinearRegression( )
```

Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Entrenamos nuestro modelo con los datos de entrenamiento y utilizando la función fit(X , y).

```
#Entrenamiento  del modelo
lr.fit( X_train , y_train )
```

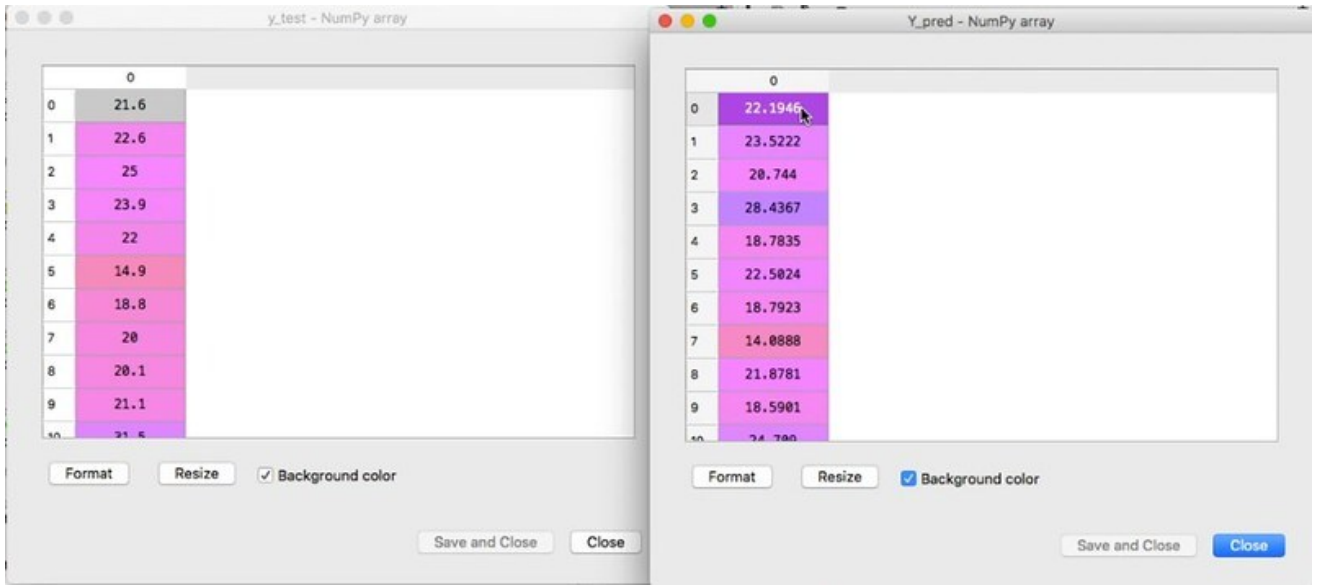
Y finalmente realizamos la predicción utilizando los datos de prueba. Utilizaremos para ello la función **.predict(X)** de nuestro objeto **lr**

Es obligatorio el uso de la función fit en el paso anterior, para que no de un error.

```
#Realizar una predicción
Y_pred = lr.predict( X_test )
```

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

Vamos a hacer una comparación rápida los valores obtenidos en nuestra predicción con los valores reales, para ver que tal es el comportamiento de nuestro modelo. Para ello usaremos el Explorador de variables de Spyder.



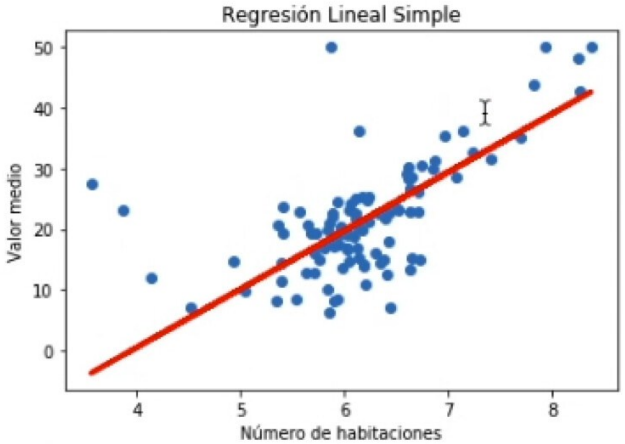
A la izquierda están los datos reales, que separamos en **y_test**. A la derecha están los datos obtenidos de la predicción **Y_pred**. Si observamos con detenimiento nos podemos dar cuenta que ocasionalmente, los datos reales se aproximan a los calculados, mientras que otras veces son muy diferentes.

Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Ya hemos visto que el modelo no es del todo eficiente, pero veamos gráficamente para detallarlo con más cuidado usando matplotlib.

```
#Reprentamos gráficamente los datos junto con el modelo
plt.scatter(X_test, y_test)
plt.plot(X_test, Y_pred, color='red', linewidth=3)
plt.title('Regresión Lineal Simple')
plt.xlabel('Número de habitaciones')
plt.ylabel('Valor medio')
plt.show()
```



La línea roja sería nuestro modelo y los puntos azules son los datos de pruebas con los que lo entrenamos.
El objetivo de un modelo de regresión lineal , es que la recta represente el mayor número de muestras y de esta forma disminuir el error dado por la distancia desde la recta hasta el punto. Todos los datos fuera de la recta no son predichos correctamente por el modelo , siendo algunas diferencias considerables.
Ya desde un principio se veía que los datos se encontraban muy dispersos por lo que es difícil que una recta cubra la mayoría de los datos. Esto modelo incurre en una mala predicción. Esto no quiere decir que este algoritmo sea malo sino más bien que este algoritmo no es adecuado para este conjunto de datos.

Scikit-Learn. Regresión lineal Simple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Calculemos ahora los valores de la pendiente e intersección para ver como queda la ecuación del modelo. Para ello accedemos a los atributos de la pendiente **.coef_** y de la constante o intersección **.intercept_** de nuestro objeto **"lr".** (y = ax +b)

```
#Valores de la ecuación del modelo de regresión
print('VALORES DEL MODELO DE REGRESIÓN LINEAL SIMPLE \n')
pendiente = lr.coef_
constante = lr.intercept_
print('La ecuación del modelo es:')
print('y = ', pendiente, 'x +', constante)
```

```
VALORES DEL MODELO DE REGRESIÓN LINEAL SIMPLE

La ecuación del modelo es:
y =  [8.7846038] x + -32.64653185536959
```

Scikit-Learn. Regresión lineal Simple Ejemplo Práctico. *"Precio de las Casas de Boston"*

Un último paso que vamos a realizar es calcular la precisión del algoritmo. Para ello empleamos el método **.score(X , y)** de nuestro objeto de regresión lineal. En X e y le pasamos las variables independientes y salidas del conjunto de entrenamiento. Nos devuelve el resultado de la estadística R^2.

```
# precisión del modelo
print('Precisión del modelo: ',end='')
print("%.3f " % lr.score(X_train, y_train))
```

```
Precisión del modelo: 0.454
```

El resultado obtenido es de 0,454, lo cual no es una predicción muy buena.

Este algoritmo no es el mejor para este conjunto de datos.

Por eso resulta interesante conocer el funcionamiento de los algoritmos de Machine Learning, para que, en base a un preanálisis de los datos, podamos deducir el algoritmo más adecuado para implementar.

Scikit-Learn

Contenidos :

- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn. Regresión lineal Múltiple.

- La Regresión Lineal Múltiple es muy parecida a la simple, pero en este caso son múltiples variables independientes, las que contribuyen a la variable dependiente.

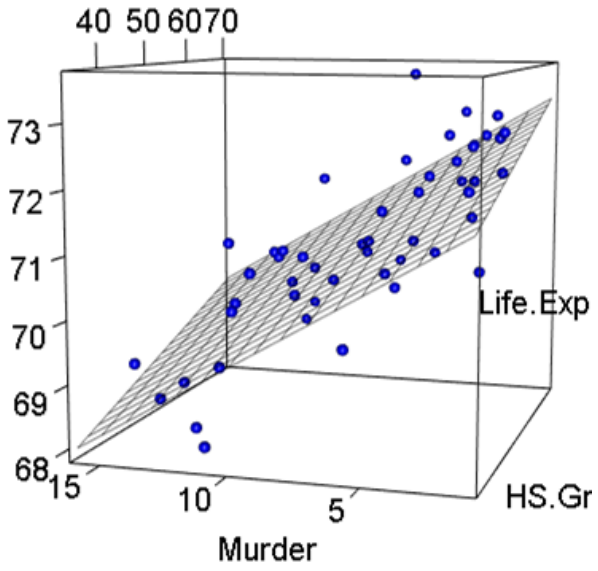
  Se manejan múltiples coeficientes y a su vez es computacionalmente más compleja debido a las variables añadidas

  La Regresión Lineal Simple es poco utilizada en escenarios del mundo real . Por lo general, una variable dependiente depende de múltiples factores, por ejemplo, para predecir el precio de una vivienda se suele mirar el lugar en donde se encuentra, el tamaño, el número de habitaciones, entre otros…

  Cuando la salida a predecir depende de más de una variable, se puede utilizar un modelo más complejo que tenga en cuenta ***n*** dimensiones adicionales:

$$y = a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots a_n \cdot x_n + b$$

  La siguiente imagen muestra un hiperplano ajustado o una regresión lineal múltiple con dos características: $y = a_1 \cdot x_1 + a_2 \cdot x_2 + b$



Scikit-Learn. Regresión lineal Múltiple.

- El uso de más variables puede contribuir a conseguir mejores predicciones. Pero no se deben incluir todas las variables independientes a la vez. Hay que seleccionar las mejores variables independientes que puedan contribuir a la variable dependiente.

  Para esto, podemos construir una matriz de correlación para todas las variables independiente e incluimos la variable dependiente.

  A partir de esta matriz, elegimos las variables independientes en orden decreciente de valor de correlación.

  Ejecutamos el modelo de regresión repetidas veces, añadiendo una nueva variable independiente cada vez, hasta que la inclusión de nuevas características no mejore de forma destacada la estimación del error.

- En algunos casos, agregar más variables independientes puede empeorar las cosas, en lugar de mejorar, esto se conoce como ajuste excesivo.

- Por otra parte, cuando se agrega más variables independientes se crean relaciones entre ellas y esto no es deseable, tal y como explicamos en la Regresión Lineal Simple con la colinealidad.

Scikit-Learn. Regresión lineal Múltiple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

De nuevo vamos a desarrollar un modelo para predecir el precio de las casas en Boston, pero esta vez tendremos en cuenta el número de habitaciones de la vivienda, el tiempo que lleva ocupada y la distancia a la que se encuentra la misma de los centros de trabajo.

Como en el caso de la regresión simple, seguiremos una rutina de pasos parecida, cada vez que procesemos un conjunto de datos:

1º Importar las librerías necesarias.

La librería Scikit-Learn proporciona una función llamada **LinearRegression** para hacer justamente este trabajo, y de esta forma ahorrarnos todos los cálculos matemáticos. **LinearRegression** pertenece al módulo **Linear_model** de sklearn, y crea objetos del tipo LinearRegression para trabajar con ellos.

```
#Importación de las librerias necesarias
from sklearn import datasets, linear_model
```

De sklearn importamos datasets, que es donde se guardan todos los conjuntos de datos, y linear_model , ya que vamos a trabajar con Regresión Lineal.

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

2º Cargar los datos.

Seguidamente importamos el famoso dataset de los precios de las casa en Boston, y lo tendremos disponible para nosotros. Para ello haremos uso de la función **.load_boston( ),** del módulo **datasets**

```
#Importamos los datos de la librería de boston
boston = datasets.load_boston()
print(boston)
print()
```

Utilizamos la función .load_boston( )
Y comprobamos lo que nos sale por pantalla.

Hay mucha información de golpe !!

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

3º Comprobar la información que contiene el conjunto de datos

Entramos en un proceso de compresión de los datos, en el cual analizaremos la información con la que vamos a trabajar. Para ello ejecutaremos el método **.keys()** de nuestro objeto, el conjunto de datos **boston**

```
#comprobar la información que contiene el dataset
print('Información en el dataset:')
print(boston.keys())
print()
```

```
Información en el dataset:
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

Como podemos observar, nos devuelve la siguiente información: los datos **"data"**, seguido de **"target"** que es la columna con las etiquetas o respuestas, posteriormente **"feature_names"** que son los nombres de cada una de las columnas de datos y **DESCR** que es la descripción completa del dataset.

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

4º Aprovechamos para echar un vistazo a la descripción del dataset:

.DESCR es una cadena de texto que podemos imprimir para analizar su contenido

```
#Verificar las características del dataset
print('Características del dataset:')
print(boston.DESCR)
```

```
.. _boston_dataset:
Boston house prices dataset
---------------------------
**Data Set Characteristics:**
    :Number of Instances: 506
:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14)
is usually the target
.............
..........
```

El dataset cuenta con 506 datos y 13 atributos, la columna 14 es el target y es la media del valor de las viviendas. También tenemos la descripción del significado de cada columnas o atributo. Un dato importante es, que nos avisa de que no existe ningún valor perdido, en consecuencia, nuestros datos están completos y necesitaremos realizar mucho preprocesamiento de los mismos.

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

5º Verificar la cantidad de datos del dataset

.data es un **numpy.ndarray** que contiene el grueso de los datos.

Así que podemos usar lo que ya sabemos de numpy y utilizar **.shape** que nos devuelve la tupla: *(filas, columnas)*

```
#Verifico la cantidad de datos que hay en los dataset
print('Cantidad de datos:')
print(boston.data.shape)
print()
```

```
Cantidad de datos:
(506, 13)
```

Es una información redundante a la que nos a dado la descripción anterior pero no siempre ,el conjunto de datos, tiene una descripción que nos dé esa información, por eso, es una buena práctica acostumbrase a una rutina de comprobaciones sistemáticas, cada vez que nos enfrentamos al conjunto de datos..

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

6º Verificar también las columnas o atributos

.feature_names es un **numpy.ndarray** que contiene los nombres de las columnas.

Así que procedemos a mostrar con print el contenido del array*)*

```
#Verificar también las columnas o atributos
print('Nombres de los atributos:')
print(boston.feature_names)
```

```
Nombres columnas:
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO' 'B' 'LSTAT']
```

De nuevo, se trata de una información que ya nos había dado la descripción del dataset, pero no siempre el conjunto de datos nos da esa información, por lo que seremos sistemáticos, con las comprobaciones.

Para nuestro análisis nos interesan las columnas correspondientes al número de habitaciones de la casa "RM", el tiempo que lleva ocupada "AGE" y la distancia a la se encuentra de los centros de trabajo de Boston "DIS".

Scikit-Learn. Regresión lineal Múltiple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

7º Preparar los datos para REGRESIÓN LINEAL MÚLTIPLE

En este caso se requieren tres variable independientes: **'RM' ,'AGE' ,'DIS'** que se encuentran ubicadas en la posición de columnas 5, 6 y 7 de los datos. Mediante el acceso al array de Numpy con **.data[ ]** del que recuperamos la parte que nos interesa.

```
#Seleccionamos las columna 5, 6 y 7 del dataset
X_multiple = boston.data[:, 5:8]
print(X_multiple)
```

```
[[ 6.575  65.2    4.09  ]
 [ 6.421  78.9    4.9671]
 [ 7.185  61.1    4.9671]
 ...
 [ 6.976  91.     2.1675]
 [ 6.794  89.3    2.3889]
 [ 6.03   80.8    2.505 ]]
```

Seleccionamos nuestra variable objetivo o dependiente, que será el precio de la vivienda

```
#variable y dependiente u objetivo
y_multiple = boston.target
```

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

8º Comenzamos con la REGRESIÓN LINEAL MÚLTIPLE

Antes de nada, debemos separar los datos en entrenamiento y prueba por lo que importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn.**

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
from sklearn.model_selection import train_test_split

#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos

X_train, X_test, y_train, y_test = train_test_split(X_multiple, y_multiple, test_size=0.2)
```

Invocaremos la creación de nuestro objeto para regresión lineal, usando la función **.LinearRegression()** , que ya habíamos importado anteriormente.

```
#Defino el algoritmo a utilizar
lr_multiple = linear_model.LinearRegression()
```

Scikit-Learn. Regresión lineal Múltiple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Entrenamos nuestro modelo con los datos de entrenamiento y utilizando la función fit(X , y).

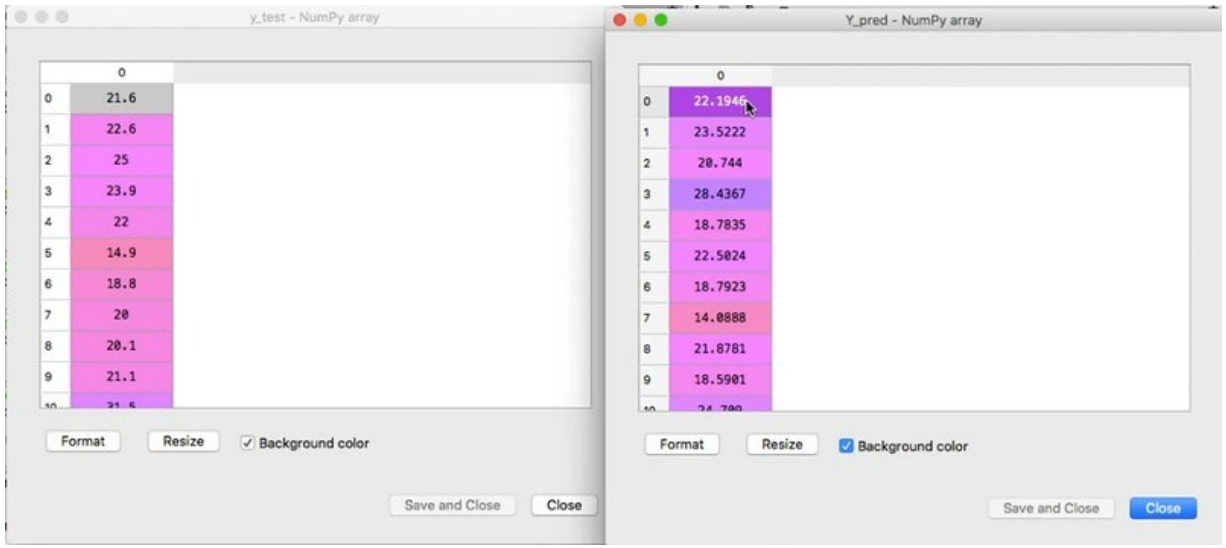
```
#Entrenamiento del modelo
lr_multiple.fit(X_train, y_train)
```

Y finalmente realizamos la predicción utilizando los datos de prueba. Utilizaremos para ello la función **.predict(X)** de nuestro objeto **lr**
Es obligatorio el uso de la función fit en el paso anterior, para que no de un error.

```
#Realizo una predicción
Y_pred_multiple = lr_multiple.predict(X_test)
```

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

En este caso también hacemos una comparación rápida los valores obtenidos en nuestra predicción con los valores reales, para ver que tal es el comportamiento de nuestro modelo. Para ello usaremos el Explorador de variables de Spyder.



A la izquierda están los datos reales, que separamos en **y_test**. A la derecha están los datos obtenidos de la predicción **Y_pred_multiple**. Al igual que en modelo de regresión simple, algunos de los datos reales se aproximan a los calculados, mientras que otras veces son muy diferentes. Por lo tanto, este modelo no es el más adecuado de implementar para este conjunto de datos.

Scikit-Learn. Regresión lineal Múltiple

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Calculemos ahora los valores de los términos independientes, para ver como queda la ecuación del modelo. Para ello accedemos al array de numpy **.coef_[ ]** y de la constante o intersección **.intercept_** de nuestro objeto **"lr_multiple".** ($y = a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + b$)

```
#Valores de la ecuación del modelo de regresión
print('VALORES DEL MODELO DE REGRESIÓN LINEAL MÚLTIPLE \n')
a1 = lr_multiple.coef_[0]
a2 = lr_multiple.coef_[1]
a3 = lr_multiple.coef_[2]
constante = lr_multiple.intercept_
print('La ecuación del modelo es:')
print('y = %.3f x1 %+.3f x2 %+.3f x3 %+.3f '  %(a1,a2,a3,constante))
```

```
VALORES DEL MODELO DE REGRESIÓN LINEAL MÚLTIPLE

La ecuación del modelo es:
y = 8.173 x1 -0.093 x2 -0.464 x3 -20.568
```

Scikit-Learn. Regresión lineal Múltiple Ejemplo Práctico. *"Precio de las Casas de Boston"*

Un último paso que vamos a realizar es calcular la precisión del algoritmo. Para ello empleamos el método **.score(X , y)** de nuestro objeto de regresión lineal. En X e y le pasamos las variables independientes y salidas del conjunto de entrenamiento. Nos devuelve el resultado de la estadística R^2.

```
#### precisión del modelo
print('Precisión del modelo: ',end='')
print('%.3f' % lr_multiple.score(X_train, y_train))
```

```
Precisión del modelo: 0.524
```

El resultado obtenido es de 0,524, que mejora un poco la de regresión lineal simple pero sigue sin ser muy buena

De nuevo observamos que este algoritmo no es el mejor para este conjunto de datos.

En conclusión podemos decir que este método es muy parecido al de Regresión Lineal Simple, la única gran diferencia es que los datos de X contendrán más de una columna o característica de nuestro dataset.

Scikit-Learn

Contenidos :

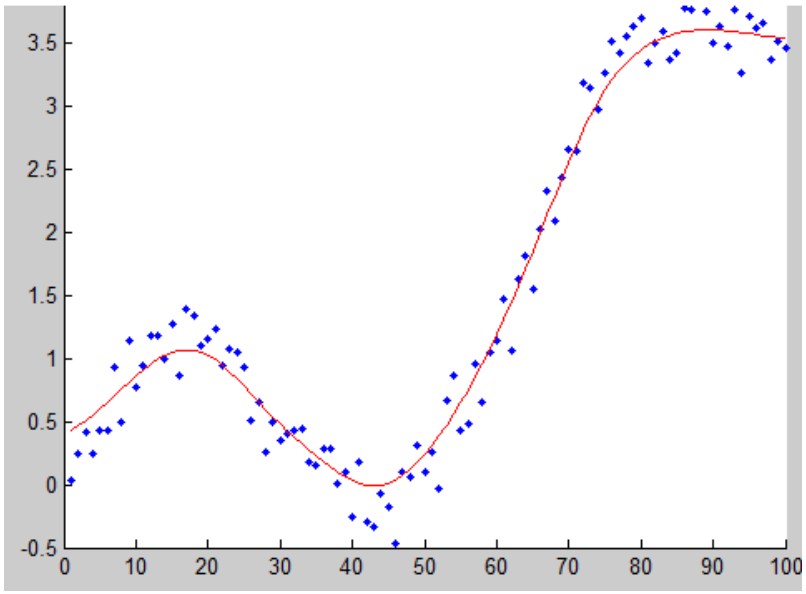
- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn. Regresión Polinómica.

- Cuando nuestra distribución de datos es más compleja que una recta, y debemos crear una curva para ajustar datos no lineales.

  La regresión polinómica es un caso especial de análisis de regresión lineal múltiple en la que la relación entre la variable independiente **x,** y la variable dependiente **y**, se modela como un polinomio de grado “n” en x. Este enfoque proporciona una forma sencilla de darle un ajuste no lineal a los datos

$$y = a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n + b$$



La siguiente imagen muestra el ajuste de una curva mediante polinomio de grado 3

$y = a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + b$

Las variables independientes que resultan de la expansión polinómica, se conocen como “términos de mayor grado”. Se han usado para describir fenómenos no lineales como la tasa de progresión de enfermedades epidémicas.

Una Regresión Polinómica de grado dos con dos variables sería de la forma:

$y = a_1 \cdot x_1 + a_2 \cdot x_1^2 + a_3 \cdot x_2 + a_4 \cdot x_2^2 + a_5 \cdot x_1 \cdot x_2 + b$

Scikit-Learn. Regresión Polinómica.

- Esto hace que sea una forma agradable y directa de modelar curvas sin tener que implementar modelos complicados no lineales.
- Un patrón común dentro de Machine Learning es usar modelos lineales entrenados en funciones no lineales de los datos. Al considerar los ajustes lineales dentro de un espacio de mayor dimensión construido con estas funciones básicas, el modelo tiene la flexibilidad de adaptarse a una gama de datos mucho más amplia.
- Desafortunadamente, la Regresión Polinómica también tiene un buen número de problemas. A medida que aumentamos el número de características, también aumenta la complejidad de la fórmula, lo que a veces es difícil de manejar.
- Además, la Regresión Polinómica tiene una tendencia a ajustarse drásticamente, incluso es un simple conjunto de datos unidimensional. Si bien podamos tener una tentación de ajustar un polinomio de mayor grado para obtener un error menor, esto puede redundar en un ajuste excesivo. Por esa razón siempre se debe trazar las relaciones para ver el ajuste y concentrarnos en que la curva se ajuste a la naturaleza del problema.
- Hay otros problemas con la Regresión Polinómica, por ejemplo: es inherentemente no local, es decir, cambiar el valor de “y” en un punto del conjunto de entrenamiento puede afectar el ajuste del polinomio para puntos de datos que están muy lejos. Por lo tanto, para evitar el uso de polinomios de alto grado en todo el conjunto de datos, podemos sustituirlo con muchas funciones de pequeño grado diferentes.

Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

De nuevo vamos a desarrollar un modelo para predecir el precio de las casas en Boston, teniendo en cuenta el número de habitaciones de la vivienda. Para ello usaremos un algoritmo de Regresión Polinómica, Como en los casos de regresión anteriores, seguiremos una rutina de pasos parecida

Como ya tenemos hechos los primeros pasos en los apartados anteriores, no necesitamos repetirlo todo otra vez. Iremos directos a parte diferente, saltando los pasos anteriores

7º Preparar los datos para REGRESIÓN POLINÓMICA

Solamente se requiere una variable independiente, que en nuestro caso será la correspondiente al número de habitaciones (**RM**), Recuperamos un array de Numpy de **.data[ ]** la posición 5, concretamente (desde 0)

```
#Seleccionamos la columna 6 del dataset
X_p = boston.data[ : , np.newaxis , 5]
```

En este caso llamaremos "X_p" a nuestro array de numpy para Polynomial Regression.

Nota: no olvidar las 2D de array X_p

Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

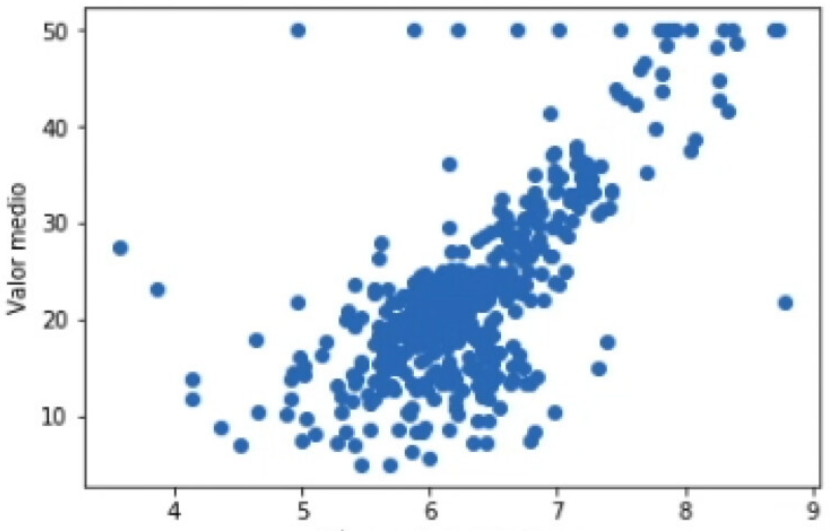
8º Seguidamente definimos los datos de "y" o "y_p", para este ejemplo, los cuales serán igual a la columna de target de nuestro dataset.

Plotearemos los datos utilizando una gráfica de dispersión para ver la distribución de los mismos. Como podemos observar los datos no se encuentran distribuidos de manera lineal en la parte inferior izquierda por lo que perfectamente podemos utilizar el algoritmo de Regresión Polinómica.

```
#Definir  los datos correspondientes a las etiquetas
#variable y dependiente u objetivo
y_p = boston.target

#Representamos gráficamente los datos correspondientes
plt.scatter(X_p, y_p)
plt.show()
```

A simple vista ya podemos deducir que el modelo vamos a construir tendrá un error muy alto.



Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

9º Separamos los conjuntos de entrenamiento y prueba.

Importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn**.

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos
from sklearn.model_selection import train_test_split

X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X_p, y_p, test_size=0.2)
```

Es muy importante y por ello se diferencia de los algoritmos de regresión anteriores, establecer el grado del polinomio, para ello se importa desde el módulo **preprocessing** de sk-Learn la clase **PolynomialFeatures**, y procedemos a definir el grado. Usaremos grado 2. Haz pruebas con diferentes grados, lo ideal es comenzar con un valor bajo como el 2 y después ir subiendo poco a poco para ver si se mejora lo resultados de la predicción, pero debes tener cuidado para no ajustar de más el modelo ocasionando un sobreajuste u overfitting.

```
#Se define el grado del polinomio
from sklearn.preprocessing import PolynomialFeatures
poli_reg = PolynomialFeatures(degree = 2)
```

Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Definido el grado, se procede a transformar las características existentes, en este caso tanto X_train_p , como X_test_p en características de mayor grado, para ello utilizamos el objeto polinomio definido anteriormente y lo entrenamos a estos datos mediante el método **.fit_transform( )**

```
#Se transforma las características existentes en características de mayor grado
X_train_poli = poli_reg.fit_transform(X_train_p)
X_test_poli = poli_reg.fit_transform(X_test_p)
```

En este punto lo que estamos haciendo es calculando los valores de "X" para cada uno de los elementos dentro de la formula, ya sabemos que "X" estaba elevado a la n potencia dependiendo del valor del coeficiente del polinomio. En este caso n es 2

$$y = a_1 \cdot x + a_2 \cdot x^2 + b$$

Invocaremos la creación de nuestro objeto para regresión polinómica, usando la función **.LinearRegression()** , del módulo importado anteriormente **"linear_model"**

```
#Defino el algoritmo a utilizar
pr = linear_model.LinearRegression()
```

Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

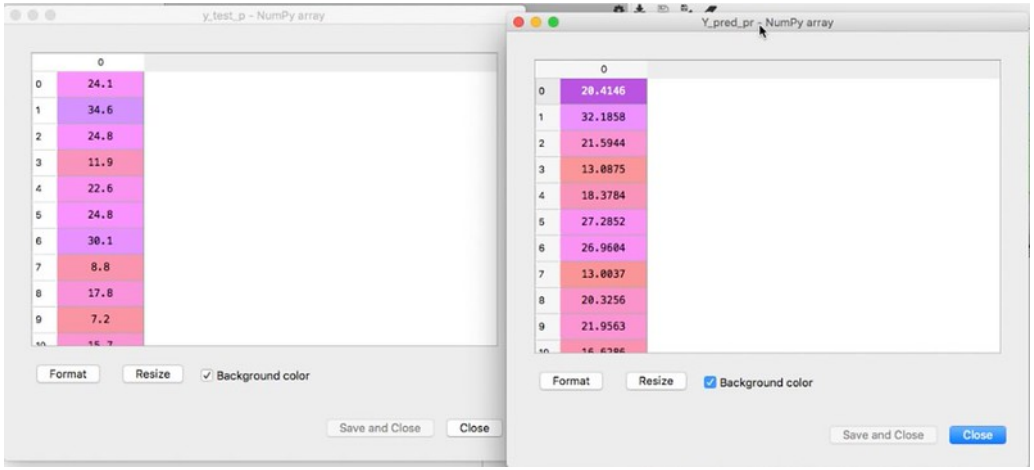
Definido el algoritmo lo utilizamos para entrenar los datos, mediante la función **fit(X,y)**. Recuerda que ahora le pasamos un array de datos que transformamos con las características de mayor grado.

```
#Entrenamiento del modelo
pr.fit(X_train_poli, y_train_p)
```

Realizado todo esto comparemos los valores obtenidos en nuestra predicción con los valores reales, para ver que tal es el comportamiento de nuestro modelo.

```
#Realizo una predicción
Y_pred_pr = pr.predict(X_test_poli)
```

Con esta pequeña comparación podemos determinar que nuestro modelo no es del todo eficiente, pero veámoslo gráficamente para detallarlo con más cuidado, para ello utilizamos la librería de matplotlib.



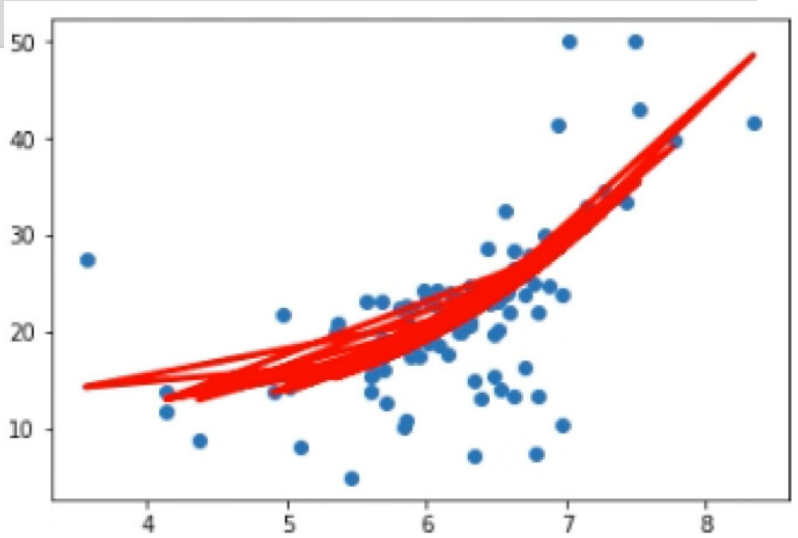
Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Como podemos observar en la gráfica, la línea roja sería nuestro modelo y los puntos azules son los datos de pruebas con los que lo entrenamos.

```
# Representación gráfica de los datos de prueba y el modelo
plt.scatter(X_test_p, y_test_p)
plt.plot(X_test_p, Y_pred_pr, color='red', linewidth=3)
#plt.scatter(X_test_p, Y_pred_pr, color='green')
plt.show()
```

Igual que todos los algoritmos de regresión que hemos desarrollado hasta ahora el objetivo de este es que el modelo abarque el mayor número de muestras y de esta forma disminuir el error que vendría siendo la distancia desde la línea hasta el punto.Todos los datos fuera de la línea serían errores.



Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Calculemos ahora los coeficientes de la formula de Regresión Polinómica. Para ello accedemos a los atributos de la pendiente **.coef_** y de la constante o intersección **.intercept_** de nuestro objeto **"pr".** $(y = a_1 \cdot x + a_2 \cdot x^2 + b)$

```
#Valores del modelo de regresión polinómica
print('VALORES DEL MODELO DE REGRESIÓN POLINÓMICA \n')
print(pr.coef_)
constante = pr.intercept_
print('Valor de la intersección o coeficiente "b":')
print(pr.intercept_)
```

```
VALORES DEL MODELO DE REGRESIÓN POLINÓMICA

[  0.        -24.45383534   2.6029966 ]
Valor de la intersección o coeficiente "b": 71.80083274325145
```

Los coeficientes de la formula de Regresión Polinómica dan como resultado: -24.4538 para a1, 2.6029 para a2. Ignora el primer valor del array = 0, siempre se obtine este valor con este algoritmo.

El valor de la intersección o coeficiente "b", este dato será igual a: 71.8008

$$y = : -24.4538 \cdot x + 2.6029 \cdot x^2 + 71.8008$$

Scikit-Learn. Regresión Polinómica

Ejemplo Práctico. *"Precio de las Casas de Boston"*

Ahora calculemos la precisión del algoritmo empleando el método **.score(X,y)**. En X e y le pasamos las variables independientes y salidas del conjunto de entrenamiento. Nos devuelve el resultado de la estadística R^2.

```
#### precisión del modelo
print('Precisión del modelo: ',end='')
print('%.3f' % pr.score(X_train_poli, y_train_p))
```

```
Precisión del modelo: 0.577
```

Como ya lo habíamos visto comparando los resultados de "y_predict" y viendo la gráfica podíamos deducir que la precisión no iba a ser muy buena.

Recuerda que esto no significa que el algoritmo sea malo sino que no es el mejor para este conjunto de datos.

Scikit-Learn. Vectores Soporte Regresión (SVR)

- Este algoritmo se basa en buscar la curva o hiperplano que modele la tendencia de los datos de entrenamiento y según ella predecir cualquier dato en el futuro.
- Esta curva siempre viene acompañada con un rango (máximo margen), tanto del lado positivo como en el negativo, el cual tiene el mismo comportamiento o forma de la curva. Los vectores de soporte hacen referencia a un pequeño subconjunto de las observaciones de entrenamiento que se utilizan como soporte para la ubicación óptima de la superficie de decisión.
- Todos los datos que se encuentren fuera del rango son considerados errores por lo que es necesario calcular la distancia entre el mismo y los rangos. Esta distancia se conoce por **ξ** y afecta la ecuación final del modelo.
- El objetivo de los Vectores de Soporte Regresión es individualizar el hiperplano maximizando el margen, teniendo en cuenta que se tolera parte del error.

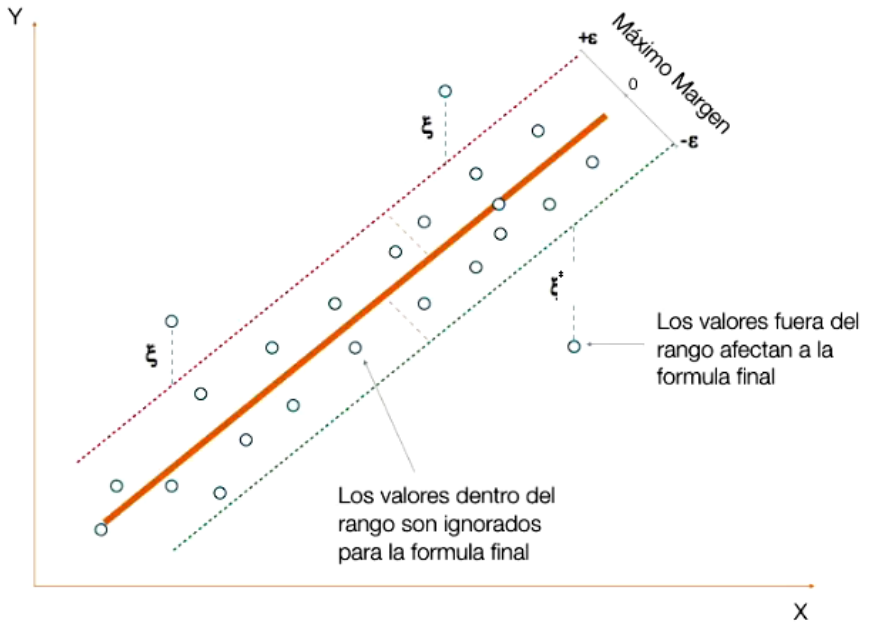
$$min\frac{1}{2} \parallel w \parallel^2 + C\sum_{i=1}^{N}(\xi + \xi^*)$$

maximizar margen — minimizar error entrenamiento

w es la magnitud del vector o hiperplano

C es una constante y debe ser mayor a 0, determina el equilibrio entre la regularidad de la función y la cuantía hasta la cual toleramos desviaciones mayores que las bandas de soporte.

ξ y ξ* son las variables que controlan el error cometido por la función de regresión al aproximar a las bandas.



Scikit-Learn. Vectores Soporte Regresión (SVR)

- Si el valor de la constante C es muy grande, en el caso límite C tiende a infinito , estaríamos considerando que el conjunto está perfectamente representado por nuestro hiperplano predictor, ξ tiende a 0. Al contrario, un número demasiado pequeño para C permitirá valores de ξ elevados, es decir, estaríamos admitiendo un número muy elevado de ejemplos mal representados.
- Para datos no lineales el procedimiento es exactamente igual, la diferencia es que se implementa un Kernel para convertir los datos lineales. EL truco del kernel nos permite operar en el espacio de la característica original, sin calcular las coordenadas de los datos, en un espacio dimensional superior.

Brevemente, algunos tipos de kernel:

Kernel lineal: puede ser utilizado como el producto normal de dos observaciones dadas. El producto entre dos vectores es la suma de la multiplicación de cada par de valores de entrada.

$$K(x,xi) = sum(x * xi)$$

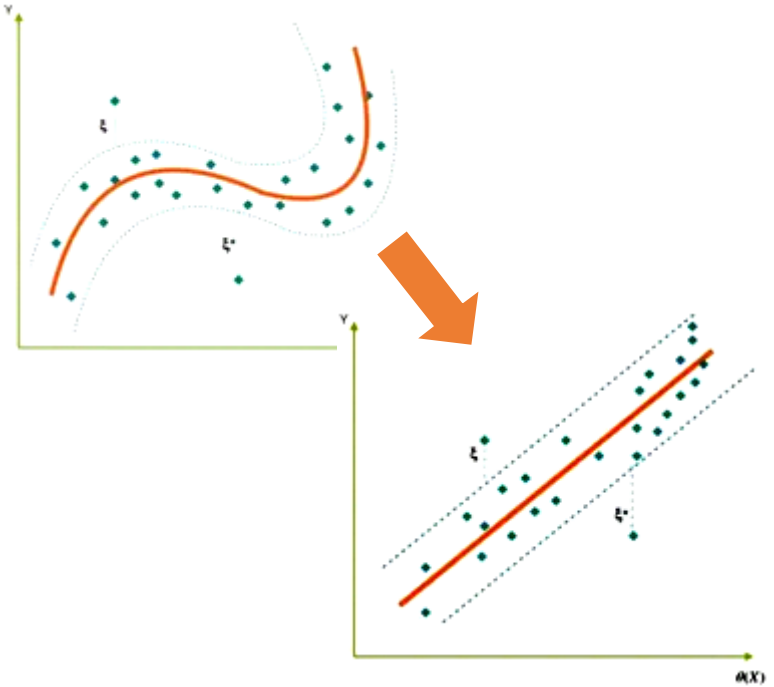
Kernel polinomial: este es una forma más generalizada del lineal. El kernel polinomial no solo observa las características dadas por las muestras de entrada para determinar su similitud, sino también las combinaciones de éstas. Con “n” características originales y “d” grados de polinomio produce n^d características expandidas.

$$K(x,xi) = 1 + sum(x * xi)^d$$

Kernel de base radial o RBF: también llamado kernel gaussiano. RBF puede mapear un espacio de entrada en un espacio dimensional infinito.

$$K(x,xi) = exp(- gamma * sum(x * xi)^2)$$

Donde gamma es un parámetro ente 0 y 1. Un valor más alto de gamma encajará perfectamente en el conjunto de datos de entrenamiento, lo que provoca un ajuste excesivo. Gamma igual a 0.1 se considera un buen valor por defecto.



Scikit-Learn. Vectores Soporte Regresión (SVR)

- El rendimiento del algoritmo de Vectores de Soporte Regresión depende de una buena configuración de los parámetros C y de los del Kernel. El problema de la selección óptima de parámetros se complica aún más por el hecho de que la complejidad del modelo. Por tal motivo usamos librerías de programación como Sciket-Learn que implementan este algoritmo para usarlo cómodamente .
- Scikit Learn cuenta con un modulo llamado **sklearn.svm** donde se incluye todo lo referente al algoritmo de Vectores de Soporte tanto clasificación como regresión. Para esta entrada nos enfocaremos solamente en regresión y más específicamente en SVR.

  class sklearn.svm.SVR(kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)

SVR se puede configurar con ciertos parámetros (ver documentación para más detalles), los más importantes que se deben considerar al momento de configurar el modelo son:

- C, por defecto es de 1 pero tu lo puedes definir según conveniencia.
- Otro parámetro es el de "épsilon", aunque se dejan por defecto, sobretodo si estás empezando en Machine Learning .
- El tipo de Kernel que se maneja: lineal, que se utiliza cuando los datos son lineales, y para los datos no lineales se cuenta con "poly", "rbf" y "sigmoid". Si este parámetro no se configura al momento de utilizar este algoritmo, se utilizará "rbf" como Kernel por defecto.

Scikit-Learn. Vectores Soporte Regresión (SVR)

Ejemplo Práctico. *"Precio de las Casas de Boston"*

De nuevo vamos a desarrollar un modelo para predecir el precio de las casas en Boston, teniendo en cuenta el número de habitaciones de la vivienda. En esta ocasión mediante un algoritmo SVR. Como en los casos de regresión anteriores, seguiremos una rutina de pasos parecida

Como ya tenemos hechos los primeros pasos en los apartados anteriores, no necesitamos repetirlo todo otra vez. Iremos directos a parte diferente, saltando los pasos anteriores

7º Preparar los datos para SVR

Solamente se requiere una variable independiente, que en nuestro caso será la correspondiente al número de habitaciones (**RM**), Recuperamos un array de Numpy de **.data[ ]** la posición 5, concretamente (desde 0)

```
#Seleccionamos la columna 6 del dataset
X_svr = boston.data[ : , np.newaxis , 5]
```

En este caso llamaremos "X_svr" a nuestro array de numpy para SVR.

Nota: no olvidar las 2D de array X_svr

Scikit-Learn. Vectores Soporte Regresión (SVR) Ejemplo Práctico. *"Precio de las Casas de Boston"*

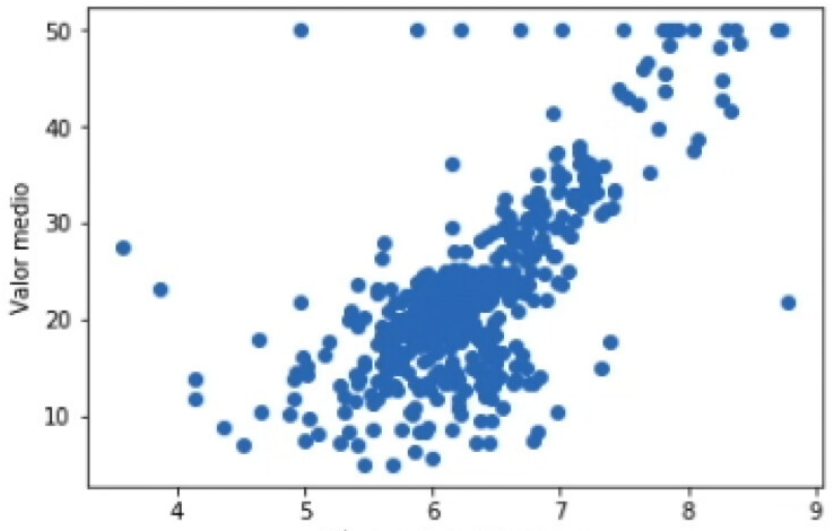
8º Seguidamente definimos los datos de "y" o "y_svr", para este ejemplo, los cuales serán igual a la columna de target de nuestro dataset.

Plotearemos los datos utilizando una gráfica de dispersión para ver la distribución de los mismos.

Como podemos observar estos datos tienen un comportamiento lineal, por lo que implementaremos el algoritmo de Vectores de Soporte Regresión, con un kernel lineal.

```
#Definir  los datos correspondientes a las etiquetas
#variable y dependiente u objetivo
y_svr = boston.target

#Representamos gráficamente los datos correspondientes
plt.scatter(X_svr, y_svr)
plt.show()
```



Scikit-Learn. Vectores Soporte Regresión (SVR) Ejemplo Práctico. *"Precio de las Casas de Boston"*

9º Separamos los conjuntos de entrenamiento y prueba.

Importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn**.

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_svr, y_svr  test_size = 0.2)
```

Invocaremos la creación de nuestro objeto para SVR desde el módulo de svm de sklearn. Le pasaremos los parámetros que hemos estado comentando.

```
#Defino el algoritmo a utilizar
from sklearn.svm import SVR
svr = SVR(kernel='linear', C=1.0, epsilon=0.2)
```

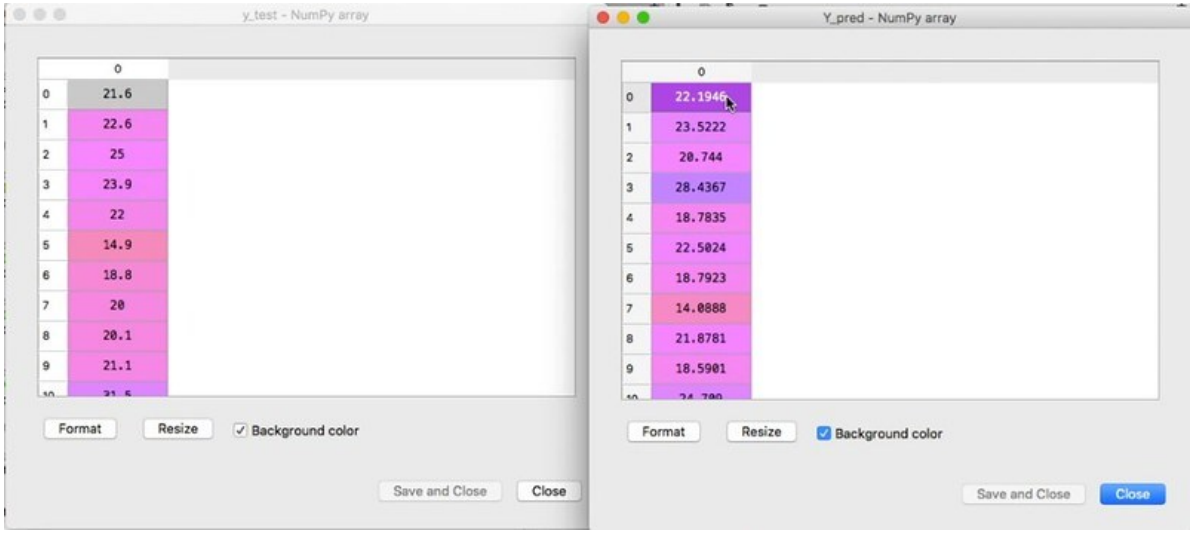
Scikit-Learn.Vectores Soporte Regresión (SVR) Ejemplo Práctico. *"Precio de las Casas de Boston"*

Definido el algoritmo lo utilizamos para entrenar los datos, mediante la función **fit(X,y)**.

```
#Entrenamiento del modelo
svr.fit(X_train, y_train)
```

Realizado todo esto comparemos los valores obtenidos en nuestra predicción con los valores reales, para ver que tal es el comportamiento de nuestro modelo.

```
#Realizo una predicción
Y_pred = svr.predict(X_test)
```

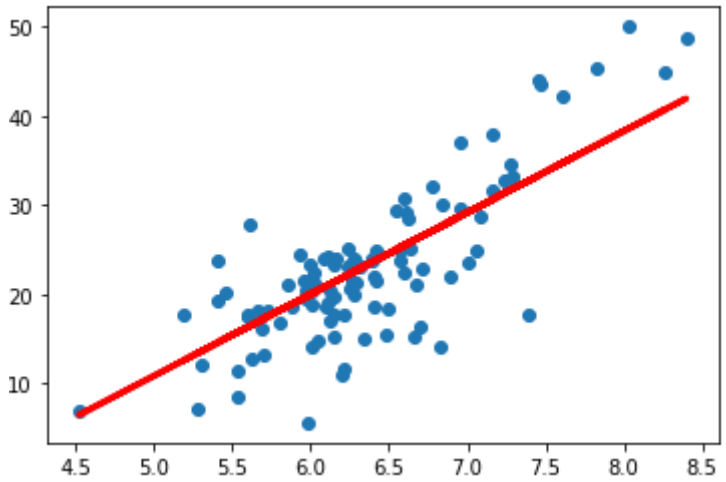


Scikit-Learn.Vectores Soporte Regresión (SVR) Ejemplo Práctico. *"Precio de las Casas de Boston"*

En la gráfica, la línea roja sería nuestro modelo y los puntos azules son los datos de pruebas con los que lo entrenamos.

```
# Representación gráfica de los datos de prueba y el modelo
plt.scatter(X_test, y_test)
plt.plot(X_test, Y_pred, color='red', linewidth=3)
plt.show()
```

Como definimos, al momento de configurar el algoritmo, el kernel que estamos utilizando es lineal por lo que el modelo es una línea, la cual va a tratar de cubrir la mayor cantidad de datos. Pero si observamos la línea tiene un comportamiento similar a los datos de entrenamiento.



Scikit-Learn. Vectores Soporte Regresión (SVR) Ejemplo Práctico. *"Precio de las Casas de Boston"*

Ahora calculemos la precisión del algoritmo empleando el método **.score(X,y)**. En X e y le pasamos las variables independientes y salidas del conjunto de entrenamiento.

```
#### precisión del modelo
print('Precisión del modelo: ',end='')
print('%.3f' % svr.score(X_train, y_train))
```

```
Precisión del modelo: 0.524
```

Este valor es muy parecido a lo que hemos obtenido con los otros algoritmos que hemos evaluado anteriormente.

Scikit-Learn

Contenidos :

- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn. Decision Tree Regression

- Los árboles de decisión son una técnica que predice valores de respuestas mediante el aprendizaje de reglas de decisión derivadas de características. Se pueden utilizar tanto en una regresión como en un contexto de clasificación.
- Se consideran uno de los mejores y más utilizados métodos de aprendizaje supervisado. Potencian modelos predictivos con alta precisión, estabilidad y facilidad de interpretación.
- Los árboles de decisión funcionan al dividir el espacio de la característica en varias regiones rectangulares simples, divididas por divisiones paralelas de ejes. Para obtener una predicción para una observación particular, se utiliza la media o el modo de las respuestas de las observaciones de entrenamiento, dentro de la partición a la que pertenece la nueva observación.
- Este algoritmo es muy bueno en el manejo de datos tabulares con características numéricas o características categóricas con menos de cientos de categorías. A diferencia de los modelos lineales, los árboles de decisiones pueden capturar la interacción no lineal entre las características y el objetivo teniendo en cuenta que se tolera parte del error.

$$f(x) = \sum_{m=1}^{M} w_m \phi(x; v_m)$$

Donde:

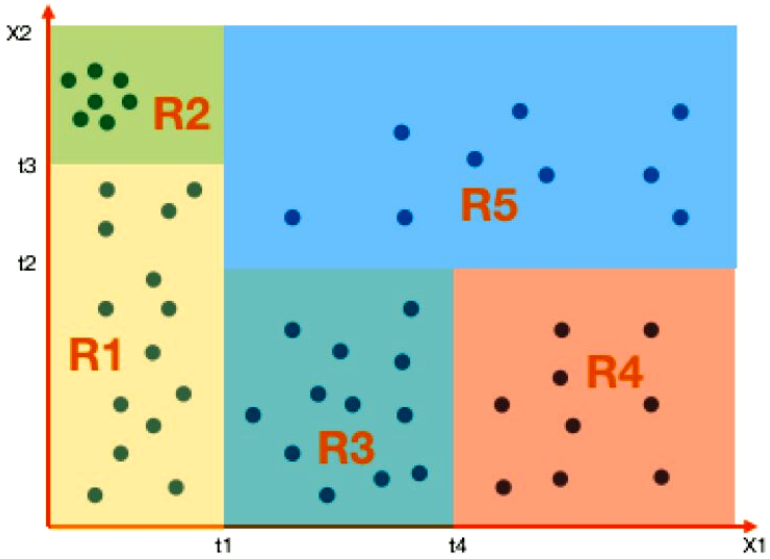
w_m es la respuesta media en una región particular (Rm).

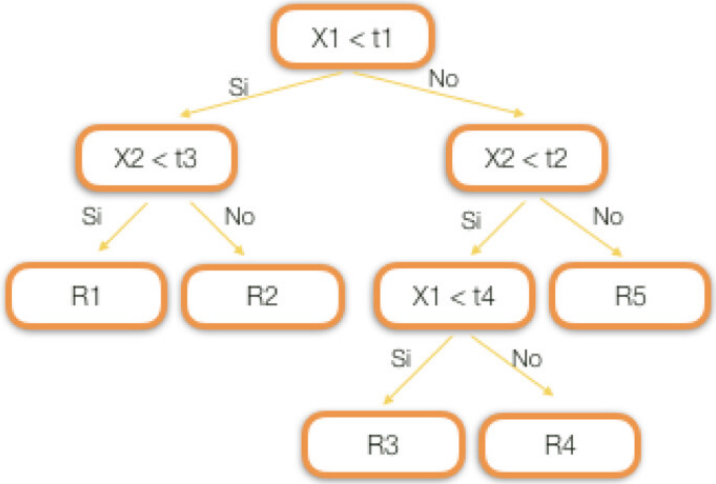
v_m representa cómo se divide cada variable en un valor de umbral particular.

Estas divisiones definen cómo el espacio de características en R^2 en M regiones separadas o hiperbloques.

Scikit-Learn. Decision Tree Regression

- En la siguiente imagen tenemos un ejemplo con dos variables de características (X1 y X2) y una respuesta numérica “y”.
- Se observa cómo se divide el dominio mediante divisiones paralelas de eje, es decir, cada división del dominio se alinea con uno de los ejes de características.
- El concepto de división paralela de ejes se generaliza directamente a dimensiones superiores a dos. Para un espacio de características de tamaño p, un subconjunto del espacio se divide en regiones M, cada una de las cuales es un hiperbloque p-dimensional.





Scikit-Learn. Decision Tree Regression

- ¿Cómo se forma la partición de una manera algorítmica?. Para eso necesitamos usar una técnica conocida como división binaria recursiva.
- Nuestro objetivo para este algoritmo es minimizar algún tipo de criterio de error. Deseamos minimizar la suma de cuadrados residual (RSS), una medida de error también utilizada en la configuración de regresión lineal.
- Se aborda el problema comenzando en la parte superior del árbol y dividiendo el árbol en dos ramas, lo que crea una partición de dos espacios. Se lleva a cabo esta división en particular en la parte superior del árbol varias veces y elige la división de las características que minimiza la suma de cuadrados residual (RSS).
- En este punto, el árbol crea una nueva rama en una partición particular y lleva a cabo el mismo procedimiento, es decir, evalúa el RSS en cada división de la partición y elige el mejor.
- Ahora, el principal problema con el árbol de decisión es que es propenso a sobreajuste. Podríamos crear un árbol que pudiera clasificar los datos a la perfección sin que quede ningún atributo para dividir. Esto funcionaría bien en el conjunto de datos de entrenamiento, pero tendrá un mal resultado en el conjunto de datos de prueba. Existen dos enfoques populares para evitar esto en los árboles de decisión: detener el crecimiento del árbol antes de que sea demasiado grande o podar el árbol después de que sea demasiado grande.

Scikit-Learn. Decision Tree Regression

Las ventajas que tiene este tipo de algoritmo son:

- Fácil de entender. La salida del árbol de decisión es muy fácil de entender, incluso para personas con antecedentes no analíticos, no se requiere ningún conocimiento estadístico para leerlos e interpretarlos.
- Útil en la exploración de datos. El árbol de decisiones es una de las forma más rápidas para identificar las variables más significativas y la relación entre dos o más. Con la ayuda de los árboles de decisión podemos crear nuevas variables o características que tengan mejor poder para predecir la variable objetivo.
- Se requiere menos limpieza de datos. Requiere menos limpieza de datos en comparación con algunas otras técnicas de modelado. A su vez, no esta influenciado por los valores atípicos y faltantes en la data.
- El tipo de datos no es una restricción. Puede manejar variables numéricas y categóricas.
- Método no paramétrico. Esto significa que los árboles de decisión no tienen suposiciones sobre la distribución del espacio y la estructura del clasificador.

Desventajas que posee este algoritmo:

- Sobreajuste. Es una de las dificultades más comunes que tiene este algoritmo, este problema se resuelve colocando restricciones en los parámetros del modelo y eliminando ramas en el análisis.
- No apto para variables continuas. Al trabajar con variables numéricas continuas, el árbol de decisiones pierde información cuando categoriza variables en diferentes categorías.
- Los modelos basados en árboles no están diseñados para funcionar con características muy dispersas. Cuando se trata de datos de entrada dispersos (por ejemplo, características categóricas con una gran dimensión), podemos preprocesar las características dispersas para generar estadísticas numéricas, o cambiar a un modelo lineal, que es más adecuado para dichos escenarios.

Scikit-Learn. Decision Tree Regression

- Scikit Learn cuenta con un modulo llamado **sklearn.tree** donde se incluye todo lo referente al algoritmo de Arboles de Decisión, por lo que para implementarlo deberás importar este modulo **sklearn.tree**. Posteriormente utilizaremos la clase que en este caso será **DecisionTreeRegressor**.

  class sklearn.tree.DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False

Veamos un ejemplo de los paármetros más importante a considerar en el momento de configurar el modelo.

- "criterion", el método usado para las divisiones de las ramas del árbol. La suma de cuadrados residual es computacionalmente muy costosa. Scikit-Learn utiliza por defecto la media del error cuadrado(MSE) o el error absoluto promedio (MAE), a partir de la versión 0.18.
- "splitter", que es la estrategia utilizada para la división en cada nodo, en esta caso se cuenta con dos opciones "best" que sería la mejor división, y "random" que elige de manera aleatoria la separación. Por defecto se encuentra seleccionada la opción "best" que por supuesto es la mejor opción.
- "max_depth", que se refiere la profundidad máxima del árbol. Por defecto, el árbol se expande hasta que todas las hojas estén puras o hasta que todas las hojas contengan menos datos. Si hacemos esto es muy probable que el algoritmo caiga en sobreajuste, por lo que es recomendable probar varios valores

Scikit-Learn. Decision Tree Regression

Ejemplo Práctico. *"Precio de las Casas de Boston"*

De nuevo vamos a desarrollar un modelo para predecir el precio de las casas en Boston, teniendo en cuenta el número de habitaciones de la vivienda. En esta ocasión mediante un algoritmo para Árboles de Decisión Regresión. Como en los casos de regresión anteriores, seguiremos una rutina de pasos parecida

Como ya tenemos hechos los primeros pasos en los apartados anteriores, no necesitamos repetirlo todo otra vez. Iremos directos a parte diferente, saltando los pasos anteriores

7º Preparar los datos para Árboles de Decisión Regresión

Solamente se requiere una variable independiente, que en nuestro caso será la correspondiente al número de habitaciones (**RM**), Recuperamos un array de Numpy de **.data[ ]** la posición 5, concretamente (desde 0)

```
#Seleccionamos la columna 6 del dataset
X_adr = boston.data[ : , np.newaxis , 5]
```

En este caso llamaremos "X_adr" a nuestro array de numpy para Árboles de Desión R.

Nota: no olvidar las 2D de array X_svr

Scikit-Learn. Decision Tree Regression

Ejemplo Práctico. *"Precio de las Casas de Boston"*

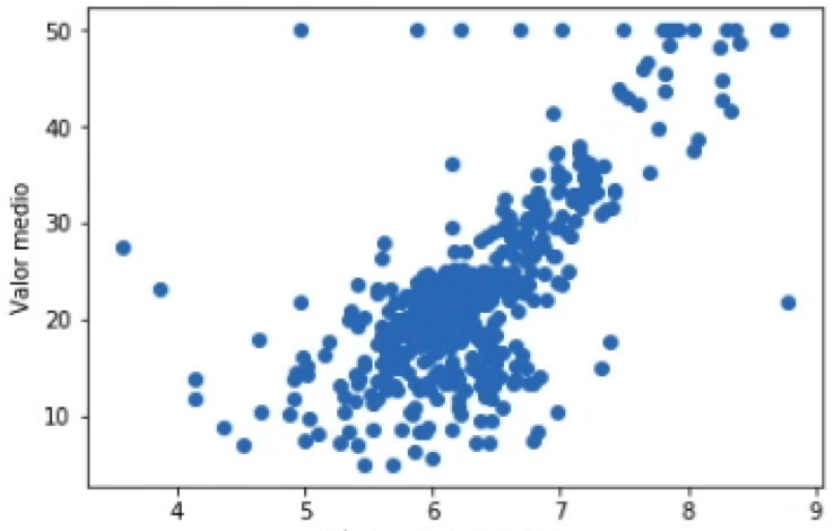
8º Seguidamente definimos los datos de "y" o "y_adr", para este ejemplo, los cuales serán igual a la columna de target de nuestro dataset.

Plotearemos los datos utilizando una gráfica de dispersión para ver la distribución de los mismos.

Como podemos observar es un conjunto de datos que se encuentra distribuido de manera uniforme..

```
#Definir  los datos correspondientes a las etiquetas
#variable y dependiente u objetivo
y_adr = boston.target

#Representamos gráficamente los datos correspondientes
plt.scatter(X_adr, y_adr)
plt.show()
```



Scikit-Learn. Decision Tree Regression

Ejemplo Práctico. *"Precio de las Casas de Boston"*

9º Separamos los conjuntos de entrenamiento y prueba.

Importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn**.

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_adr, y_adr  test_size = 0.2)
```

Invocaremos la creación de nuestro objeto para Árboles de Decisión R. desde el módulo de tree de sklearn. Le pasaremos el parámetro max_depth, dejando los demás por defecto.

```
#Defino el algoritmo a utilizar
from sklearn.tree import DecisionTreeRegressor
adr = DecisionTreeRegressor(max_depth = 5)
```

Jugar con este parámetro

Scikit-Learn. Decision Tree Regression Ejemplo Práctico. *"Precio de las Casas de Boston"*

Definido el algoritmo lo utilizamos para entrenar los datos, mediante la función **fit(X,y)**.

```
#Entrenamiento del modelo
adr.fit(X_train, y_train)
```

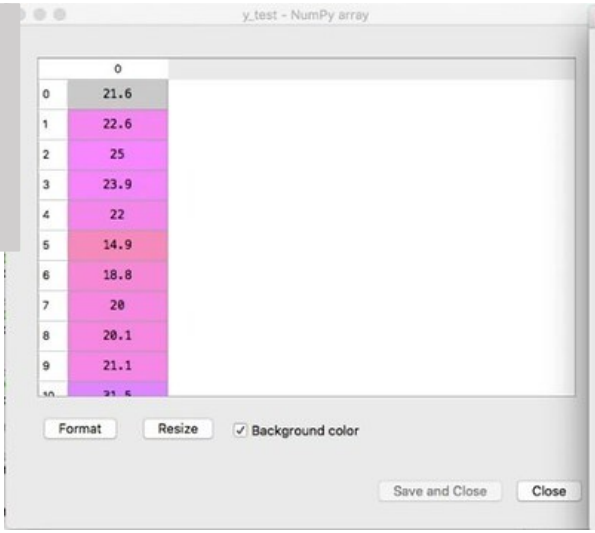
Realizado todo esto comparemos los valores obtenidos en nuestra predicción con los valores reales, para ver que tal es el comportamiento de nuestro modelo.

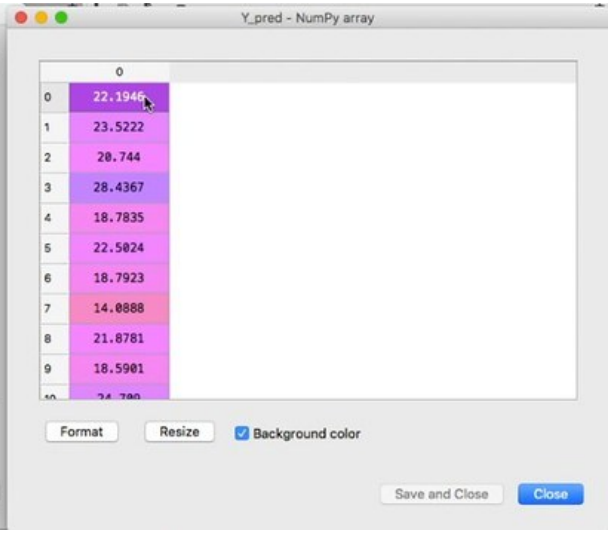
```
#Realizo una predicción

X_test_ord=np.sort(X_test,axis=0)

Y_pred = adr.predict(X_test_ord)
```

X_test se ha ordenado previamente





Scikit-Learn. Decision Tree Regression Ejemplo Práctico. *"Precio de las Casas de Boston"*

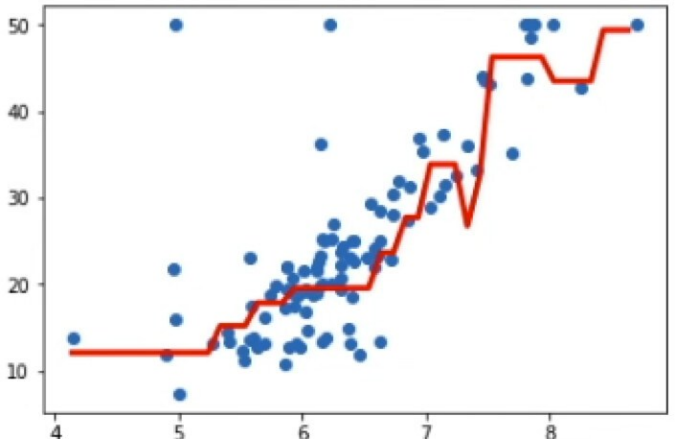
En la gráfica, la línea roja sería nuestro modelo y los puntos azules son los datos de pruebas con los que lo entrenamos.

En caso de no haber ordenado previamente los datos con sort, podría utilizar la instrucción arange para espaciar de manera uniforme los datos, el intervalo a utilizar será el valor mínimo y máximo de los datos de prueba y los pasos para el espaciado será de 0.1.

```
# Representación gráfica de los datos de prueba y el modelo

plt.scatter(X_test, y_test)
plt.plot(X_test_ord, Y_pred, color='red', linewidth=1)
plt.show()
```

Como podemos observar, el modelo no es una línea recta como la habíamos visto en los anteriores algoritmos, el modelo cuenta con subidas y bajadas.



Scikit-Learn. Decision Tree Regression Ejemplo Práctico. *"Precio de las Casas de Boston"*

Ahora calculemos la precisión del algoritmo empleando el método **.score(X,y)**. En X e y le pasamos las variables independientes y salidas del conjunto de entrenamiento.

```
#### precisión del modelo
print('Precisión del modelo: ',end='')
print('%.3f' % adr.score(X_train, y_train))
```

```
Precisión del modelo: 0.722
```

Este valor mejora a lo que habíamos obtenido con los otros algoritmos que hemos evaluado anteriormente.

Realiza pruebas modificando la profundidad del árbol, para comprobar los resultados.

Scikit-Learn. Ventajas y desventajas de los algoritmos de regresión.

Regresión Lineal

Ventajas

- Fácil de entender y explicar, lo que puede ser muy valioso para las decisiones de negocios.
- Es rápido de modelar y es particularmente útil cuando la relación a modelar no es extremadamente compleja y no tiene mucha información.
- Es menos propenso al sobreajuste.

Desventajas

- No se puede modelar relaciones complejas.
- No se pueden capturar relaciones no lineales sin transformar la entrada, por lo que tienes que trabajar duro para que se ajuste a funciones no lineales.
- Puede sufrir con valores atípicos.

Son útiles

- Dar un primer vistazo a un conjunto de datos.
- Cuando se tiene datos numéricos con muchas características.
- Realizar predicción econométricas.
- Modelando las respuestas de marketing.

Scikit-Learn. Ventajas y desventajas de los algoritmos de regresión.

Vectores de Soporte Regresión

Ventajas

- Se pueden modelar relaciones complejas, no lineales.
- Robusto al ruido, esto se debe a que maximizan los márgenes.

Desventajas

- Necesidad de seleccionar una buena función de kernel.
- Los parámetros del modelo son difíciles de interpretar.
- Requiere memoria significativa y poder de procesamiento.
- Cuando se tiene muchos datos toma demasiado tiempo para entrenar.

Son útiles

- Clasificación de texto e imágenes.
- Reconocimiento de escritura a mano.

Scikit-Learn. Ventajas y desventajas de los algoritmos de regresión.

Árboles de Decisión Regresión

Ventajas

- Muy fácil de interpretar y entender.
- Rápido.
- Robusto al ruido y valores perdidos.
- Preciso
- Excelente para aprender relaciones complejas, altamente no lineales. Por lo general, pueden lograr un rendimiento bastante alto, mejor que la regresión polinómica.

Desventajas

- Los árboles complejos son difíciles de interpretar.
- Es posible la duplicación dentro del mismo subárbol.
- En ocasiones no es utilizado por ser un algoritmo tan sencillo y no tan poderoso para datos complejos.

Son útiles

- Diagnóstico médico.
- Análisis de riesgo crediticio

Scikit-Learn. Ventajas y desventajas de los algoritmos de regresión.

Bosques Aleatorios Regresión

Definición: Es una combinación árboles de decisión tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos.

Ventajas
- Puede trabajar en paralelo.
- Rara vez se sobreajusta.
- Maneja automáticamente los valores perdidos.
- No es necesario transformar ninguna variable.
- No hay necesidad de ajustar parámetros.
- Puede ser utilizado por casi cualquier persona con excelentes resultados.

Desventajas
- Difícil de interpretar.
- Más débil en la regresión al estimar valores en los extremos de la distribución de los valores de respuesta.
- Parcialmente en problemas multiclase hacia clases más frecuentes.

Son útiles
- Para casi cualquier problema de Machine Learning.
- Bioinformática.

Scikit-Learn

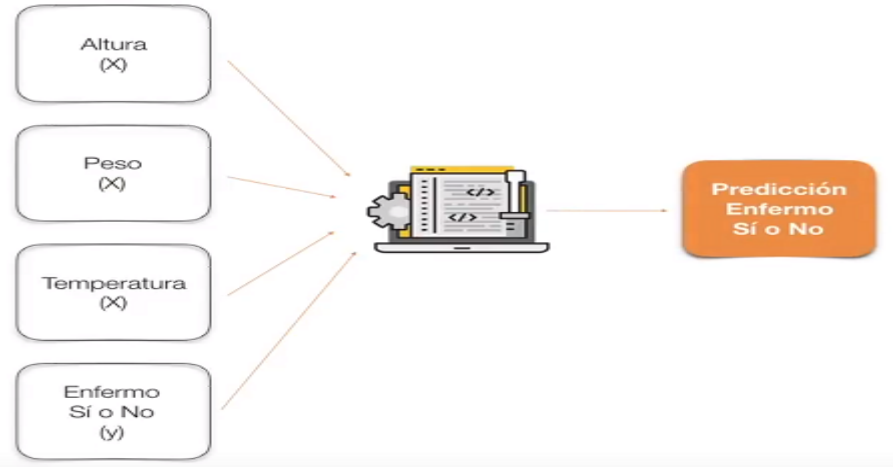
Contenidos :

- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
  - Ventajas y desventajas de los algoritmos de regresión
    - Regresión Lineal
    - Vectores de Soporte Regresión
    - Árboles de Decisión Regresión
    - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn. Regresión Logística

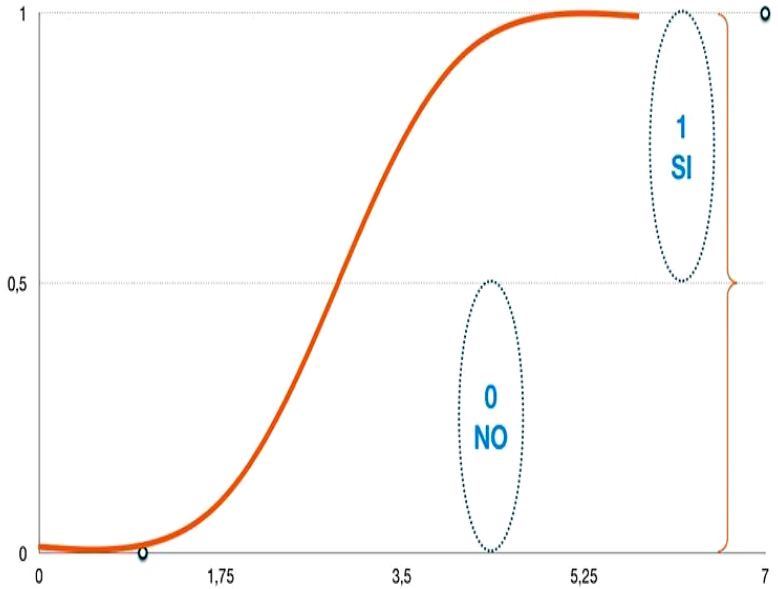
- Las técnicas de clasificación son una parte esencial de Machine Learning, ya que aproximadamente el 70% de los problemas son de clasificación.
- La regresión logística o Logistic Regression es un algoritmo de clasificación que se utiliza para predecir la probabilidad de una variable dependiente categórica.
- Diferencias entre Regresión Lineal y Regresión Logística. La Regresión Lineal proporciona una salida continua, pero la Regresión Logística proporciona una salida discreta.
- Hay muchos algoritmos de clasificación, pero la Regresión Logística es común y muy popular, después de Regresión Lineal. Se trata de un método útil y simple para resolver problemas de clasificación

Tipos de Regresión Logística
Regresión Logística Binaria: la variable objetivo tiene solo dos resultados posible, Llueve o NO Llueve, Sube o Baja.
Regresión Logística Multinomial: la variable objetivo tiene tres o más categorías nominales, como predecir el tipo de vino.
Regresión Logística Ordinal: la variable objetivo tiene tres o más categorías ordinales, como clasificar un restaurante o un producto del 1 al 5.



Scikit-Learn. Regresión Logística

La Regresión Logística es uno de algoritmos de Machine Learning más utilizados para la clasificación de dos clases. Es fácil de implementar y se puede usar como línea de base para cualquier problema de clasificación binaria. La Regresión Logística describe y estima la relación entre una variable binaria dependiente y las variables independientes. La función utilizada en el núcleo del método, es una función Sigmoide. Esta función es una curva en forma de S que puede tomar cualquier número de valor real y asignar a un valor entre 0 y 1.



Si la curva va a infinito positivo la predicción se convertirá en 1, y si la curva pasa el infinito negativo, la predicción se convertirá en 0. Si la salida de la función Sigmoide es mayor que 0.5, podemos clasificar el resultado como 1 o SI, y si es menor que 0.5 podemos clasificarlo como 0 o NO.

Scikit-Learn. Regresión Logística

- Pero veamos este algoritmo de manera matemática. La ecuación de Regresión Lineal es está:

  $y = a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n + b$

- Por su parte, la ecuación de la función Sigmoide es la siguiente:

  $p = 1 / (1 + e^{-y})$

- Entonces si aplicamos la función Sigmoide en la Regresión Lineal nos quedaría algo como esto:

$$p = \frac{1}{1 + e^{-(a_1 x_1 + a_2 x_2 + a_n x_n + b)}}$$

Esta forma funcional es comúnmente identificada como un "perceptrón" de una capa simple o red neuronal artificial de una sola capa.

Scikit-Learn. Regresión Logística

Consideraciones y características de la Regresión Logística.

- La regresión logística requiere tamaños de muestra bastante grandes.
- La razón por la cual la regresión logística es ampliamente utilizada, a pesar de los algoritmos avanzados como redes neuronales, es porque es muy eficiente y no requiere demasiados recursos computacionales.
- Eliminar ruido. La regresión logística no asume ningún error en la variable de salida (y). Considera la eliminación de los valores atípicos y posiblemente las instancias más clasificadas de sus datos de entrenamiento.
- Distribución gaussiana. La regresión logística es un algoritmo lineal, con una transformación no lineal en la salida. Las transformaciones de datos de sus variables de entrada que exponen mejor esta relación lineal pueden dar como resultado un modelo más preciso.
- Eliminar entradas correlacionadas. Al igual que la regresión lineal, el modelo puede sobreajustarse si tiene múltiples entradas altamente correlacionadas. Consideremos calcular las correlaciones por pares entre todas las entradas y eliminar las entradas altamente correlacionadas.
- No se puede converger. Es posible que el proceso de estimación de probabilidad esperado aprenda que los coeficientes no convergen, esto puede suceder si hay muchas entradas altamente correlacionadas en sus datos o si los datos son muy escasos.

Scikit-Learn. Regresión Logística
Ejemplo Práctico. *"Predicción del cáncer de mama"*

Nuestro objetivo es desarrollar un modelo para predecir si un tumor es maligno o benigno, de acuerdo a las características del tumor.

Es aconsejable seguir una rutina de pasos, que va a ser muy parecida cada vez que procesemos un conjunto de datos

1º Importar las librerías necesarias.

La librería Scikit-Learn importaremos el módulo **datasets ,** que nos dará acceso a los conjuntos de datos, para trabajar con ellos.

```
# Importación de las librerías necesarias

from sklearn import datasets
```

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

2º Cargar los datos.

Seguidamente importamos un dataset también muy conocido, sobre el cáncer de mama en Wisconsin. Lo tendremos disponible en numerosas fuentes de datos, pero desde Scilit-Learn es mucho más cómodo de importar y usar.

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

```
#Importamos los datos de la librería breast_cancer
dataset = datasets.load_breast_cancer()
print(boston)
print()
```

Utilizamos la función
.load_breast_cancer( )
Y comprobamos lo que nos sale por pantalla.

Hay mucha información de golpe !!

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

3º Comprobar la información que contiene el conjunto de datos

Entramos en un proceso de compresión de los datos, en el cual analizaremos la información con la que vamos a trabajar. Para ello ejecutaremos el método **.keys()** de nuestro objeto, el conjunto de datos **dataset**

```
#comprobar la información que contiene el dataset
print('Información en el dataset:')
print(dataset.keys())
print()
```

```
Información en el dataset:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

Como podemos observar, nos devuelve la siguiente información: los datos **"data"**, seguido de **"target"** que es la columna con las etiquetas o respuestas, posteriormente **"feature_names"** que son los nombres de cada una de las columnas de datos y **DESCR** que es la descripción completa del dataset.

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

4º Aprovechamos para echar un vistazo a la descripción del dataset:

.DESCR es una cadena de texto que podemos imprimir para analizar su contenido

```
#Verificar las características del dataset
print('Características del dataset:')
print(dataset.DESCR)
```

```
Características del dataset:
.. _breast_cancer_dataset:
Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------
**Data Set Characteristics:**
    :Number of Instances: 569
    :Number of Attributes: 30 numeric, predictive attributes and the class
.....
```

El dataset cuenta con 569 datos y 30 atributos. Algunos de los atributos o variables independientes son: el radio, la textura, el perímetro y el área. También tenemos la descripción del significado de cada columnas o atributo. Un dato importante es, que nos avisa de que no existe ningún valor perdido, en consecuencia, nuestros datos están completos y no necesitaremos realizar mucho preprocesamiento de los mismos.

Finalmente nos indica que entre los datos hay 212 tumores malignos mientras que 357 son benignos. Esto indica que los datos se encuentran balanceados por lo que no nos tenemos que preocupar por trabajar con un dataset desbalanceado

Scikit-Learn. Regresión Logística
Ejemplo Práctico. *"Predicción del cáncer de mama"*

5º Verificar la cantidad de datos del dataset

.data es un **numpy.ndarray** que contiene el grueso de los datos.

Así que podemos usar lo que ya sabemos de numpy y utilizar **.shape** que nos devuelve la tupla: *(filas, columnas)*

```
#Verifico la cantidad de datos que hay en los dataset
print('Cantidad de datos:')
print(dataset.data.shape)
print()
```

```
Cantidad de datos:
(569, 30)
```

Es una información redundante a la que nos a dado la descripción anterior pero no siempre ,el conjunto de datos, tiene una descripción que nos dé esa información, por eso, es una buena práctica acostumbrase a una rutina de comprobaciones sistemáticas, cada vez que nos enfrentamos al conjunto de datos..

Scikit-Learn. Regresión Logística
Ejemplo Práctico. *"Predicción del cáncer de mama"*

6º Verificar también las columnas o atributos

.feature_names es un **numpy.ndarray** que contiene los nombres de las columnas.

Procedemos a mostrar con print el contenido del array

```
#Verificar también las columnas o atributos
print('Nombres de los atributos:')
print(dataset.feature_names)
```

```
Nombres de los atributos:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'……
```

De nuevo, se trata de una información que ya nos había dado la descripción del dataset, pero no siempre el conjunto de datos nos da esa información, por lo que seremos sistemáticos, con las comprobaciones.

Vamos a utilizar todas las variables que se encuentran dentro de "data"

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

7º Preparar los datos para REGRESIÓN LOGÍSTICA

Ahora vamos a proceder a definir las variables de "X" y "y" que vamos emplear en nuestro modelo..

```
#Seleccionamos todas las columnas
X = dataset.data
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
.......
```

Seleccionamos nuestra variable objetivo o dependiente, que será el precio de la vivienda

```
y = dataset.target
```

Scikit-Learn. Regresión Logística

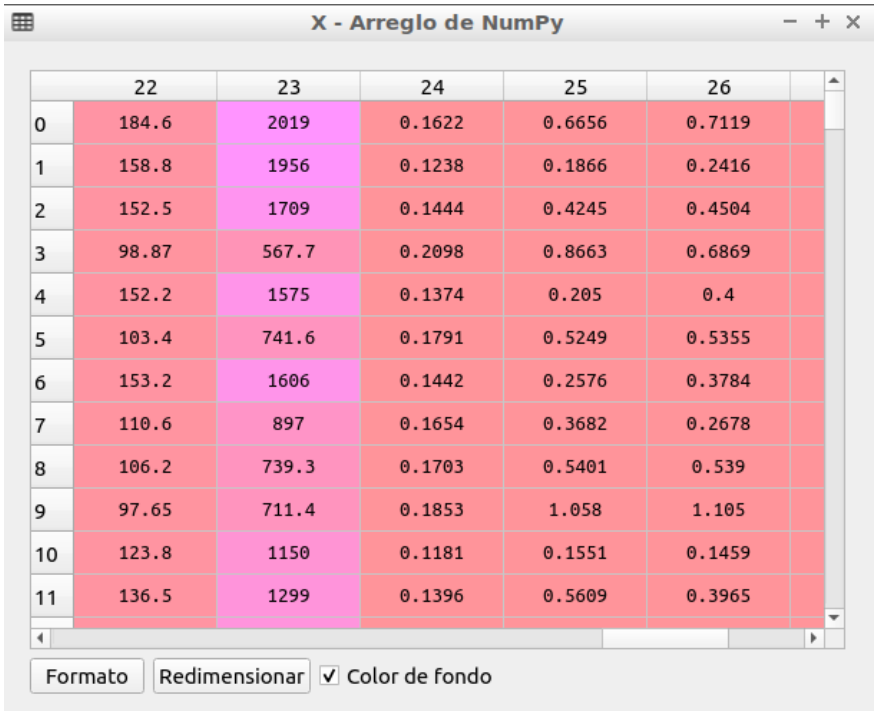
Ejemplo Práctico. *"Predicción del cáncer de mama"*

8º Echaremos un vistazo a los datos. Veamos tanto y como X

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1
```

"y" cuenta con una sola columna con ceros y unos, los unos indican que el tumor es maligno mientras que los ceros corresponden al tumor benigno.

"X" cuenta con 30 columnas distintas en magnitudes, unidades y rango por lo que debemos preprocesar los datos para normalizarlos.



	22	23	24	25	26
0	184.6	2019	0.1622	0.6656	0.7119
1	158.8	1956	0.1238	0.1866	0.2416
2	152.5	1709	0.1444	0.4245	0.4504
3	98.87	567.7	0.2098	0.8663	0.6869
4	152.2	1575	0.1374	0.205	0.4
5	103.4	741.6	0.1791	0.5249	0.5355
6	153.2	1606	0.1442	0.2576	0.3784
7	110.6	897	0.1654	0.3682	0.2678
8	106.2	739.3	0.1703	0.5401	0.539
9	97.65	711.4	0.1853	1.058	1.105
10	123.8	1150	0.1181	0.1551	0.1459
11	136.5	1299	0.1396	0.5609	0.3965

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

9º Preparamos los datos para la Regresión Logística.

Antes de nada, debemos separar los datos en entrenamiento y prueba por lo que importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn**.

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
from sklearn.model_selection import train_test_split
#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.2 )
```

Para normalizar nuestros datos, importamos la clase **StandardScaler** del modulo **preprocessing** de sklearn y crearemos nuestro objeto escalar.
El método **.fit_transform** y **.transform** realizan automáticamente la normalización

```
#Se escalan todos los datos usando la clas StandardScaler
from sklearn.preprocessing import StandardScaler
escalar= StandardScaler()
X_train = escalar.fit_transform(X_train)
X_test = escalar.transform(X_test)
```

$$x' = \frac{x - \mu}{\sigma}$$

x'= valor normalizado
x = valor a normalizar
μ = media
σ = desviación típica

fit_transform debe ejecutarse antes para que calcule las medias y desviaciones típicas antes de hacer transform

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

10º Uso del algoritmo de Regresión logística

Importaremos la clase **LogisticRegression** del módulo linear_model de sklearn.

```
#Defino el algoritmo a utilizar
from sklearn.linear_model import LogisticRegression

algoritmo = LogisticRegression()
```

Una vez que tenemos nuestro modelo para regresión logística instanciado, pasamos a entrenarlo como hemos hecho en otras ocasiones, usando .fit(X,y).

```
#Entrenamiento del modelo
algoritmo.fit(X_train, y_train)
```

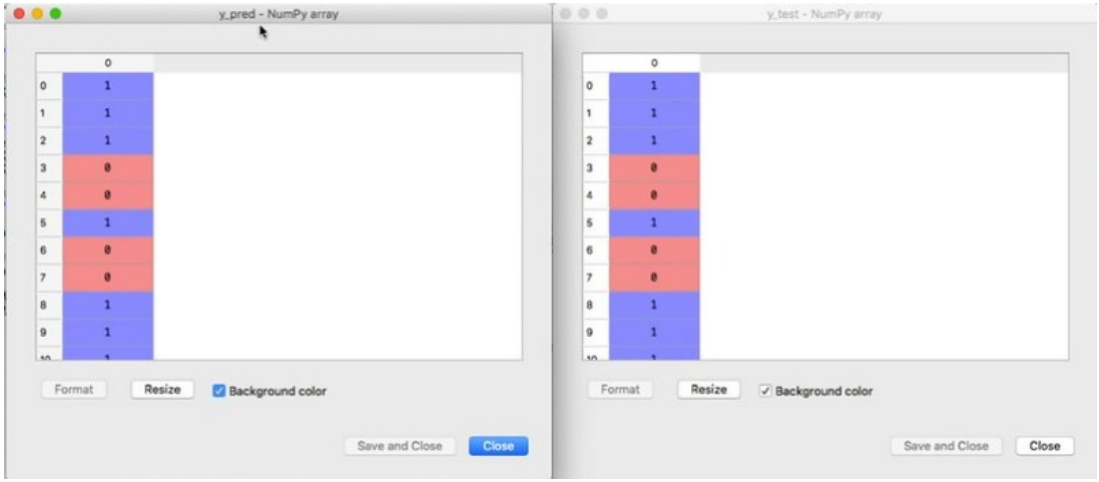
Y finalmente realizamos una predicción, utilizando la instrucción predict y los datos de prueba.

```
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
```

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Vamos a hacer una comparación rápida los valores obtenidos en nuestra predicción con los valores reales, para ver que tal es el comportamiento de nuestro modelo. Para ello usaremos el Explorador de variables de Spyder.



A la izquierda están los datos reales, que separamos en **y_test**. A la derecha están los datos obtenidos de la predicción **Y_pred**. Si observamos con detenimiento podemos ver que nuestro modelo realizo un excelente trabajo ya que a simple vista podemos observar que los datos predichos son exactamente igual a los datos reales.

Scikit-Learn. Regresión Logística

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Pero veamos si esto es cierto, para ello calculamos las métricas respectivas para verificar el rendimiento del modelo. Para esto debemos primero obtener la matriz de confusión utilizando el módulo metrics de scikit learn y la función confusion-matrix.

Y lo utilizamos con los datos reales, del conjunto de prueba y los que hemos predicho con anterioridad.

```
#Verifico la matriz de Confusión
from sklearn.metrics import confusion_matrix

matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz)
```

```
Matriz de Confusión:
[[44  3]
 [ 1 66]]
```

La Matriz de Confusión es una de las métricas más intuitivas y sencillas que se utiliza para encontrar la precisión y exactitud del modelo. Se utiliza para el problema de clasificación donde la salida puede ser de dos o más tipos de clases.

	Predice 0	**Predice 1**
Real 0	Verdadero Negativo	Falso Positivo
Real 1	Falso Negativo	Verdadero Positivo

Scikit-Learn

Contenidos :

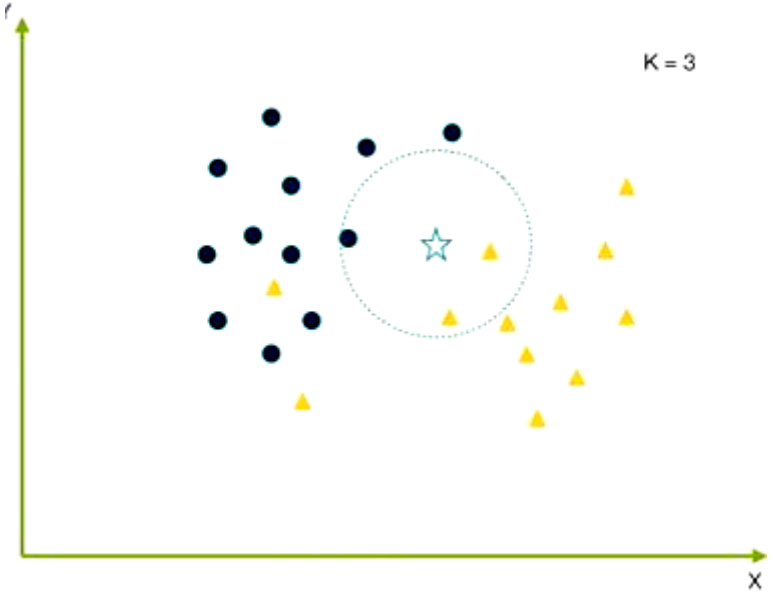
- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

Scikit-Learn. K vecinos más cercanos

- K vecinos más cercanos KNN (K-Nearest Neighbors) es uno de los algoritmos de clasificación más básicos y esenciales en Machine Learning. Pertenece al dominio del aprendizaje supervisado y a pesar de su simplicidad, KNN puede superar a los clasificadores más potentes.
- Se usa en una variedad de aplicaciones tales como pronósticos económicos, compresión de datos, genética, encontrando una aplicación intensa en el reconocimiento de patrones y la minería de datos.

Funcionamiento:

Si Z es el punto a predecir, primero, se encuentran los K puntos más cercanos a Z y luego se clasifican los puntos para el voto mayoritario de sus vecinos K. Cada objeto vota por su clase y la clase con más votos se toma como la predicción. Para encontrar los puntos similares más cercanos, se calcula la distancia entre puntos utilizando por ejemplo la distancia euclidia.



Distancia Euclidiana

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Distancia Manhattan

$$\sum_{i=1}^{k}|x_i - y_i|$$

Distancia Minkowski

$$[\sum_{i=1}^{k}(|x_i - y_i|)^4]^{\frac{1}{4}}$$

Scikit-Learn. K vecinos más cercanos

¿Cómo elegir el número de vecinos K ?

- Debe elegirse en el momento de la construcción del modelo. Puedes pensar en K como una variable de control para el modelo de predicción.
- La investigación ha demostrado que no existe un número óptimo de vecinos que se adapte a todo tipo de conjuntos de datos. Cada conjunto de datos tiene sus propios requisitos.
- Se ha demostrado que una pequeña cantidad de vecinos son los más flexibles, que tendrán un bajo sesgo, pero una alta varianza, y un gran número de vecinos tendrán un límite de decisión más suave, lo que significa una varianza más baja pero un sesgo más alto.
- Generalmente, se recomienda elegir un número impar si el número de clases es par. También puede comprobar generando el modelo en diferentes valores de K y comprobar su rendimiento.

Scikit-Learn. K vecinos más cercanos

CARACTERÍSTICAS de K vecinos más cercanos:

- Es algoritmo de aprendizaje no paramétrico: significa que no requieren que el modelo haga suposiciones sobre la distribución de la población.
- Es un algoritmo de aprendizaje **basado en la instancia**: significa que nuestro algoritmo no aprende explícitamente un modelo. En lugar de ello, opta por memorizar las instancias de formación que posteriormente se utilizan como “conocimiento” para la fase de predicción. Concretamente, esto significa que solo cuando se realiza una consulta a nuestra base de datos, es decir, cuando le pedimos que pronostique una etiqueta con una entrada, el algoritmo utilizará las instancias de formación para dar una respuesta.
- Computacionalmente costoso. Cada vez que se realiza la consulta.
- Requisito de memoria alta. Almacena todos (o casi todos) los datos de entrenamiento.
- Alta precisión (relativa). Es bastante alta pero no competitiva en comparación con modelos de aprendizaje mejor supervisados.
- Insensible a los valores atípicos. La precisión no suele verse afectada por el ruido o las características irrelevantes.

Scikit-Learn. K vecinos más cercanos

- La librería Scikit Learn cuenta con un modelo que se puede utilizar para implementar el algoritmo de K vecinos más cercanos clasificación. Para ello debemos importar el módulo skelearn.neigbors. Realizado esto se procede a importar la classe NeighborsClassifier.

  *class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)*

  La librería te ofrece varios parámetros que puedes utilizar para configurar el algoritmo. Veamos algunos importantes:

- "n_neighbors". Número de vecinos o K. Por defecto es 5, pero es recomendable que en cada proyecto sea el desarrollador quien defina este valor para tener más control sobre el modelo.
- Distancia que se utilizará para verificar los vecinos del dato a predecir. Para configurar esto en el algoritmo se deben definir dos parámetros:
  - "p". p=1 utilizar la distancia manhattan, p=2 distancia euclidea. Si p= valor arbitrario, utilizar la distancia minkowski
  - "metric". por defecto es 'minkowski' que unida a p=2 define una distancia euclidia

Puedes ver los otros parámetros en la página oficial de la librería:

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Scikit-Learn. K vecinos más cercanos

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Vamos a volver a usar el conjunto de datos "breast cancer"**.** Con la intención de predecir si un tumor es maligno o benigno, de acuerdo a las características del tumor.

Como ya hemos realizado los pasos previos, en el algoritmo anterior, (cargar el dataset y analizarlo) pasaremos a usar el algoritmo KNN

Utilizaremos todas las variables que se encuentran dentro de "data"

Ahora vamos a proceder a definir las variables de "X" y "y" que vamos emplear en nuestro modelo.

```
#Seleccionamos todas las columnas
X = dataset.data
```

Seleccionamos nuestra variable objetivo o dependiente, que será el precio de la vivienda

```
y = dataset.target
```

Scikit-Learn. K vecinos más cercanos

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Preparamos los datos para el entrenamiento:

Antes de nada, debemos separar los datos en entrenamiento y prueba por lo que importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn.**

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
from sklearn.model_selection import train_test_split

#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.2 )
```

Scikit-Learn. K vecinos más cercanos

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Uso del algoritmo de **K vecinos más cercanos**

Importaremos la clase **KNeighborsClassifier** del módulo neighbors de sklearn.

Le indicamos que "k" o el número de vecinos será igual a 5, la métrica será "minkowski" y "p" será igual a 2. Esta combinación de parámetros le indica al algoritmo que utilizará la distancia Euclidea entre los vecinos

```
#Defino el algoritmo a utilizar
from sklearn.neighbors import KNeighborsClassifier

algoritmo = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
```

Una vez que tenemos nuestro modelo para regresión logística instanciado, pasamos a entrenarlo como hemos hecho en otras ocasiones, usando .fit(X,y).

```
#Entrenamiento del modelo
algoritmo.fit(X_train, y_train)
```

Y finalmente realizamos una predicción, utilizando la instrucción predict y los datos de prueba.

```
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
```

Scikit-Learn. K vecinos más cercanos

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Calculamos las métricas respectivas para verificar el rendimiento del modelo. Para esto debemos primero obtener la matriz de confusión utilizando el módulo metrics de scikit learn y la función confusion-matrix.

Y lo utilizamos con los datos reales, del conjunto de prueba y los que hemos predicho con anterioridad.

```
#Verifico la matriz de Confusión
from sklearn.metrics import confusion_matrix

matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz)
```

```
Matriz de Confusión:
[[33  5]
 [ 2 74]]
```

	Predice 0	Predice 1
Real 0	Verdadero Negativo	Falso Positivo
Real 1	Falso Negativo	Verdadero Positivo

Veamos ahora la precisión del modelo, para ello importamos la función "precision_score" del módulo metrics y la implementamos junto con los datos predichos y los reales.

```
#Calculo la precisión del modelo
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print('Precisión del modelo:')
print(precision)
```

```
Precisión del modelo:
0.9367088607594937
```

Scikit-Learn

Contenidos :

- Scikit Learn ¿Qué es?
- Instalación
- Componentes de Scikit-Learn
- Aprendizaje Supervisado. Regresión
  - Regresión lineal Simple
    - Ejemplo práctico
  - Regresión lineal Múltiple
    - Ejemplo práctico
  - Regresión Polinómica
    - Ejemplo práctico
  - Regresión con Vectores Soporte (SVR)
    - Ejemplo práctico
  - Regresión con Árboles de decisión
    - Ejemplo práctico
- Ventajas y desventajas de los algoritmos de regresión
  - Regresión Lineal
  - Vectores de Soporte Regresión
  - Árboles de Decisión Regresión
  - Bosques Aleatorios Regresión
- Aprendizaje Supervisado. Clasificación
  - Regresión Logística
    - Ejemplo práctico
  - K vecinos más cercanos
    - Ejemplo práctico
  - Máquinas de Vectores Soporte Clasificación (SVC)
    - Ejemplo práctico
- Ejercicio. Predecir la supervivencia a bordo del Titanic

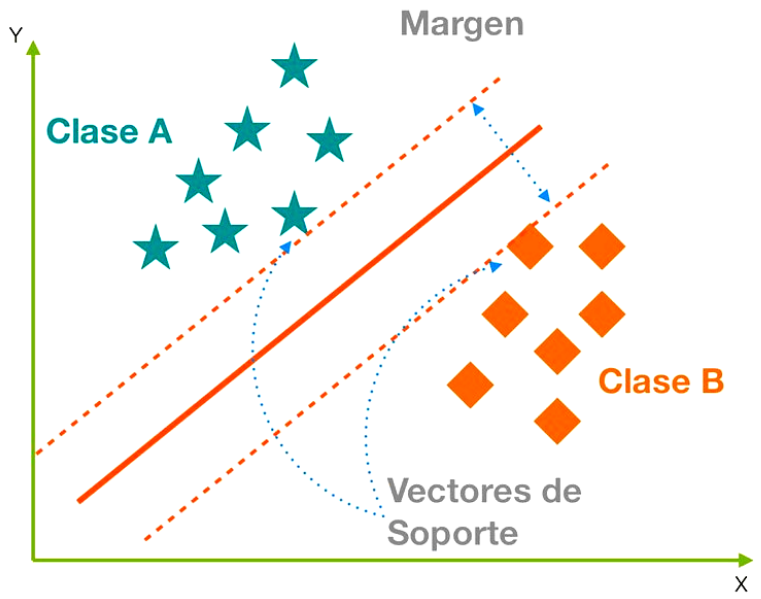
Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

- Las Máquinas Vectores de Soporte clasificación ofrece una precisión muy alta en comparación con otros clasificadores como la Regresión Logística y los Árboles de Decisión.
- Se utiliza una variedad de aplicaciones tales como detección de rostros, detección de intrusos, clasificación de correos electrónicos, artículos de noticias y páginas web, entre otros.
- La idea central de SVM es encontrar un hiperplano marginal máximo que mejor divida el conjunto de datos en clases.

Hiperplano: es un plano de decisión que separa entre conjuntos de objetos que pertenecen a clases diferentes

Vectores de Soporte: son los puntos de datos más cercanos al hiperplano. Estos puntos definirán mejor la línea de separación calculando los márgenes. Estos puntos son más relevantes para la construcción del clasificador.

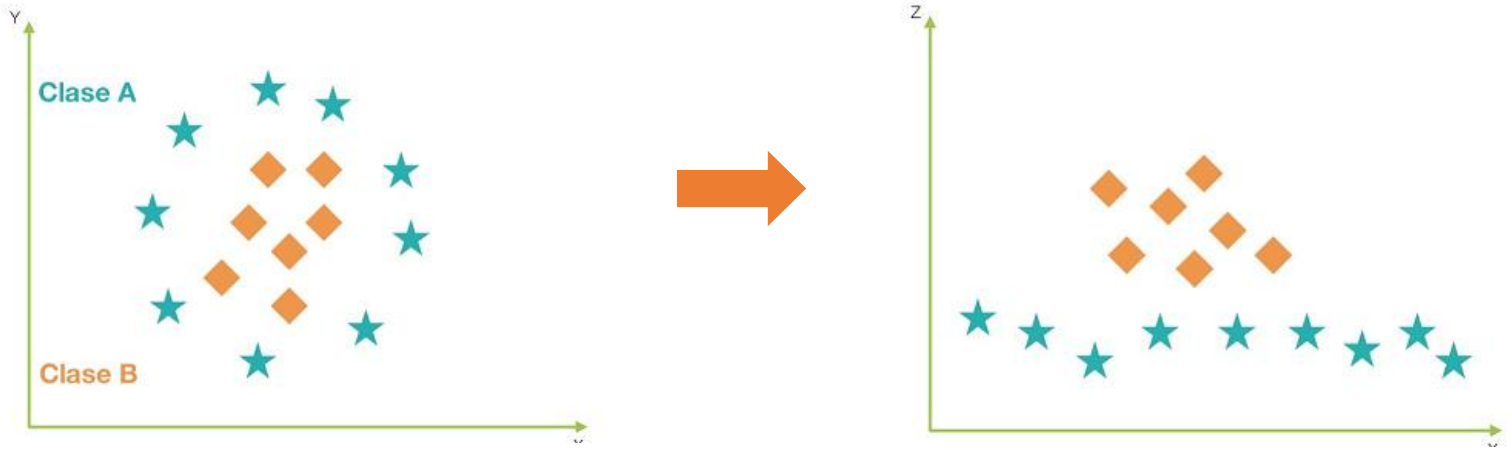
Margen: Se calcula como la distancia perpendicular desde el hiperplano hasta los vectores de soporte o puntos más cercanos. Si el margen es mayor entre las clases, entonces se considera un buen margen.



Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

- No siempre se puede usar un hiperplano lineal. En tales situaciones, el algoritmo utiliza un truco del núcleo para transformar el espacio de entrada en un espacio dimensional superior, como se muestra en la figura
- EL truco del kernel nos permite operar en el espacio de la característica original, sin calcular las coordenadas de los datos, en un espacio dimensional superior.

En el apartado sobre Vectores Soporte para Regresión, vimos algunos tipos de Kernel.
En el ejemplo que tenemos en la figura, los puntos de datos se representan en los ejes “x” y “z”, en donde “z” es la suma cuadrada de “x” e “y” (z = x^2 + y^2). Ahora se puede separar fácilmente estos puntos utilizando la separación lineal.



Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

Ventajas

- Los clasificadores de Máquinas de Vectores de Soporte ofrecen una buena precisión y realizan predicciones más rápidas que otros algoritmos
- También utilizan menos memoria porque utilizan un subconjunto de puntos de entrenamiento en la fase de decisión. Este algoritmo funciona bien con un claro margen de separación y con un espacio dimensional elevado.

Desventajas

- Las Máquinas de Vectores de Soporte no son adecuadas para grandes conjuntos de datos debido a su alto tiempo de formación. Funciona mal con clases superpuestas y también es sensible al tipo de núcleo utilizado.

Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

- Scikit Learn cuenta con un modulo llamado **sklearn.svm** donde se incluye todo lo referente al algoritmo de Vectores de Soporte tanto clasificación como regresión. Para esta entrada nos enfocaremos solamente en clasificación, concretamente en la clase SVC.

  class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

  SVC se puede configurar con ciertos parámetros (ver documentación para más detalles).

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Los más importantes que se deben considerar al momento de configurar el modelo son:

- C. Es el parámetro de penalización, del término de error. Indica a la optimización del algoritmo cuánto error es soportable. Con él se puede controlar el compromiso entre el límite de decisión y el concepto de clasificación errónea. Un valor menor de C crea un hiperplano de pequeño margen y un valor de mayor de C crea un hiperplano de mayor.
- "Kernel" Puede ser 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' . Las funciones polinomial y RBF o radial, son útiles para un hiperplano no lineal. Los núcleos polinomiales y RBF calculan la separación de clases en la dimensión superior. En algunas de las aplicaciones, se sugiere utilizar un núcleo más complejo para separar las clases que son curvas o no lineales. Esta transformación puede conducir a clasificadores más precisos.

Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Vamos a volver a usar el conjunto de datos "breast cancer". Con la intención de predecir si un tumor es maligno o benigno, de acuerdo a las características del tumor.

Como ya hemos realizado los pasos previos, en el algoritmo anterior, (cargar el dataset y analizarlo) pasaremos a usar el algoritmo SVC

Utilizaremos todas las variables que se encuentran dentro de "data"

Ahora vamos a proceder a definir las variables de "X" y "y" que vamos emplear en nuestro modelo.

```
#Seleccionamos todas las columnas
X = dataset.data
```

Seleccionamos nuestra variable objetivo o dependiente, que será el precio de la vivienda

```
y = dataset.target
```

Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Preparamos los datos para el entrenamiento:

Antes de nada, debemos separar los datos en entrenamiento y prueba por lo que importaremos la función **train_test_split**, del módulo **model_selection** de **sklearn.**

Para la separación de los datos, vamos a tomar un **20%** de los mismos para utilizarlos como prueba una vez que hayamos obtenido el modelo, por lo que el parámetro **test_size** será **0.2**

```
from sklearn.model_selection import train_test_split

#Separo los datos de "train" en entrenamiento y prueba para probar los algoritmos

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.2 )
```

Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Uso del algoritmo de **SVC**

Importaremos la clase SVC del módulo svm de sklearn.

Le indicamos que "kernel" será igual a 'linear'. El resto de parámetros los dejamos por defecto.

```
#Defino el algoritmo a utilizar
from sklearn.svm import SVC

algoritmo = SVC(kernel='linear')
```

Una vez que tenemos nuestro modelo para regresión logística instanciado, pasamos a entrenarlo como hemos hecho en otras ocasiones, usando .fit(X,y).

```
#Entrenamiento del modelo
algoritmo.fit(X_train, y_train)
```

Y finalmente realizamos una predicción, utilizando la instrucción predict y los datos de prueba.

```
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
```

Scikit-Learn. Máquinas de Vectores Soporte Clasificación (SVC)

Ejemplo Práctico. *"Predicción del cáncer de mama"*

Calculamos las métricas respectivas para verificar el rendimiento del modelo. Para esto debemos primero obtener la matriz de confusión utilizando el módulo metrics de scikit learn y la función confusion-matrix.

Y lo utilizamos con los datos reales, del conjunto de prueba y los que hemos predicho con anterioridad.

```
#Verifico la matriz de Confusión
from sklearn.metrics import confusion_matrix

matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz)
```

```
Matriz de Confusión:
[[42  2]
 [ 1 69]]
```

	Predice 0	Predice 1
Real 0	Verdadero Negativo	Falso Positivo
Real 1	Falso Negativo	Verdadero Positivo

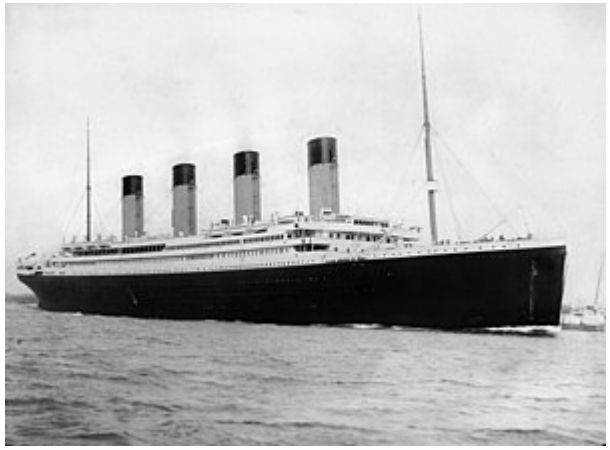
Veamos ahora la precisión del modelo, para ello importamos la función "precision_score" del módulo metrics y la implementamos junto con los datos predichos y los reales.

```
#Calculo la precisión del modelo
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print('Precisión del modelo:')
print(precision)
```

```
Precisión del modelo:
0.971830985915493
```

Scikit-Learn. Ejercicio

Predecir la supervivencia a bordo del Titanic

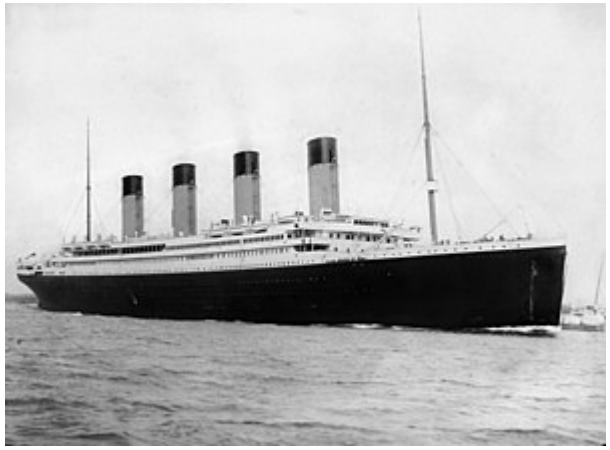


Unos de los primeros y más famosos ejercicios, cuando se está comenzando en Machine Learning, es el de predecir la supervivencia en la tragedia del Titanic.
Es decir, usando un conjunto de datos, sobre el pasaje que iba en el barco, con características como la edad, el sexo, o la clase, entrenamos a nuestro algoritmo de aprendizaje automático para ver si es capaz de predecir que pasajeros se salvaron.

En esta ocasión, el dataset no lo vamos a coger de Scikit-Learn sino de una fuente externa.
Podéis ir, en caso de estar registrados a https://www.kaggle.com/c/titanic
En este curso ya los hemos descargado por vosotros y los podéis bajar de:
http://python.iaa.es/

Scikit-Learn. Ejercicio

Predecir la supervivencia a bordo del Titanic



El dataset consiste en dos archivos CSV:

- train.csv Conjunto de datos de entrenamiento
- test.csv Conjunto de datos de prueba.

Los atributos de cada columna son (salvo el conjunto de test.csv, que no tiene la columna survival):

PassengerId: Identificador del pasajero
survival: {0, 1}
pclass: {1st, 2nd, 3rd}
name: nombre
sex: sexo
age: edad
sibsp: número de parientes (hermano/a, hermanastro/a) / cónyuge (esposo/a) a bordo
parch: número de padres (madre/padre) / hijos a bordo (hijo/a, hijastro/a)
ticket: número de ticket
fare: precio del ticket
cabin: número del camarote
embarked: puerto de embarque

Scikit-Learn. Ejercicio

Predecir la supervivencia a bordo del Titanic

Realiza el siguiente programa en el que deberás seguir los siguientes pasos para preprocesar y tratar los datos:

- Importar las librerías necesarias.
- Importar el conjunto de datos de entrenamiento y prueba y guardar dichos datos, en variables de Pandas para poderlos manipular a lo largo del programa.
- Explorar el conjunto de datos de entrenamiento y prueba para determinar información concerniente al tipo, datos faltantes, etc…
- Sustituir datos que no son numéricos como el sexo: female, male, por 0 y 1 o el puerto de embarque embarked: 'Q', 'S', 'C' por 0, 1, 2
- Reemplazar los datos que faltan en la edad (NaN), por la media de esta columna
- Crear varios grupos de edades en intervalos : 0-8, 9-15, 16-18, 19-25, 26-40, 41-60, 61-100 y nombra los grupos del 1 al 7 ['1', '2', '3', '4', '5', '6', '7']

Pistas: Para sustituir datos con Pandas usa ***.replace*** y para crear bandas usa ***.cut***

Scikit-Learn. Ejercicio

Predecir la supervivencia a bordo del Titanic

• Contabilizar los datos nulos en ambos conjuntos. Eliminar completamente las filas con datos perdidos, cuando falten pocos datos y no suponga mucha pérdida.
Usar ***.dropna( )*** de Pandas

• Hay una serie de datos que no serán necesarios (utiliza **.drop( )** de Pandas):
  • Allí donde hay muchos datos perdidos. Ej: eliminar la columna cabin del conjunto de entrenamiento y prueba
  • Las columnas 'PassengerId', 'name', 'ticket' del conjunto de entrenamiento y prueba.
  Nota: antes guardar los datos de PassengerId y name del conjunto de prueba para identificar los viajeros al final

• Separar la columna con la información de los sobrevivientes (nuestro target **"y"**) del resto de datos **"X"**, sólo para el conjunto de entrenamiento.

• Dividir el dataset de entrenamiento, tal y como hemos hecho hasta ahora en X_train, X_test, y_train, y_test 20% de datos para test.

Scikit-Learn. Ejercicio

Predecir la supervivencia a bordo del Titanic

- Implementar los algoritmos de Machine Learning, vistos en los métodos de clasificación de los ejemplo anteriores: regresión logística, vectores de soporte y vecinos más cercanos

- Calcular la matriz de confusión

- Realizar predicciones, si se desea, sobre el conjunto de datos de prueba original, del que se desconocía si sobrevivían o no, publicando los nombres de los pasajeros y si sobrevive o no.

- ¿Cuáles son los nombres de los pasajeros supervivientes?

Gracias por su atención