Aprendizaje Supervisado

Abraham Zamudio

VI Programa de Especialización en Machine Learning con Python

2020

CTIC-UNI
BUSINESS SCHOOL

Algunos topicos de machine learning

El Machine Learning es el diseño y estudio de las herramientas informáticas que utilizan la experiencia pasada para tomar decisiones futuras; es el estudio de programas que pueden aprenden de los datos. El objetivo fundamental del Machine Learning es generalizar, o inducir una regla desconocida a partir de ejemplos donde esa regla es aplicada. El ejemplo más típico donde podemos ver el uso del Machine Learning es en el filtrado de los correo basura o spam. Mediante la observación de miles de correos electrónicos que han sido marcados previamente como basura, los filtros de spam aprenden a clasificar los mensajes nuevos. El Machine Learning combina conceptos y técnicas de diferentes áreas del conocimiento, como las matemáticas, estadísticas y las ciencias de la computación; por tal motivo, hay muchas maneras de aprender la disciplina.

Algunos topicos de machine learning

El Machine Learning tiene una amplia gama de aplicaciones, incluyendo motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, juegos y robótica. Pero para poder abordar cada uno de estos temas es crucial en primer lugar distingir los distintos tipos de problemas de Machine Learning con los que nos podemos encontrar.

Aprendizaje supervisado

En los problemas de aprendizaje supervisado se enseña o entrena al algoritmo a partir de datos que ya vienen etiquetados con la respuesta correcta. Cuanto mayor es el conjunto de datos más el algoritmo puede aprender sobre el tema. Una vez concluído el entrenamiento, se le brindan nuevos datos, ya sin las etiquetas de las respuestas correctas, y el algoritmo de aprendizaje utiliza la experiencia pasada que adquirió durante la etapa de entrenamiento para predecir un resultado. Esto es similar al método de aprendizaje que se utiliza en las escuelas, donde se nos enseñan problemas y las formas de resolverlos, para que luego podamos aplicar los mismos métodos en situaciones similares.

Aprendizaje no supervisado

En los problemas de aprendizaje no supervisado el algoritmo es entrenado usando un conjunto de datos que no tiene ninguna etiqueta; en este caso, nunca se le dice al algoritmo lo que representan los datos. La idea es que el algoritmo pueda encontrar por si solo patrones que ayuden a entender el conjunto de datos. El aprendizaje no supervisado es similar al método que utilizamos para aprender a hablar cuando somos bebes, en un principio escuchamos hablar a nuestros padres y no entendemos nada; pero a medida que vamos escuchando miles de conversaciones, nuestro cerebro comenzará a formar un modelo sobre cómo funciona el lenguaje y comenzaremos a reconocer patrones y a esperar ciertos sonidos.

Aprendizaje por refuerzo

En los problemas de aprendizaje por refuerzo, el algoritmo aprende observando el mundo que le rodea. Su información de entrada es el feedback o retroalimentación que obtiene del mundo exterior como respuesta a sus acciones. Por lo tanto, el sistema aprende a base de ensayo-error. Un buen ejemplo de este tipo de aprendizaje lo podemos encontrar en los juegos, donde vamos probando nuevas estrategias y vamos seleccionando y perfeccionando aquellas que nos ayudan a ganar el juego. A medida que vamos adquiriendo más practica, el efecto acumulativo del refuerzo a nuestras acciones victoriosas terminará creando una estrategia ganadora.

La idea fundamental de Machine Learning es encontrar patrones que podamos generalizar para poder aplicar esta generalización sobre los casos aún no observados y realizar predicciones. Pero también ocurre que durante el entrenamiento, el sistema solo descubra casualidades y esto se le conoce como sobreentrenamiento.

Todos los modelos de Machine Learning tienen tendencia al sobreentrenamiento; es por esto que debemos aprender a convivir con el mismo y tratar de tomar medidas preventivas para reducirlo lo más posible. Las dos principales estrategias para evitar el sobreentrenamiento son:

- Retención de datos
- Validación cruzada.

Para la retención de datos, el objetivo es dividir nuestro conjunto de datos en uno o varios conjuntos de entrenamiento y otros conjuntos de evaluación. Es decir, que no le vamos a pasar todos nuestros datos al algoritmo durante el entrenamiento, sino que vamos a retener una parte de los datos de entrenamiento para realizar una evaluación de la efectividad del modelo.

Para la validación cruzada se requiere realizar un análisis estadístico para obtener otras medidas del rendimiento estimado, como la media y la varianza, y así poder entender cómo se espera que el rendimiento varíe a través de los distintos conjuntos de datos.

Nota: En la práctica el más común es el uso de la retención de datos.

Recolectar los datos.
Podemos recolectar los datos desde muchas fuentes, podemos por ejemplo extraer los datos de un sitio web o obtener los datos utilizando una API o desde una base de datos. Podemos también utilizar otros dispositivos que recolectan los datos por nosotros; o utilizar datos que son de dominio público. El número de opciones que tenemos para recolectar datos no tiene fin!. Este paso parece obvio, pero es uno de los que más complicaciones trae y más tiempo consume.

Preprocesar los datos

Una vez que tenemos los datos, tenemos que asegurarnos que tiene el formato correcto para nutrir nuestro algoritmo de aprendizaje. Es prácticamente inevitable tener que realizar varias tareas de preprocesamiento antes de poder utilizar los datos. Igualmente este punto suele ser mucho más sencillo que el paso anterior.

Explorar los datos

Una vez que ya tenemos los datos y están con el formato correcto, podemos realizar un pre análisis para corregir los casos de valores faltantes o intentar encontrar a simple vista algún patrón en los mismos que nos facilite la construcción del modelo. En esta etapa suelen ser de mucha utilidad las medidas estadísticas y los gráficos en 2 y 3 dimensiones para tener una idea visual de como se comportan nuestros datos. En este punto podemos detectar valores atípicos que debamos descartar; o encontrar las características que más influencia tienen para realizar una predicción.

Entrenar el algoritmo
Aquí es donde comenzamos a utilizar las técnicas de Machine Learning realmente. En esta etapa nutrimos al o los algoritmos de aprendizaje con los datos que venimos procesando en las etapas anteriores. La idea es que los algoritmos puedan extraer información útil de los datos que le pasamos para luego poder hacer predicciones.

Evaluar el algoritmo

En esta etapa ponemos a prueba la información o conocimiento que el algoritmo obtuvo del entrenamiento del paso anterior. Evaluamos que tan preciso es el algoritmo en sus predicciones y si no estamos muy conforme con su rendimiento, podemos volver a la etapa anterior y continuar entrenando el algoritmo cambiando algunos parámetros hasta lograr un rendimiento aceptable.

Utilizar el modelo

En esta ultima etapa, ya ponemos a nuestro modelo a enfrentarse al problema real. Aquí también podemos medir su rendimiento, lo que tal vez nos obligue a revisar todos los pasos anteriores.

Librerías de Python para machine learning

Una de las grandes ventajas que ofrece Python sobre otros lenguajes de programación; es lo grande y prolifera que es la comunidad de desarrolladores que lo rodean; comunidad que ha contribuido con una gran variedad de librerías de primer nivel que extienden la funcionalidades del lenguaje. Para el caso de Machine Learning, listamos aqui algunas de las mas comunes.

Librerías de Python para machine learning

Scikit-Learn

Scikit-learn es la principal librería que existe para trabajar con Machine Learning, incluye la implementación de un gran número de algoritmos de aprendizaje. La podemos utilizar para clasificaciones, extraccion de características, regresiones, agrupaciones, reducción de dimensiones, selección de modelos, o preprocesamiento. Posee una API que es consistente en todos los modelos y se integra muy bien con el resto de los paquetes científicos que ofrece Python. Esta librería también nos facilita las tareas de evaluación, diagnostico y validaciones cruzadas ya que nos proporciona varios métodos de fábrica para poder realizar estas tareas en forma muy simple.

Statsmodels es otra gran librería que hace foco en modelos estadísticos y se utiliza principalmente para análisis predictivos y exploratorios. Al igual que Scikit-learn, también se integra muy bien con el resto de los paquetes cientificos de Python. Si deseamos ajustar modelos lineales, hacer una análisis estadístico, o tal vez un poco de modelado predictivo, entonces Statsmodels es la librería ideal. Las pruebas estadísticas que ofrece son bastante amplias y abarcan tareas de validación para la mayoría de los casos.

Librerías de Python para machine learning

PyMC

pyMC es un módulo de Python que implementa modelos estadísticos bayesianos, incluyendo la cadena de Markov Monte Carlo(MCMC). pyMC ofrece funcionalidades para hacer el análisis bayesiano lo mas simple posible. Incluye los modelos bayesianos, distribuciones estadísticas y herramientas de diagnostico para la covarianza de los modelos. Si queremos realizar un análisis bayesiano esta es sin duda la librería a utilizar.

Librerías de Python para machine learning

NTLK

NLTK es la librería líder para el procesamiento del lenguaje natural o NLP por sus siglas en inglés. Proporciona interfaces fáciles de usar a más de 50 cuerpos y recursos léxicos, como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para la clasificación, tokenización, el etiquetado, el análisis y el razonamiento semántico.

Los algoritmos que más se suelen utilizar en los problemas de Machine Learning son los siguientes:

- Regresión Lineal
- Regresión Logística
- Arboles de Decision
- Random Forest
- SVM o Máquinas de vectores de soporte.
- KNN o K vecinos más cercanos.
- K-means

Primero recordar que una regresión lineal se puede implementar con scipy,numpy,statsmodel o Scikit-Learn. Por ejemplo haciendo uso solo de numpy obtendríamos el siguiente código :

```
1  from numpy import arange,array,ones,linalg
2  from pylab import plot,show
3
4  xi = arange(0,9)
5  A = array([ xi, ones(9)])
6  # secuencia generada
7  y = [19, 20, 20.5, 21.5, 22, 23, 23, 25.5, 24]
8  w = linalg.lstsq(A.T,y)[0] # obtencion de parametros
9
10 # graficando la linea
11 line = w[0]*xi+w[1] # linea de regresion
12 plot(xi,line,'r-',xi,y,'o')
13 show()
```

Ahora veamos el codigo usando scipy:

```
1  from numpy import arange,array,ones
2  from pylab import plot,show
3  from scipy import stats
4
5  xi = arange(0,9)
6  A = array([ xi, ones(9)])
7  # secuencia generada
8  y = [19, 20, 20.5, 21.5, 22, 23, 23, 25.5, 24]
9
10 slope, intercept, r_value, p_value, std_err = stats.linregress(xi,y)
11
12 print('r value', r_value)
13 print('p_value', p_value)
14 print('standard deviation', std_err)
15
16 line = slope*xi+intercept
17 plot(xi,line,'r-',xi,y,'o')
18 show()
```

Topicos de Machine Learning

Lo basico de una regresion lineal

```
1   import numpy as np
2   import statsmodels.api as sm
3
4   y = [1,2,3,4,3,4,5,4,5,5,4,5,4,5,4,5,6,5,4,5,4,3,4]
5
6   x = [
7   [4,2,3,4,5,4,5,6,7,4,8,9,8,8,6,6,5,5,5,5,5,5,5],
8   [4,1,2,3,4,5,6,7,5,8,7,8,7,8,7,8,7,7,7,7,7,6,5],
9   [4,1,2,5,6,7,8,9,7,8,7,8,7,7,7,7,7,7,6,6,4,4,4]
10  ]
11
12  def reg_m(y, x):
13          ones = np.ones(len(x[0]))
14          X = sm.add_constant(np.column_stack((x[0], ones)))
15          for ele in x[1:]:
16                  X = sm.add_constant(np.column_stack((ele, X)))
17          results = sm.OLS(y, X).fit()
18          return results
19
20
21  print(reg_m(y, x).summary())
```

El Boston Housing Dataset consiste en el precio de las casas en varios lugares en Boston. Junto con el precio, el conjunto de datos también proporciona información como el crimen (CRIM), áreas de negocios no minoristas en la ciudad (INDUS), la edad de las personas que poseen la casa (AGE), y hay muchos otros atributos disponibles.

`https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names`

El dataset en sí está disponible en linea. Sin embargo, como vamos a usar scikit-learn, podemos importarlo directamente desde scikit-learn, ademas usaremos otros modulos de python.

```
1  import numpy as np
2  import pandas as pd
3  import scipy.stats as stats
4  import matplotlib.pyplot as plt
5  import sklearn.cross_validation
6  import statsmodels.api as sm
```

Exploremos un poco el dataset,en primer lugar, al igual que lo que hacemos con cualquier otro conjunto de datos, vamos a importar el dataset Boston Housing y almacenarlo en una variable llamada boston. Para importarlo de scikit-learn tendremos que ejecutar este fragmento de código

```
1 from sklearn.datasets import load_boston
2 boston = load_boston()
```

La variable boston es un diccionario, por ello veamos sus llaves :

```
1 print(boston.keys())
```

Veamos la dimencion de la data en el dataset,asi como otras propiedades :

```
1 print(boston.data.shape)
2 print(boston.feature_names)
3 print(boston.DESCR)
```

Ahora convirtamos la data del dataset en un DataFrame de pandas.

```
1 bos = pd.DataFrame(boston.data)
2 print(bos.head())
```

Ojo con los nombres de cada una de las columnas. Utilicemos el campo `features_names` del dataset `boston` original.

```
1  bos.columns = boston.feature_names
2  print(bos.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

OJO, ¿Alguien se da cuenta de que no hay una columna llamada 'PRECIO' en el dataset?

Sí, es porque la columna `target` está disponible en otro atributo llamado `target`. Así que vamos a verificar el `shape` del boston.target.

```
print(boston.target.shape)
```

Entonces, como resulta que coincide con el número de filas del dataset. Vamos a agregarlo al DataFrame como una nueva columna

```
bos['PRICE'] = boston.target
print(bos.head())
```

Con la data ya preparada (o pre-procesada) podemos empezar con el resumen estadístico

```
1  print(bos.describe())
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	T
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	5
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	4
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	1
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	2
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	3
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	6
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	7

El siguiente paso es partir (split) la data a estudiar (DataFrame) , inicialmente, en dos conjuntos: `Train` y `Test`. A diferencia del dataset `titanic`, en `boston` solo tenemos un solo conjunto de datos. No hay datos de entrenamiento (Train) y Testeo (Test). Así que hay que hacer esta discretización nosotros mismos.
Básicamente antes de dividir los datos en los conjuntos `train-test`, tendríamos que dividir el conjunto de datos en valores `target` y valores `predictor` .

Tomemos

Y = Precios de las casas en Boston.
X = Las otras características.

```
1  X = bos.drop('PRICE', axis = 1)
2  Y = bos['PRICE']
```

Ahora, finalmente podemos dividir el conjunto de datos en entrenamiento (`Train`) y pruebas (`test`) con el siguiente fragmento de código

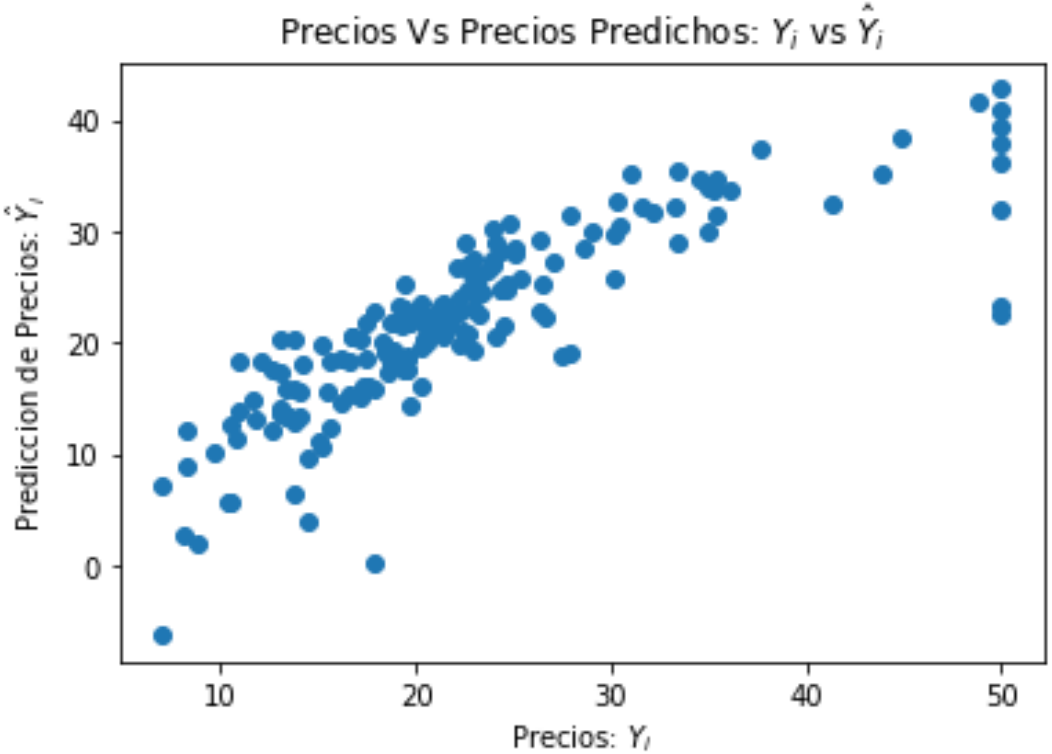
```
1  X_train, X_test, Y_train, Y_test = sklearn.cross_validation. \
2          train_test_split(X, Y, test_size = 0.33, random_state = 5)
3  print(X_train.shape)
4  print(X_test.shape)
5  print(Y_train.shape)
6  print(Y_test.shape)
```

Los conjuntos de datos han sido separados (discretizados) en 66.6% para entrenamiento (`Train`) y 33.3% para pruebas (`Test`).

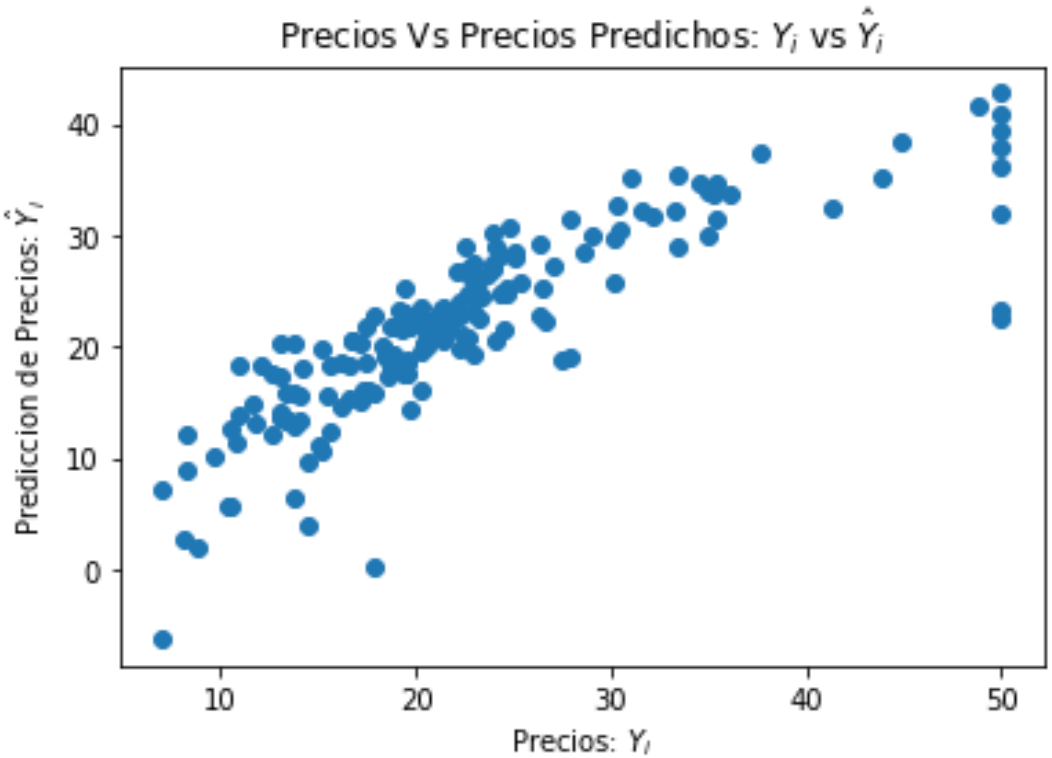
Ahora si ejecutemos la regresion lineal:

```
1  from sklearn.linear_model import LinearRegression
2
3  lm = LinearRegression()
4  lm.fit(X_train, Y_train)
5
6  Y_pred = lm.predict(X_test)
7
8  plt.scatter(Y_test, Y_pred)
9  plt.xlabel("Precios: $Y_i$")
10 plt.ylabel("Prediccion de Precios: $\hat{Y}_i$")
11 plt.title("Precios Vs Precios Predichos: $Y_i$ vs $\hat{Y}_i$")
```

El fragmento de codigo del slide anterior ajustara el modelo usando `X_Train` e `Y_Train`, es decir, ya tenemos el modelos lineal. Ahora predecimos con `X_Test` y tenemos sus predicciones en `Y_Pred`.Para visualizar las diferencias entre los precios reales y los precios predicho realizamos un diagrama de dispercion:



Idealmente, el diagrama de dispersión debería mostrar una linea cuyo comportamiento es la funcion $y = x$.



Para terminar, veamos la información ofrecida en el error cuadrático medio,

Para verificar el nivel de error de este primer modelo, lo podemos hacer usando el `Mean Squared Error`. Este es un procedimiento para medir el cuadrado de los errores. . Básicamente, verificará la diferencia entre el valor real y el valor predicho.

```
1  mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
2  print(mse)
```

Veamos un ejemplo que nos permitira mostrar unas ideas mas profundas del machine learning.

```
1  from sklearn.linear_model import LinearRegression
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  X = np.arange(1, 11).reshape(10, 1)
6  y = np.array([7, 8, 7, 13, 16, 15, 19, 23, 18, 21]).reshape(10, 1)
7
8  plt.plot(X, y, 'ro')
9  plt.show()
10
11 model = LinearRegression()
12 model.fit(X, y)
13 model.coef_
14 model.intercept_
15 a = model.coef_ * X + model.intercept_
16
17 plt.plot(X, y, 'ro', X, a)
18 axes = plt.gca()
19 axes.set_ylim([0, 30])
20 plt.show()
```

Indicadores del rendimiento de la regresion lineal

Obviamente, podemos ver que la línea recta de arriba se adapta bastante bien, pero no lo suficientemente buena. Y necesitamos un enfoque más adecuado. Pero primero, evaluemos qué tan bien está funcionando el modelo numéricamente, calculemos la precisión sobre los datos de entrenamiento:

```
1  print(model.score(X, y))
```

No siempre podemos evaluar algo con solo verlo, ¿verdad? Necesitamos algo que sea más concreto, sí, un `número`. Al mirar los números, tendremos una mejor visión y compararemos fácilmente diferentes cosas. scikit-learn nos proporciona la función de puntuación (score function), cuyos parámetros son similares a la función `fit`.

Y puede ver eso, nuestro modelo ahora tiene una precisión del 84.9% sobre los datos de entrenamiento. Comúnmente, exigimos una mayor precisión, digamos 90% o 95%. Entonces, al observar la precisión actual, podemos decir que nuestro Modelo no está funcionando como esperamos. Así que pensemos en una mejora. Pero, ¿cómo podemos hacer eso?

La primera opción seria ver un modelo polinomial, es decir agregar mas características. Significa que si tenemos X, entonces podemos usar X^2, X^3, etc. como características adicionales. Usemos este enfoque y veamos si podemos mejorar el modelo actual. Primero, tenemos que modificar nuestra matriz X agregando X^2:

```
1  X = np.c_[X, X**2]
```

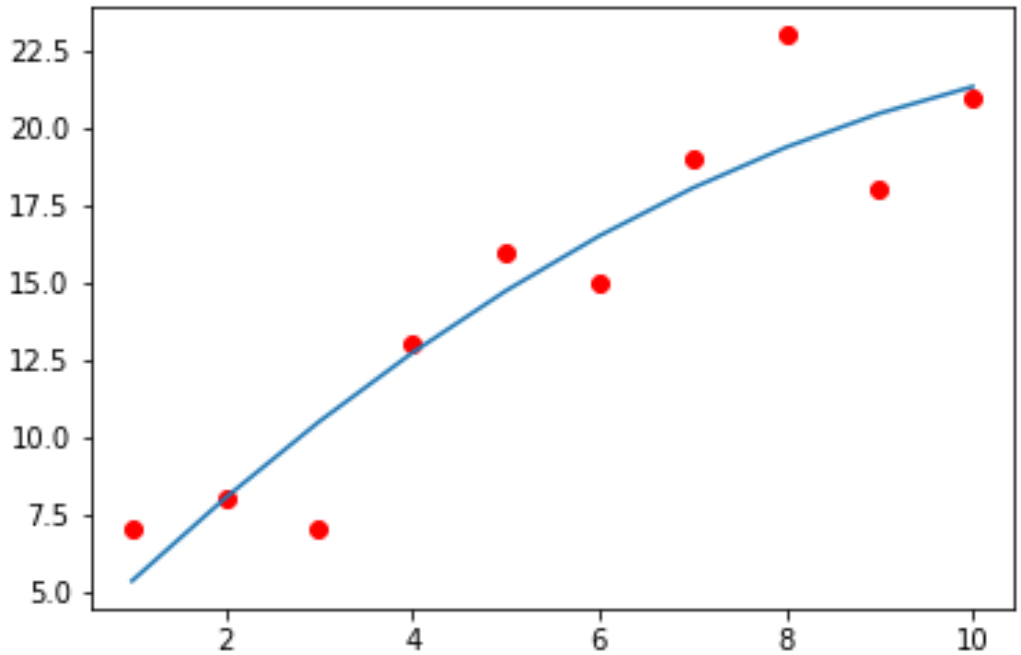
Similar a lo hecho anteriormente, entrenemos nuestro nuevo modelo y luego calculemos el vector de predicción:

```
1  model.fit(X, y)
2  x = np.arange(1, 11, 0.1)
3  x = np.c_[x, x**2]
4  a = np.dot(X, model.coef_.transpose()) + model.intercept_
```

Matematicamente, ahora tenemos que $a = \theta_0 + \theta_1 X + \theta_2 X^2$. Hay que tener en cuenta que ahora tenemos una matriz de datos mas complicados, asi que tendremos que usar la funcion `dot`.Se producira un error si solo utilizamos la operacion de multiplicar usada anteriormente.

Tambien se creo una nueva variable x, el cual almacena numeros de 1 a 10 con saltos de 0.1. Usamos este nuevo x para calcular a dara como resultado un grafico mas suave de a , ya que ahora a no es una linea recta.

```
1  plt.plot(X[:, 0], y, 'ro', x[:, 0], a)
2  plt.show()
```

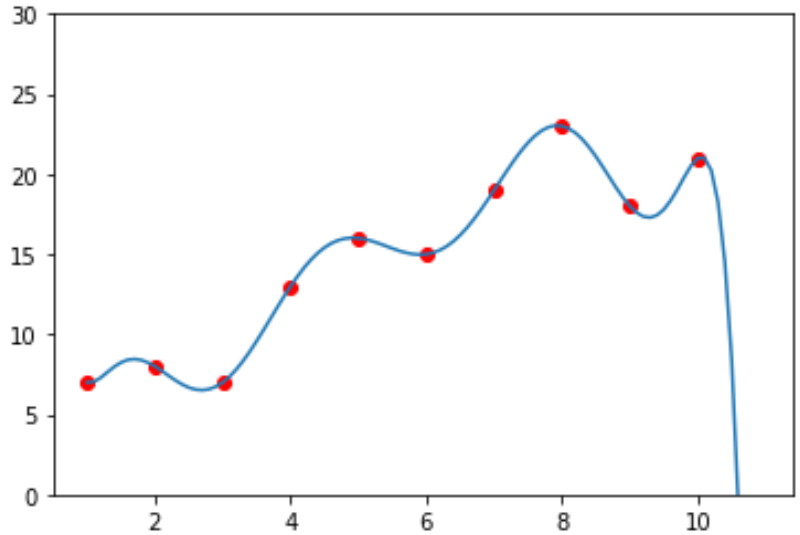


Como se puede ver, ahora obtenemos una línea curva, que parece ajustarse mucho mejor a nuestros datos de entrenamiento. Para ser más concretos, usemos la función `score`:

```
model.score(X, y)
```

Probemos ahora con un polinomio de grado 9.

```
1  X = np.arange(1, 11)
2  X = np.c_[X, X**2, X**3, X**4, X**5, X**6, X**7, X**8, X**9]
3  x = np.arange(1, 11, 0.1)
4  x = np.c_[x, x**2, x**3, x**4, x**5, x**6, x**7, x**8, x**9]
5
6  model.fit(X, y)
7  a = np.dot(x, model.coef_.transpose()) + model.intercept_
8
9  plt.plot(X[:, 0], y, 'ro', x[:, 0], a)
10 axes = plt.gca()
11 axes.set_ylim([0, 30])
12 plt.show()
```



Ahora acabamos de obtener una nueva curva que se ajusta perfectamente a nuestros datos de entrenamiento. Vamos a utilizar la función `score` de nuevo para obtener un número exacto:

```
model.score(X, y)
```

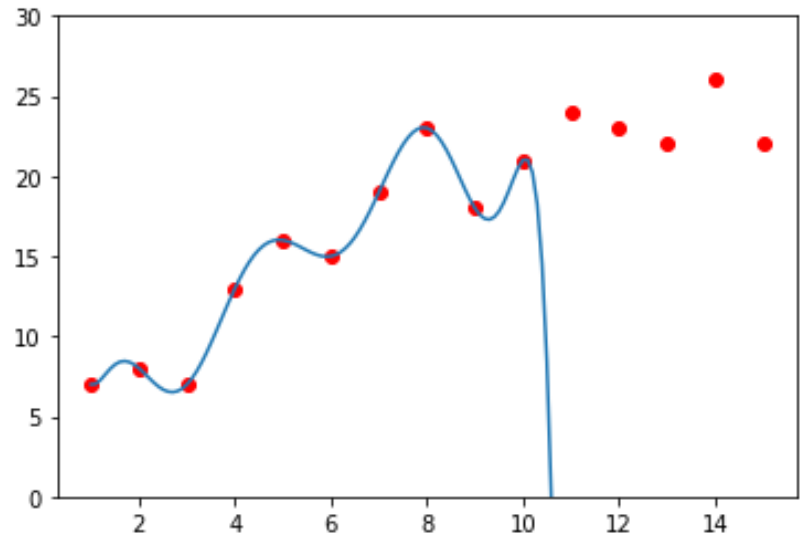
Y como es de esperar nuestra puntuación (score) es casi 100 , en realidad tiene un valor de 99.99999999769595%.

Ahora imaginemos que nuestros datos tienen un total de 15 elementos,

```
1  X = np.arange(1, 16)
2  y = np.append(y, [24, 23, 22, 26, 22])
3
4  plt.plot(X, y, 'ro')
5  plt.show()
```

Veamos qué sucede si utilizamos nuestro excelente modelo obtenido a partir de las características polinómicas de un polinomio de grado 9:

```
plt.plot(X, y, 'ro', x[:, 0], a)
axes = plt.gca()
axes.set_ylim([0, 30])
plt.show()
```



¿Ves lo que estoy viendo? ¡Qué trágico! ¡No parece ajustarse a los nuevos datos en absoluto! ¡Ni siquiera sentimos la necesidad de calcular la precisión de los nuevos datos! Entonces, ¿de qué se trata todo esto?

Como dijimos antes, solo proporcionamos un conjunto fijo de datos de entrenamiento, y el Modelo tendrá que tratar con datos nuevos que nunca antes había visto. ¡Nuevos datos, que pueden variar de manera impredecible en la vida real, estropearon a nuestro modelo capacitado. En el lenguaje del Aprendizaje automático, lo llamamos problema de `overfitting` (o de alta varianza).

Este fenómeno se conoce como overfitting o sobreajuste. Un modelo predictivo debe capturar la esencia del fenómeno que describe, nada más. Christopher M. Bishop, en su libro Pattern Recognition and Machine Learning (Springer, 2006) nos ofrece un gran ejemplo de este problema, con un enfoque muy matemático.

Un modelo de datos es una expresión matemática que describe cómo se relacionan un conjunto de variables. Los modelos se crean y se ajustan a partir de un conjunto completo de datos, en los que tenemos todas las variables de interés. Una vez ajustado, el modelo es útil para poder predecir alguna de las variables a partir de la observación de otras.

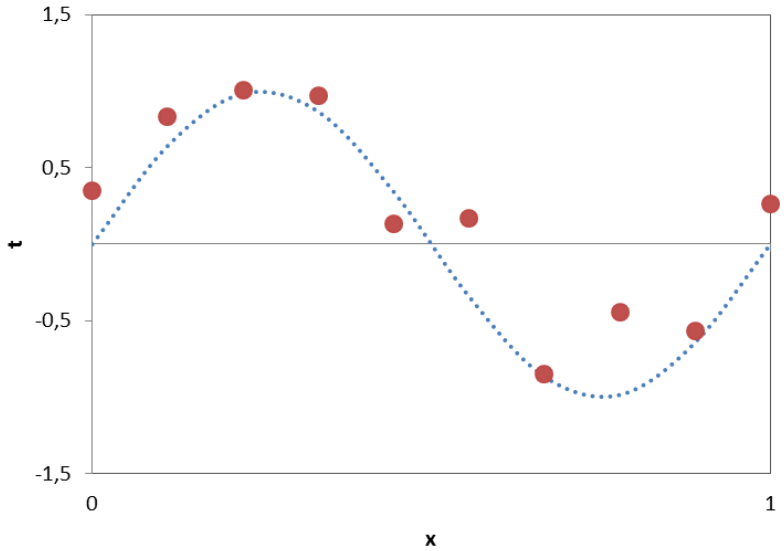
Por ejemplo: podemos crear un modelo que relaciona el mes del año con la temperatura media diaria. Para crear y ajustar el modelo, podríamos observar datos de las temperaturas de los últimos 50 años. Una vez creado el modelo, podemos usarlo para predecir qué temperatura media puedo esperar que se produzca en un mes concreto durante el próximo año. En este caso, el mes sería la variable de entrada (x) y la temperatura la variable objetivo o variable predicha (t).

Dependiendo del fenómeno investigado, la relación entre variables de entrada y variables objetivo es más o menos fuerte. Esa relación define la capacidad predictiva del modelo. Rara vez un modelo puede describir por completo un fenómeno, lo que sería un modelo determinista: lo habitual es que las predicciones sean imperfectas.

Por ejemplo, los modelos meteorológicos toman como variables de entrada la temperatura, humedad, presión atmosférica y otros datos similares para predecir la temperatura futura y la probabilidad de lluvia. Los modelos meteorológicos no son infalibles. Cuando el modelo trata de predecir el tiempo con más días de antelación, más puede fallar.

Bishop nos propone un interesante ejercicio para explicar el over-fitting. Para ello, crea una relación artificial entre una variable de entrada x y una variable objetivo t, lo que permite crear datos sintéticos. Posteriormente, trata de diseñar un modelo que describa esos datos sin usar información de cómo han sido creados.

Para crear los datos sintéticos, Bishop usa una función conocida, $t = sin(2\pi x)$, a la cual añade un poco de ruido, sumando a la variable t una cierta cantidad aleatoria. Usando este método, generamos un conjunto de datos formado por 10 valores de x uniformemente distribuidos entre 0 y 1 con sus correspondientes valores de t.



La forma en que hemos generado estos datos captura una propiedad muy habitual de los datos que encontramos en problemas reales: poseen una regularidad subyacente, que es la que queremos capturar y modelizar, pero al observar un conjunto individual de datos estos se ven corrompidos por algún tipo de ruido aleatorio. Este ruido puede surgir por la propia naturaleza aleatoria del proceso, pero lo más habitual es que se deba a que no estamos observando alguna fuente de variabilidad que afecta al fenómeno estudiado.

Un par de ideas sobre el riesgo de overfitting

La primera es que debe existir un equilibrio entre la cantidad de datos que tenemos y la complejidad del modelo. En nuestro ejemplo, cuando usamos un modelo con 10 parámetros para describir un problema para el que tenemos 10 datos, el resultado es previsible: vamos a construir un modelo a medida de los datos que tenemos, estamos resolviendo un sistema de ecuaciones con tantas incógnitas como ecuaciones. Dicho de otra manera: si este modelo con 10 parámetros lo hubiésemos ajustado con un total de 100 datos en lugar de 10, seguramente funcionaría mejor que un modelo más básico.

La segunda idea básica: otra forma de detectar que nuestro modelo padece over-fitting es observar el valor de los parámetros. Cuando se produce over-fitting, los parámetros de nuestro modelo crecen de forma desmesurada, haciendo oscilar nuestra curva abruptamente. De hecho, una manera de evitar el over-fitting sin necesidad determinar manualmente la complejidad del modelo es añadir a la función de error que queremos minimizar la suma de los cuadrados de los parámetros. De esta forma penalizamos los modelos más complejos, promoviendo que los parámetros sean más suaves.

Por el contrario, ¿qué pasará si usamos solo una característica como lo hicimos al principio (o podemos decir que proporcionamos un conjunto de datos que es poco informativo)? Ya has visto que resultó en una precisión muy baja, que tampoco es lo que esperábamos. Llamamos a este problema UNDERFITTING.

Overfitting y Underfitting, en ambos casos, son algo que tratamos de evitar. Y, en general, se enfrentará a estos problemas todo el tiempo que trabaje con Machine Learning. Por supuesto, hay muchas maneras de lidiar con ellos.Una de estas maneras muy usada es la que hemos en anteriores slides, la de partir la data en dos conjuntos de datos.

Utilizaremos algoritmos de Machine Learning en Python para resolver un problema de Regresión Logística. A partir de un conjunto de datos de entrada (características), nuestra salida será discreta (y no continua) por eso utilizamos Regresión Logística (y no Regresión Lineal). La Regresión Logística es un Algoritmo Supervisado y se utiliza para clasificación.

Vamos a clasificar problemas con dos posibles estados "SI/NO": binario o un número finito de "etiquetas" o "clases": múltiple. Algunos Ejemplos de Regresión Logística son:

- Clasificar si el correo que llega es Spam o No es Spam
- Dados unos resultados clínicos de un tumor clasificar en "Benigno" o "Maligno".
- El texto de un artículo a analizar es: Entretenimiento, Deportes, Política ó Ciencia
- A partir de historial bancario conceder un crédito o no

Para nuestro ejercicio he creado un archivo csv (`usuarios_win_mac_lin.csv`) con datos de entrada a modo de ejemplo para clasificar si el usuario que visita un sitio web usa como sistema operativo Windows, Macintosh o Linux.

Nuestra información de entrada son 4 características que tomé de una web que utiliza Google Analytics y son:

1. Duración de la visita en Segundos
2. Cantidad de Páginas Vistas durante la Sesión
3. Cantidad de Acciones del usuario (click, scroll, uso de checkbox, sliders,etc)
4. Suma del Valor de las acciones (cada acción lleva asociada una valoración de importancia)

Como la salida es discreta, asignaremos los siguientes valores a las etiquetas:

0 Windows
1 Macintosh
2 Linux

La muestra es pequeña: son 170 registros para poder comprender el ejercicio, pero recordemos que para conseguir buenos resultados siempre es mejor contar con un número abundante de datos que darán mayor exactitud a las predicciones y evitarán problemas de overfitting u underfitting. (Por decir algo, de mil a 5 mil registros no estaría mal).

Identificar Sistema Operativo de los usuarios

Para comenzar hacemos los Import necesarios con los paquetes que utilizaremos en el Ejercicio.

```
1  import pandas as pd
2  import numpy as np
3  from sklearn import linear_model
4  from sklearn import model_selection
5  from sklearn.metrics import classification_report
6  from sklearn.metrics import confusion_matrix
7  from sklearn.metrics import accuracy_score
8  import matplotlib.pyplot as plt
9  import seaborn as sb
```

Leemos el archivo csv y lo asignamos mediante Pandas a la variable dataframe. Mediante el método dataframe.head() vemos en pantalla los 5 primeros registros.

```
1  dataframe = pd.read_csv("usuarios_win_mac_lin.csv")
2  dataframe.head()
```

	duracion	paginas	acciones	valor	clase
0	7.0	2	4	8	2
1	21.0	2	6	6	2
2	57.0	2	4	4	2
3	101.0	3	6	12	2
4	109.0	2	6	12	2

A continuación llamamos al método dataframe.describe() que nos dará algo de información estadística básica de nuestro set de datos. La Media, el desvío estándar, valores mínimo y máximo de cada característica.

```
1  dataframe.describe()
```

	duracion	paginas	acciones	valor	clase
count	170.000000	170.000000	170.000000	170.000000	170.000000
mean	111.075729	2.041176	8.723529	32.676471	0.752941
std	202.453200	1.500911	9.136054	44.751993	0.841327
min	1.000000	1.000000	1.000000	1.000000	0.000000
25%	11.000000	1.000000	3.000000	8.000000	0.000000
50%	13.000000	2.000000	6.000000	20.000000	0.000000
75%	108.000000	2.000000	10.000000	36.000000	2.000000
max	898.000000	9.000000	63.000000	378.000000	2.000000

Luego analizaremos cuantos resultados tenemos de cada tipo usando la función groupby y vemos que tenemos 86 usuarios "Clase 0", es decir Windows, 40 usuarios Mac y 44 de Linux.

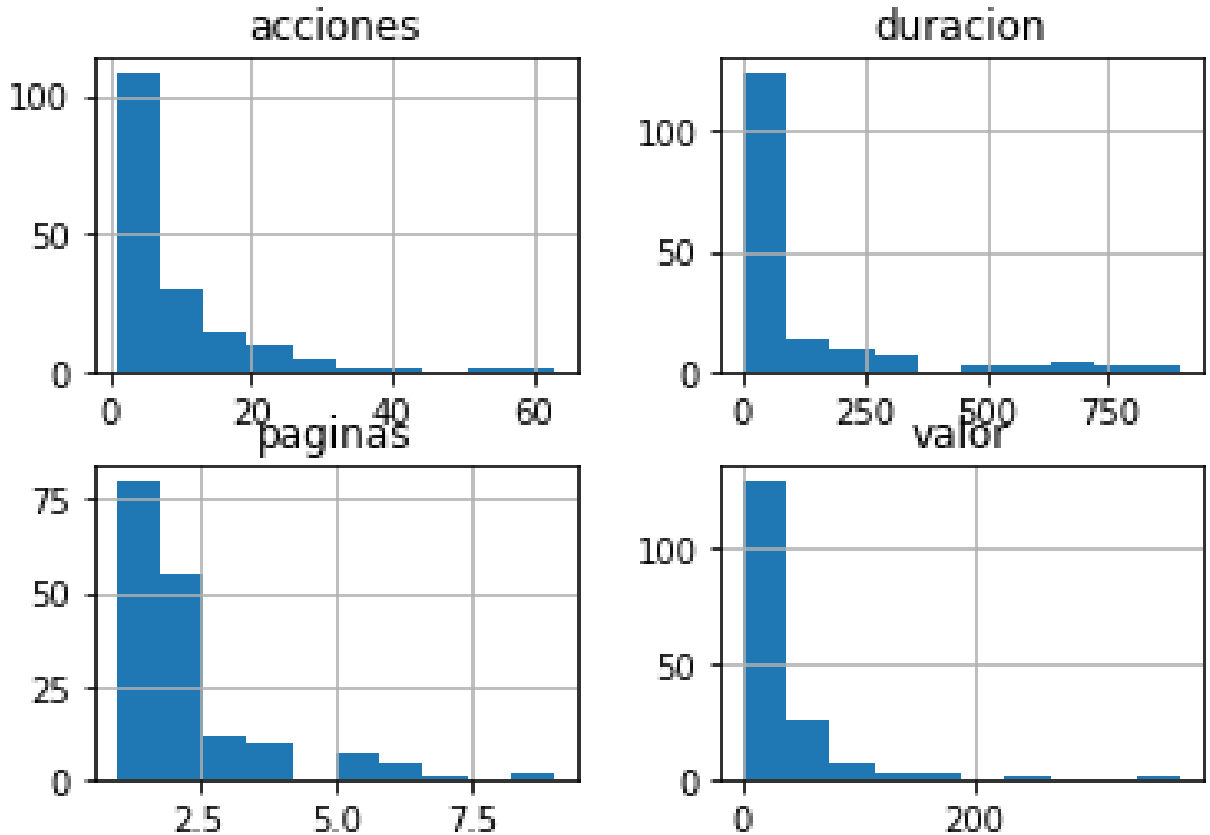
```
1  print(dataframe.groupby('clase').size())
```

Visualización de Datos Antes de empezar a procesar el conjunto de datos, vamos a hacer unas visualizaciones que muchas veces nos pueden ayudar a comprender mejor las características de la información con la que trabajamos y su correlación.

Primero visualizamos en formato de historial los cuatro Features de entrada con nombres "duración", "páginas","acciones" y "valor" podemos ver gráficamente entre qué valores se comprenden sus mínimos y máximos y en qué intervalos concentran la mayor densidad de registros.

```
dataframe.drop(['clase'],1).hist()
plt.show()
```

Visualización de Datos

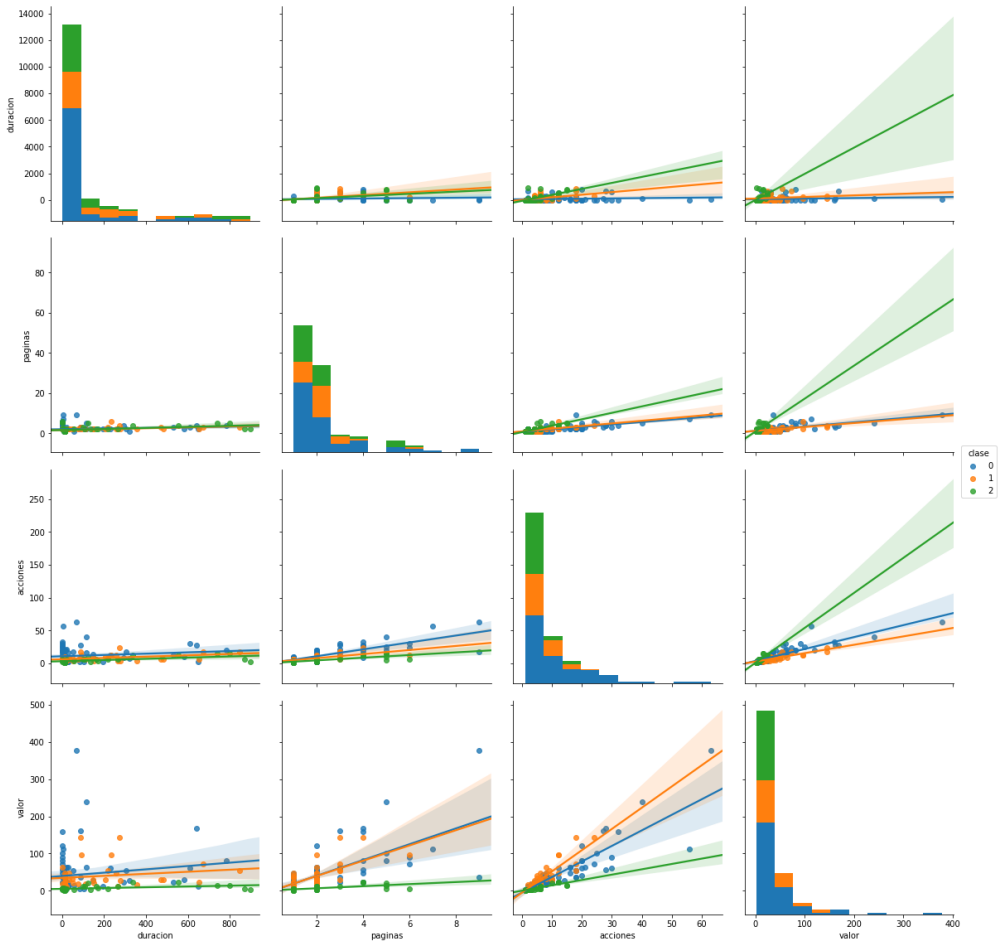


Y también podemos interrelacionar las entradas de a pares, para ver como se concentran linealmente las salidas de usuarios por colores: Sistema Operativo Windows en azul, Macintosh en verde y Linux en rojo.

```
1  sb.pairplot(dataframe.dropna(), hue='clase',size=4, \
2          vars=["duracion", "paginas","acciones","valor"],kind='reg')
```

Topicos de Machine Learning

Regresión Logística con SKLearn



Creamos el Modelo de Regresión Logística Ahora cargamos las variables de las 4 columnas de entrada en X excluyendo la columna "clase" con el método drop(). En cambio agregamos la columna "clase" en la variable y. Ejecutamos X.shape para comprobar la dimensión de nuestra matriz con datos de entrada de 170 registros por 4 columnas.

```
1  X = np.array(dataframe.drop(['clase'],1))
2  y = np.array(dataframe['clase'])
3  X.shape
```

Y creamos nuestro modelo y hacemos que se ajuste (fit) a nuestro conjunto de entradas X y salidas 'y'.

```
model = linear_model.LogisticRegression()
model.fit(X,y)
```

Una vez compilado nuestro modelo, le hacemos clasificar todo nuestro conjunto de entradas X utilizando el método "predict(X)" y revisamos algunas de sus salidas y vemos que coincide con las salidas reales de nuestro archivo csv.

```
predictions = model.predict(X)
print(predictions)[0:5]
```

Y confirmamos cuan bueno fue nuestro modelo utilizando model.score() que nos devuelve la precisión media de las predicciones, en nuestro caso del 77%.

```
1  model.score(X,y)
```

Validación de nuestro modelo Una buena práctica en Machine Learning es la de subdividir nuestro conjunto de datos de entrada en un set de entrenamiento y otro para validar el modelo (que no se utiliza durante el entrenamiento y por lo tanto la máquina desconoce). Esto evitará problemas en los que nuestro algoritmo pueda fallar por "sobregeneralizar" el conocimiento.

Para ello, subdividimos nuestros datos de entrada en forma aleatoria (mezclados) utilizando 80% de registros para entrenamiento y 20% para validar.

```
1  validation_size = 0.20
2  seed = 7
3  X_train, X_validation, Y_train, Y_validation = \
4          model_selection.train_test_split \
5          (X, y, test_size=validation_size, random_state=seed)
```

Volvemos a compilar nuestro modelo de Regresión Logística pero esta vez sólo con 80% de los datos de entrada y calculamos el nuevo scoring que ahora nos da 74%.

```
1  name='Logistic Regression'
2  kfold = model_selection.KFold(n_splits=10, random_state=seed)
3  cv_results = model_selection.cross_val_score(model, X_train,
4                                                Y_train, cv=kfold,
5                                                scoring='accuracy')
6  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
7  print(msg)
```

Y ahora hacemos las predicciones -en realidad clasificación- utilizando nuestro "cross validation set", es decir del subconjunto que habíamos apartado. En este caso vemos que los aciertos fueron del 85

```
1  predictions = model.predict(X_validation)
2  print(accuracy_score(Y_validation, predictions))
```

Finalmente vemos en pantalla la "matriz de confusión" donde muestra cuantos resultados equivocados tuvo de cada clase (los que no están en la diagonal), por ejemplo predijo 3 usuarios que eran Mac como usuarios de Windows y predijo a 2 usuarios Linux que realmente eran de Windows.

Reporte de Resultados del Modelo

```
1  print(confusion_matrix(Y_validation, predictions))
```

También podemos ver el reporte de clasificación con nuestro conjunto de Validación. En nuestro caso vemos que se utilizaron como "soporte" 18 registros windows, 6 de mac y 10 de Linux (total de 34 registros). Podemos ver la precisión con que se acertaron cada una de las clases y vemos que por ejemplo de Macintosh tuvo 3 aciertos y 3 fallos (0.5 recall). La valoración que de aqui nos conviene tener en cuenta es la de F1-score, que tiene en cuenta la precisión y recall. El promedio de F1 es de 84

```
1  print(classification_report(Y_validation, predictions))
```

Clasificación (o predicción) de nuevos valores Como último ejercicio, vamos a inventar los datos de entrada de navegación de un usuario ficticio que tiene estos valores:

Tiempo Duración: 10
Paginas visitadas: 3
Acciones al navegar: 5
Valoración: 9
Lo probamos en nuestro modelo y vemos que lo clasifica como un usuario tipo 2, es decir, de Linux.

```
X_new = pd.DataFrame({'duracion': [10], 'paginas': [3], \
        'acciones': [5], 'valor': [9]})
model.predict(X_new)
```

K-Means es un algoritmo no supervisado de Clustering. Se utiliza cuando tenemos un montón de datos sin etiquetar. El objetivo de este algoritmo es el de encontrar "K" grupos (clusters) entre los datos crudos.

Cómo funciona K-Means

El algoritmo trabaja iterativamente para asignar a cada "punto" (las filas de nuestro conjunto de entrada forman una coordenada) uno de los "K" grupos basado en sus características. Son agrupados en base a la similitud de sus features (las columnas).

Como resultado de ejecutar el algoritmo tendremos:

Los "centroids" de cada grupo que serán unas "coordenadas" de cada uno de los K conjuntos que se utilizarán para poder etiquetar nuevas muestras.
Etiquetas para el conjunto de datos de entrenamiento. Cada etiqueta perteneciente a uno de los K grupos formados.
Los grupos se van definiendo de manera "orgánica", es decir que se va ajustando su posición en cada iteración del proceso, hasta que converge el algoritmo. Una vez hallados los centroids deberemos analizarlos para ver cuales son sus características únicas, frente a la de los otros grupos. Estos grupos son las etiquetas que genera el algoritmo.

Casos de Uso de K-Means El algoritmo de Clustering K-means es usado uno de los más usados para encontrar grupos ocultos, o sospechados en teoría sobe un conjunto de datos no etiquetado. Esto puede servir para confirmar -o desterrar- alguna teoría que teníamos asumida de nuestros datos. Y también puede ayudarnos a descubrir relaciones asombrosas entre conjuntos de datos, que de manera manual, no hubiéramos reconocido. Una vez que el algoritmo ha ejecutado y obtenido las etiquetas, será fácil clasificar nuevos valores o muestras entre los grupos obtenidos.

Algunos casos de uso son: *Segmentación por Comportamiento*: relacionar el carrito de compras de un usuario, sus tiempos de acción e información del perfil.
Categorización de Inventario: agrupar productos por actividad en sus ventas
Detectar anomalías o actividades sospechosas: según el comportamiento en una web reconocer un troll -o un bot- de un usuario normal

Datos de Entrada para K-Means Las "features" o características que utilizaremos como entradas para aplicar el algoritmo k-means deberán ser de valores numéricos, continuos en lo posible. En caso de valores categóricos (por ej. Hombre/Mujer o Ciencia Ficción, Terror, Novela,etc) se puede intentar pasarlo a valor numérico, pero no es recomendable pues no hay una "distancia real" -como en el caso de géneros de película o libros-. Además es recomendable que los valores utilizados estén normalizados, manteniendo una misma escala. En algunos casos también funcionan mejor datos porcentuales en vez de absolutos. No conviene utilizar features que estén correlacionados o que sean escalares de otros.

El Algoritmo K-means El algoritmo utiliza una proceso iterativo en el que se van ajustando los grupos para producir el resultado final. Para ejecutar el algoritmo deberemos pasar como entrada el conjunto de datos y un valor de K. El conjunto de datos serán las características o features para cada punto. Las posiciones iniciales de los K centroids serán asignadas de manera aleatoria de cualquier punto del conjunto de datos de entrada.

Luego se itera en dos pasos:

1. Paso de Asignación de datos
   En este paso, cada "fila" de nuestro conjunto de datos se asigna al centroide más cercano basado en la distancia cuadrada Euclideana. Se utiliza la siguiente fórmula (donde dist() es la distancia Euclideana standard):

$$argmin_{c_i \in C} dist(c_i, x)^2$$

2. Paso de actualización de Centroid
   En este paso los centroid de cada grupo son recalculados. Esto se hace tomando una media de todos los puntos asignados en el paso anterior.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

El algoritmo itera entre estos pasos hasta cumplir un criterio de detención:

* si no hay cambios en los puntos asignados a los grupos,
* o si la suma de las distancias se minimiza,
* o se alcanza un número máximo de iteraciones.

El algoritmo converge a un resultado que puede ser el óptimo local, por lo que será conveniente volver a ejecutar más de una vez con puntos iniciales aleatorios para confirmar si hay una salida mejor.

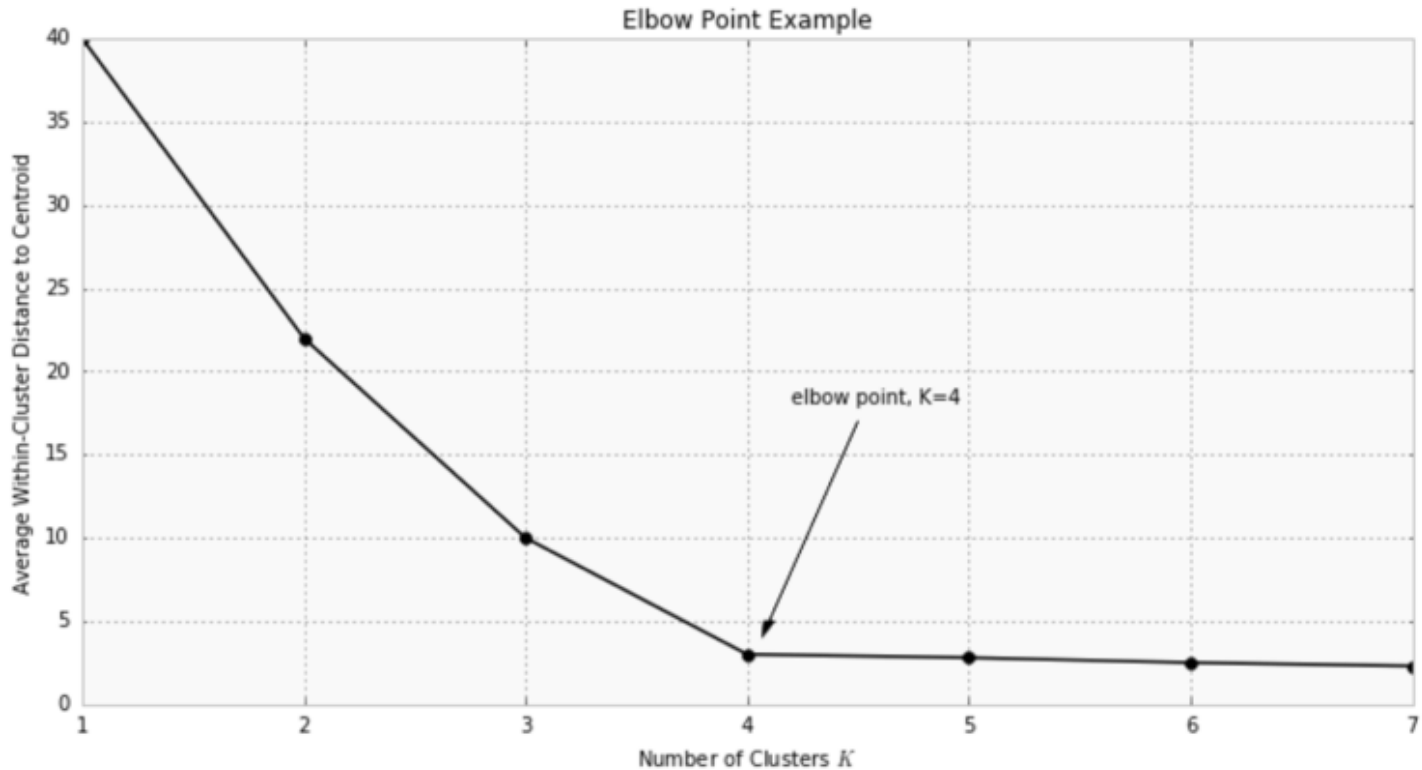
Elegir el valor de K

Este algoritmo funciona pre-seleccionando un valor de K. Para encontrar el número de clusters en los datos, deberemos ejecutar el algoritmo para un rango de valores K, ver los resultados y comparar características de los grupos obtenidos. En general no hay un modo exacto de determinar el valor K, pero se puede estimar con aceptable precisión siguiendo la siguiente técnica:

Una de las métricas usada para comparar resultados es la distancia media entre los puntos de datos y su centroid. Como el valor de la media diminuirá a medida de aumentemos el valor de K, deberemos utilizar la distancia media al centroide en función de K y entontrar el "punto codo", donde la tasa de descenso se "afila". Aquí vemos una gráfica a modo de ejemplo:

Topicos de Machine Learning

Ejemplo de Clusterizacion usando K-Means



Un ejemplo K-Means en Python con Sklearn Como ejemplo utilizaremos de entradas un conjunto de datos que obtuve de un proyecto propio, en el que se analizaban rasgos de la personalidad de usuarios de Twitter. He filtrado a 140 "famosos" del mundo en diferentes areas: deporte, cantantes, actores, etc. Basado en una metodología de psicología conocida como "Ocean: The Big Five" tendemos como características de entrada:

* usuario (el nombre en Twitter)
* "op" = Openness to experience – grado de apertura mental a nuevas experiencias, curiosidad, arte
* "co" =Conscientiousness – grado de orden, prolijidad, organización
* "ex" = Extraversion – grado de timidez, solitario o participación ante el grupo social
* "ag" = Agreeableness – grado de empatía con los demás, temperamento
* "ne" = Neuroticism, – grado de neuroticismo, nervioso, irritabilidad, seguridad en sí mismo.
* Wordcount – Cantidad promedio de palabras usadas en sus tweets
* Categoria – Actividad laboral del usuario (actor, cantante, etc.)

Utilizaremos el algoritmo K-means para que agrupe estos usuarios -no por su actividad laboral- si no, por sus similitudes en la personalidad. Si bien tenemos 8 columnas de entrada, sólo utilizaremos 3 en este ejemplo, de modo que podamos ver en un gráfico tridimensional -y sus proyecciones a 2D- los grupos resultantes. Pero para casos reales, podemos utilizar todas las dimensiones que necesitemos. Una de las hipótesis que podríamos tener es: "Todos los cantantes tendrán personalidad parecida" (y así con cada rubro laboral). Pues veremos si lo probamos, o por el contrario, los grupos no están relacionados necesariamente con la actividad de estas Celebridades.

Agrupar usuarios Twitter de acuerdo a su personalidad con K-means

Comenzaremos importando las librerías que nos asistirán para ejecutar el algoritmo y graficar.

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import seaborn as sb
5  from sklearn.cluster import KMeans
6  from sklearn.metrics import pairwise_distances_argmin_min
7  from mpl_toolkits.mplot3d import Axes3D
8  plt.rcParams['figure.figsize'] = (16, 9)
9  plt.style.use('ggplot')
```

Importamos el archivo.csv -para simplificar, suponemos que el archivo se encuentra en el mismo directorio que el notebook- y vemos los primeros 5 registros del archivo tabulados.

```
dataframe = pd.read_csv("analisis.csv")
dataframe.head()
```

También podemos ver una tabla de información estadística que nos provee Pandas dataframe:

```
dataframe.describe()
```

El archivo contiene diferenciadas 9 categorías -actividades laborales- que son:

1. Actor/actriz
2. Cantante
3. Modelo
4. Tv, series
5. Radio
6. Tecnología
7. Deportes
8. Politica
9. Escritor

Para saber cuantos registros tenemos de cada uno hacemos:

```
1  print(dataframe.groupby('categoria').size())
```

Como vemos tenemos 34 cantantes, 27 actores, 17 deportistas, 16 políticos,etc.

Visualización de Datos Veremos graficamente nuestros datos para tener una idea de la dispersión de los mismos:

```
dataframe.drop(['categoria'],1).hist()
plt.show()
```

En este caso seleccionamos 3 dimensiones: op, ex y ag y las cruzamos para ver si nos dan alguna pista de su agrupación y la relación con sus categorías.

```
sb.pairplot(dataframe.dropna(), hue='categoria',size=4, \
        vars=["op","ex","ag"],kind='scatter')
```

Si revisamos la gráfica no pareciera que hay algún tipo de agrupación o correlación entre los usuarios y sus categorías.

Definimos la entrada Concretamos la estructura de datos que utilizaremos para alimentar el algoritmo. Como se ve, sólo cargamos las columnas op, ex y ag en nuestra variable X.

```
1  X = np.array(dataframe[["op","ex","ag"]])
2  y = np.array(dataframe['categoria'])
3  X.shape
```

Ahora veremos una gráfica en 3D con 9 colores representando las categorías.

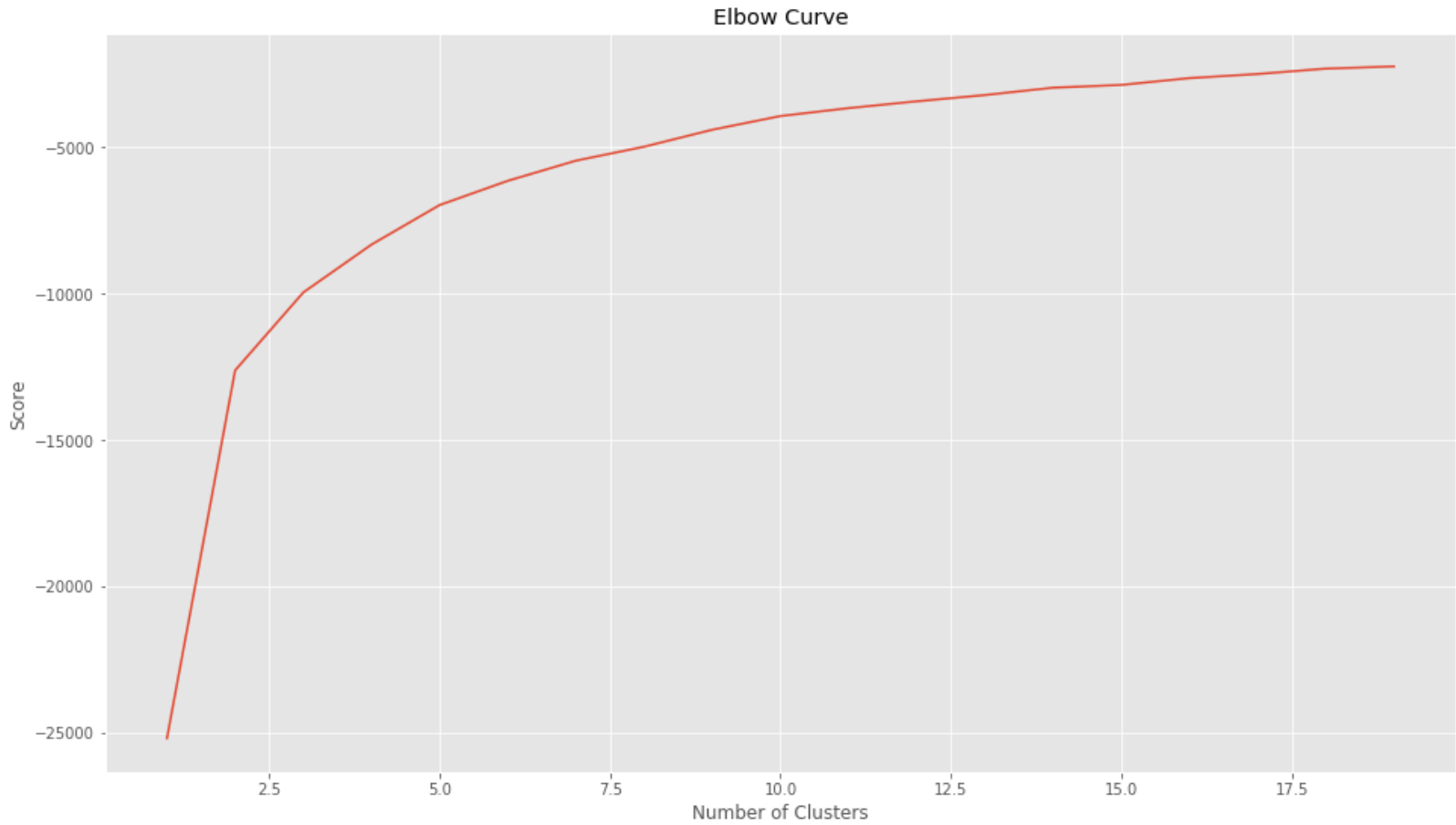
```
1  fig = plt.figure()
2  ax = Axes3D(fig)
3  colores=['blue','red','green','blue','cyan','yellow','orange','black','pink',
4           'brown','purple']
5  asignar=[]
6  for row in y:
7          asignar.append(colores[row])
8  ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=asignar,s=60)
```

Veremos si con K-means, podemos "pintar" esta misma gráfica de otra manera, con clusters diferenciados.

Obtener el valor K

Vamos a hallar el valor de K haciendo una gráfica e intentando hallar el "punto de codo" que comentábamos antes. Este es nuestro resultado:

```
1   Nc = range(1, 20)
2   kmeans = [KMeans(n_clusters=i) for i in Nc]
3   kmeans
4   score = [kmeans[i].fit(X).score(X) for i in range(len(kmeans))]
5   score
6   plt.plot(Nc,score)
7   plt.xlabel('Number of Clusters')
8   plt.ylabel('Score')
9   plt.title('Elbow Curve')
10  plt.show()
```



Realmente la curva es bastante "suave". Considero a 5 como un buen número para K. Según vuestro criterio podría ser otro.

Ejecutamos K-Means Ejecutamos el algoritmo para 5 clusters y obtenemos las etiquetas y los centroids.

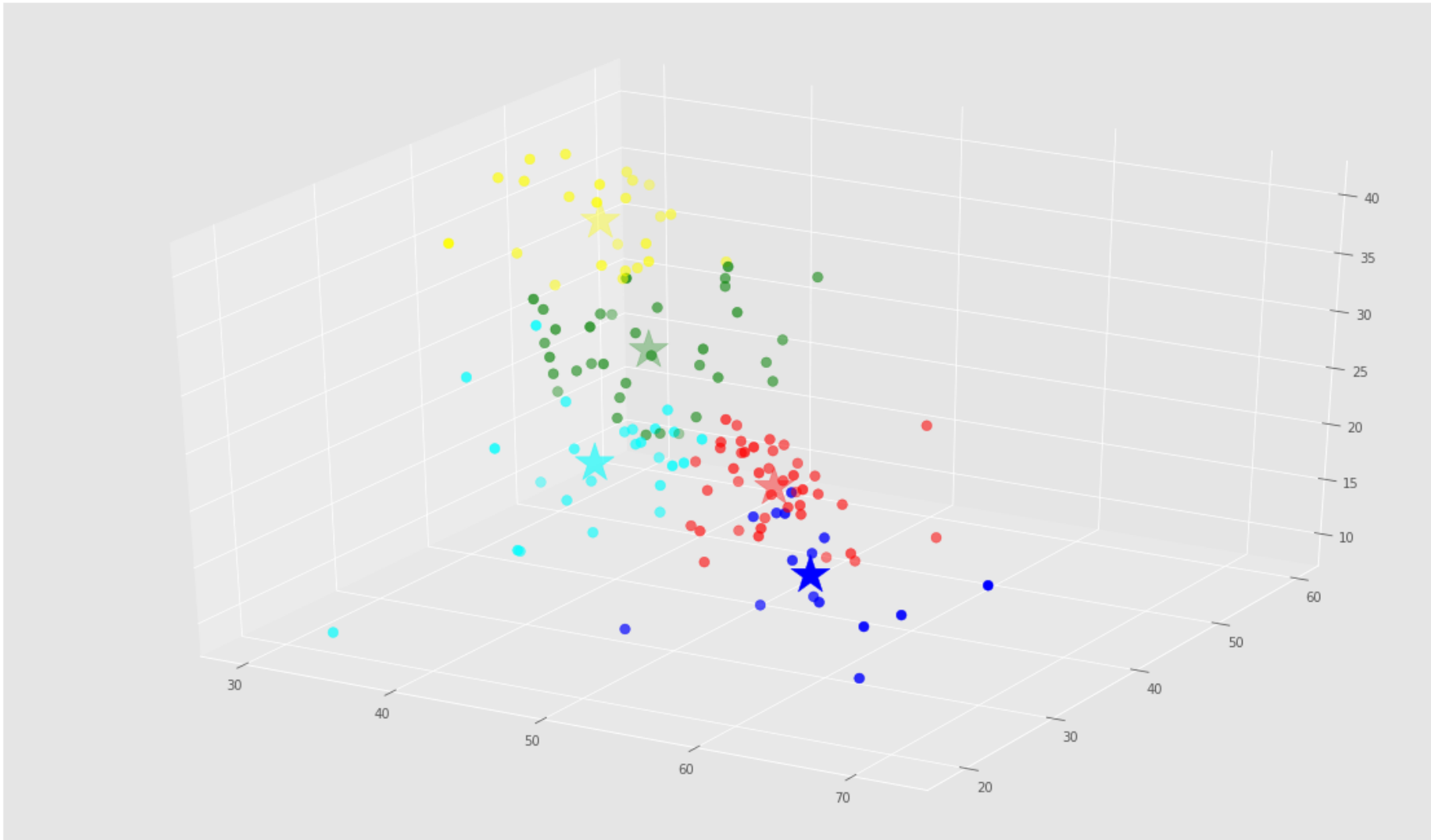
```
kmeans = KMeans(n_clusters=5).fit(X)
centroids = kmeans.cluster_centers_
print(centroids)
```

Ahora veremos esto en una gráfica 3D con colores para los grupos y veremos si se diferencian: (las estrellas marcan el centro de cada cluster)

```
1   # prediciendo los clusteres
2   labels = kmeans.predict(X)
3   # Obteniendo centroids
4   C = kmeans.cluster_centers_
5   colores=['red','green','blue','cyan','yellow']
6   asignar=[]
7   for row in labels:
8           asignar.append(colores[row])
9
10  fig = plt.figure()
11  ax = Axes3D(fig)
12  ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=asignar,s=60)
13  ax.scatter(C[:, 0], C[:, 1], C[:, 2], marker='*', c=colores, s=1000)
```

Topicos de Machine Learning

Ejemplo de Clusterizacion usando K-Means



Aqui podemos ver que el Algoritmo de K-Means con K=5 ha agrupado a los 140 usuarios Twitter por su personalidad, teniendo en cuenta las 3 dimensiones que utilizamos: Openess, Extraversion y Agreeablenes. Pareciera que no hay necesariamente una relación en los grupos con sus actividades de Celebrity.

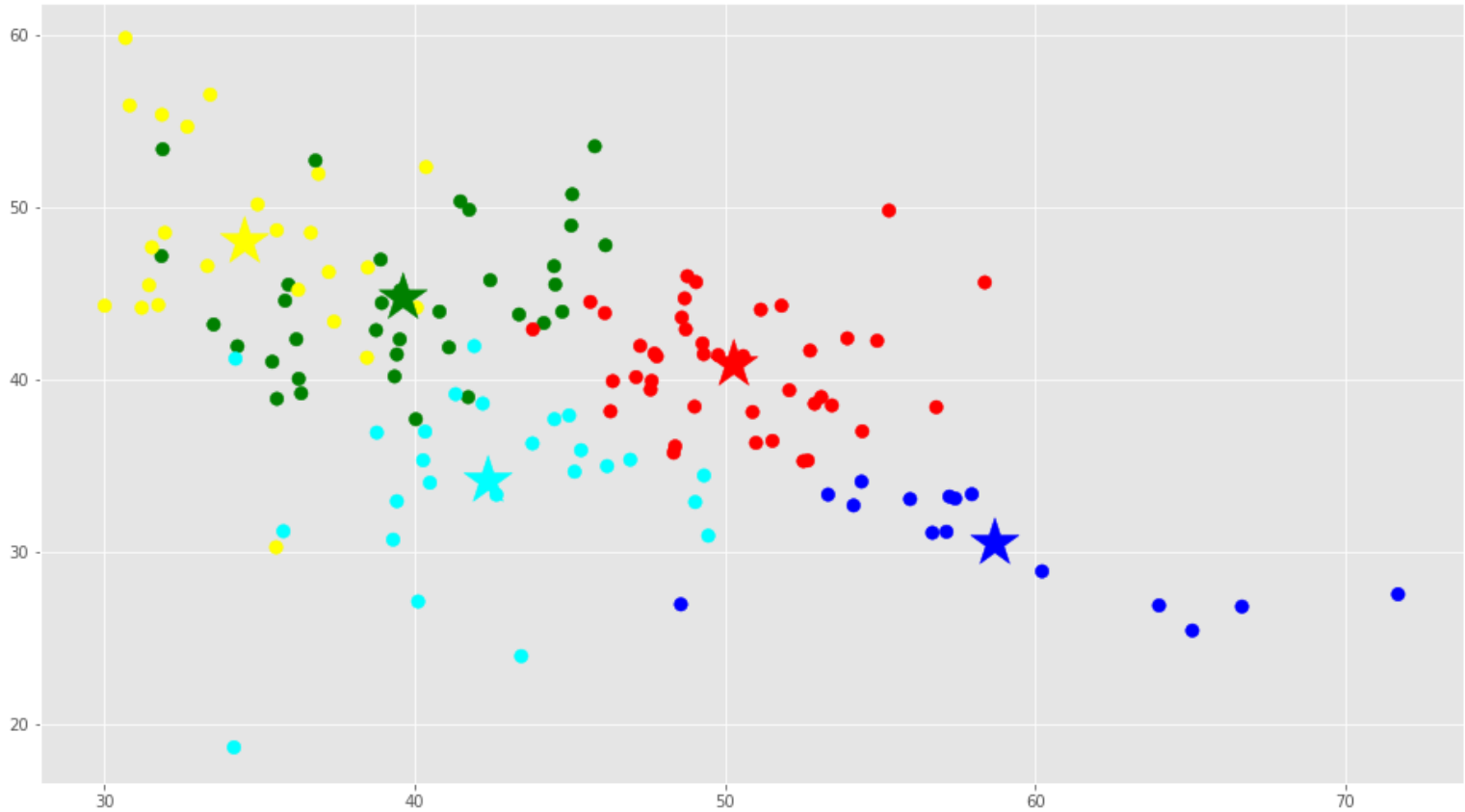
Haremos 3 gráficas en 2 dimensiones con las proyecciones a partir de nuestra gráfica 3D para que nos ayude a visualizar los grupos y su clasificación:

```
# Getting the values and plotting it
f1 = dataframe['op'].values
f2 = dataframe['ex'].values

plt.scatter(f1, f2, c=asignar, s=70)
plt.scatter(C[:, 0], C[:, 1], marker='*', c=colores, s=1000)
plt.show()
```

Topicos de Machine Learning

Ejemplo de Clusterizacion usando K-Means

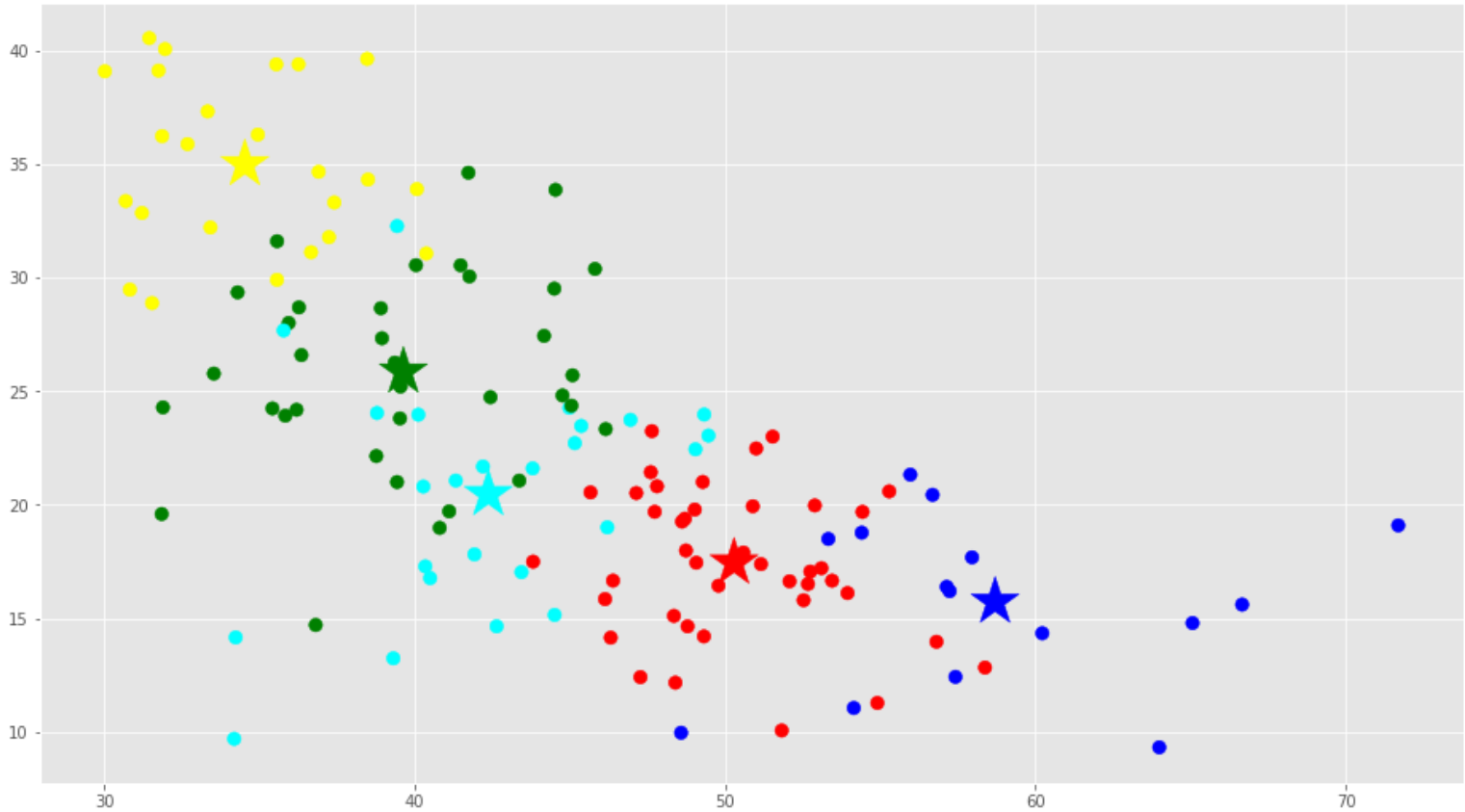


```
# Getting the values and plotting it
f1 = dataframe['op'].values
f2 = dataframe['ag'].values

plt.scatter(f1, f2, c=asignar, s=70)
plt.scatter(C[:, 0], C[:, 2], marker='*', c=colores, s=1000)
plt.show()
```

Topicos de Machine Learning

Ejemplo de Clusterizacion usando K-Means

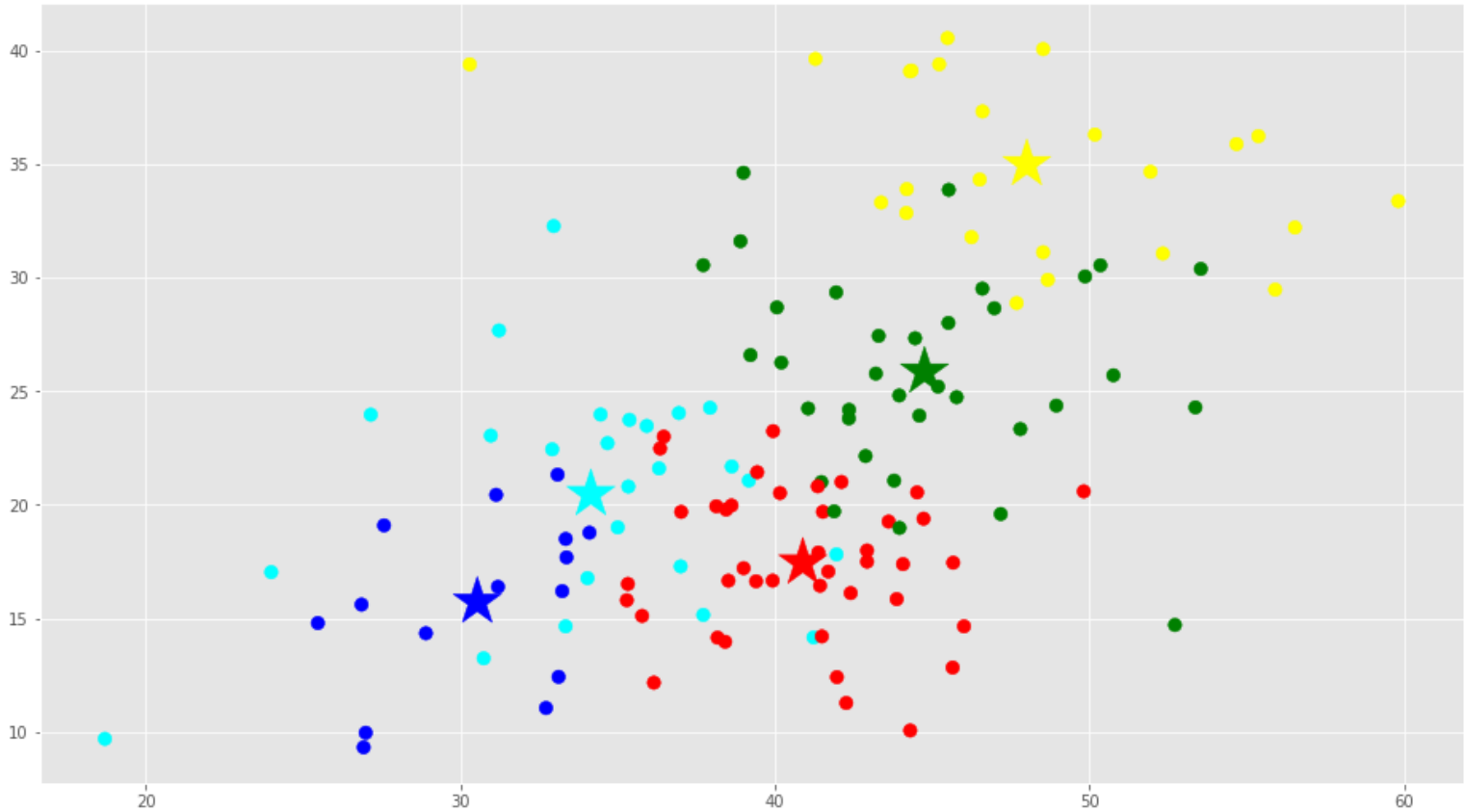


```
f1 = dataframe['ex'].values
f2 = dataframe['ag'].values

plt.scatter(f1, f2, c=asignar, s=70)
plt.scatter(C[:, 1], C[:, 2], marker='*', c=colores, s=1000)
plt.show()
```

Topicos de Machine Learning

Ejemplo de Clusterizacion usando K-Means



En estas gráficas vemos que están bastante bien diferenciados los grupos.

Podemos ver cada uno de los clusters cuantos usuarios tiene:

```
1  copy =  pd.DataFrame()
2  copy['usuario']=dataframe['usuario'].values
3  copy['categoria']=dataframe['categoria'].values
4  copy['label'] = labels;
5  cantidadGrupo =  pd.DataFrame()
6  cantidadGrupo['color']=colores
7  cantidadGrupo['cantidad']=copy.groupby('label').size()
8  cantidadGrupo
```

Y podemos ver la diversidad en rubros laborales de cada uno. Por ejemplo en el grupo 0 (rojo), vemos que hay de todas las actividades laborales aunque predominan de actividad 1 y 2 correspondiente a Actores y Cantantes con 11 y 15 famosos.

```
group_referrer_index = copy['label'] ==0
group_referrals = copy[group_referrer_index]

diversidadGrupo =  pd.DataFrame()
diversidadGrupo['categoria']=[0,1,2,3,4,5,6,7,8,9]
diversidadGrupo['cantidad']=group_referrals.groupby('categoria').size()
diversidadGrupo
```

De categoría 3 “modelos” hay 6 sobre un total de 9.

Buscaremos los usuarios que están más cerca a los centroids de cada grupo que podríamos decir que tienen los rasgos de personalidad característicos que representan a cada cluster:

```
1  closest,_= pairwise_distances_argmin_min( \
2          kmeans.cluster_centers_, X)
3  closest
4
5  users=dataframe['usuario'].values
6  for row in closest:
7  print(users[row])
```

En los centros vemos que tenemos una modelo, un político, presentadora de Tv, locutor de Radio y un deportista.