https://colab.research.google.com/drive/1OwBLVJmV-xdwwSdEIINgyioetB\_5jjVQ?usp=sharing

Reto: explora diferentes ejemplos de modelos y qué pueden hacer

Natasha Seelam, PhD

Head of AI/ML Research @ MindsDB

**Agradecimientos especiales: Dr. Dariusz K. Murakowski**

En esta notebook aprenderás la base del uso de diferentes algoritmos básicos de machine learning.

Para esta clase usaremos la librería [scikit-learn](https://scikit-learn.org/stable/), una de las librerías más sencillas de utilizar para aprender machine learning.

Para los siguientes ejemplos, usaremos los argumentos predeterminados. En la práctica, deberás ajustar los parámetros. Recuerda consultar todas las páginas de documentación de cada uno de los modelos:

* Regresión lineal: [aquí](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression" \l "sklearn.linear_model.LinearRegression" \t "_blank)
* Regresión logística: [aquí](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic%20regression" \l "sklearn.linear_model.LogisticRegression" \t "_blank)
* Clasificador de bosque aleatorio (árboles de decisión): [aquí](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=randomforest" \l "sklearn.ensemble.RandomForestClassifier" \t "_blank)

# Importamos librerías de manejo numérico y procesamiento de datos.  
import numpy as np  
import pandas as pd  
import sklearn  
from sklearn import datasets  
from sklearn import linear\_model  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.cluster import KMeans  
from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
  
# Definimos el tamaño del graficado (opcional).  
plt.rcParams['font.size']=15

Para este ejemplo usaremos el [Iris dataset](https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html). Contiene datos de dimensiones de partes de flores.

Iris dataset puede ser descargado automáticamente del módulo datasets de scikit-learn de la siguiente manera:

iris = datasets.load\_iris()  
print(iris)

{'data': array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

[5.4, 3.9, 1.7, 0.4],

[4.6, 3.4, 1.4, 0.3],

[5. , 3.4, 1.5, 0.2],

[4.4, 2.9, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.1],

[5.4, 3.7, 1.5, 0.2],

[4.8, 3.4, 1.6, 0.2],

[4.8, 3. , 1.4, 0.1],

[4.3, 3. , 1.1, 0.1],

[5.8, 4. , 1.2, 0.2],

[5.7, 4.4, 1.5, 0.4],

[5.4, 3.9, 1.3, 0.4],

[5.1, 3.5, 1.4, 0.3],

[5.7, 3.8, 1.7, 0.3],

[5.1, 3.8, 1.5, 0.3],

[5.4, 3.4, 1.7, 0.2],

[5.1, 3.7, 1.5, 0.4],

[4.6, 3.6, 1. , 0.2],

[5.1, 3.3, 1.7, 0.5],

[4.8, 3.4, 1.9, 0.2],

[5. , 3. , 1.6, 0.2],

[5. , 3.4, 1.6, 0.4],

[5.2, 3.5, 1.5, 0.2],

[5.2, 3.4, 1.4, 0.2],

[4.7, 3.2, 1.6, 0.2],

[4.8, 3.1, 1.6, 0.2],

[5.4, 3.4, 1.5, 0.4],

[5.2, 4.1, 1.5, 0.1],

[5.5, 4.2, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.2],

[5. , 3.2, 1.2, 0.2],

[5.5, 3.5, 1.3, 0.2],

[4.9, 3.6, 1.4, 0.1],

[4.4, 3. , 1.3, 0.2],

[5.1, 3.4, 1.5, 0.2],

[5. , 3.5, 1.3, 0.3],

[4.5, 2.3, 1.3, 0.3],

[4.4, 3.2, 1.3, 0.2],

[5. , 3.5, 1.6, 0.6],

[5.1, 3.8, 1.9, 0.4],

[4.8, 3. , 1.4, 0.3],

[5.1, 3.8, 1.6, 0.2],

[4.6, 3.2, 1.4, 0.2],

[5.3, 3.7, 1.5, 0.2],

[5. , 3.3, 1.4, 0.2],

[7. , 3.2, 4.7, 1.4],

[6.4, 3.2, 4.5, 1.5],

[6.9, 3.1, 4.9, 1.5],

[5.5, 2.3, 4. , 1.3],

[6.5, 2.8, 4.6, 1.5],

[5.7, 2.8, 4.5, 1.3],

[6.3, 3.3, 4.7, 1.6],

[4.9, 2.4, 3.3, 1. ],

[6.6, 2.9, 4.6, 1.3],

[5.2, 2.7, 3.9, 1.4],

[5. , 2. , 3.5, 1. ],

[5.9, 3. , 4.2, 1.5],

[6. , 2.2, 4. , 1. ],

[6.1, 2.9, 4.7, 1.4],

[5.6, 2.9, 3.6, 1.3],

[6.7, 3.1, 4.4, 1.4],

[5.6, 3. , 4.5, 1.5],

[5.8, 2.7, 4.1, 1. ],

[6.2, 2.2, 4.5, 1.5],

[5.6, 2.5, 3.9, 1.1],

[5.9, 3.2, 4.8, 1.8],

[6.1, 2.8, 4. , 1.3],

[6.3, 2.5, 4.9, 1.5],

[6.1, 2.8, 4.7, 1.2],

[6.4, 2.9, 4.3, 1.3],

[6.6, 3. , 4.4, 1.4],

[6.8, 2.8, 4.8, 1.4],

[6.7, 3. , 5. , 1.7],

[6. , 2.9, 4.5, 1.5],

[5.7, 2.6, 3.5, 1. ],

[5.5, 2.4, 3.8, 1.1],

[5.5, 2.4, 3.7, 1. ],

[5.8, 2.7, 3.9, 1.2],

[6. , 2.7, 5.1, 1.6],

[5.4, 3. , 4.5, 1.5],

[6. , 3.4, 4.5, 1.6],

[6.7, 3.1, 4.7, 1.5],

[6.3, 2.3, 4.4, 1.3],

[5.6, 3. , 4.1, 1.3],

[5.5, 2.5, 4. , 1.3],

[5.5, 2.6, 4.4, 1.2],

[6.1, 3. , 4.6, 1.4],

[5.8, 2.6, 4. , 1.2],

[5. , 2.3, 3.3, 1. ],

[5.6, 2.7, 4.2, 1.3],

[5.7, 3. , 4.2, 1.2],

[5.7, 2.9, 4.2, 1.3],

[6.2, 2.9, 4.3, 1.3],

[5.1, 2.5, 3. , 1.1],

[5.7, 2.8, 4.1, 1.3],

[6.3, 3.3, 6. , 2.5],

[5.8, 2.7, 5.1, 1.9],

[7.1, 3. , 5.9, 2.1],

[6.3, 2.9, 5.6, 1.8],

[6.5, 3. , 5.8, 2.2],

[7.6, 3. , 6.6, 2.1],

[4.9, 2.5, 4.5, 1.7],

[7.3, 2.9, 6.3, 1.8],

[6.7, 2.5, 5.8, 1.8],

[7.2, 3.6, 6.1, 2.5],

[6.5, 3.2, 5.1, 2. ],

[6.4, 2.7, 5.3, 1.9],

[6.8, 3. , 5.5, 2.1],

[5.7, 2.5, 5. , 2. ],

[5.8, 2.8, 5.1, 2.4],

[6.4, 3.2, 5.3, 2.3],

[6.5, 3. , 5.5, 1.8],

[7.7, 3.8, 6.7, 2.2],

[7.7, 2.6, 6.9, 2.3],

[6. , 2.2, 5. , 1.5],

[6.9, 3.2, 5.7, 2.3],

[5.6, 2.8, 4.9, 2. ],

[7.7, 2.8, 6.7, 2. ],

[6.3, 2.7, 4.9, 1.8],

[6.7, 3.3, 5.7, 2.1],

[7.2, 3.2, 6. , 1.8],

[6.2, 2.8, 4.8, 1.8],

[6.1, 3. , 4.9, 1.8],

[6.4, 2.8, 5.6, 2.1],

[7.2, 3. , 5.8, 1.6],

[7.4, 2.8, 6.1, 1.9],

[7.9, 3.8, 6.4, 2. ],

[6.4, 2.8, 5.6, 2.2],

[6.3, 2.8, 5.1, 1.5],

[6.1, 2.6, 5.6, 1.4],

[7.7, 3. , 6.1, 2.3],

[6.3, 3.4, 5.6, 2.4],

[6.4, 3.1, 5.5, 1.8],

[6. , 3. , 4.8, 1.8],

[6.9, 3.1, 5.4, 2.1],

[6.7, 3.1, 5.6, 2.4],

[6.9, 3.1, 5.1, 2.3],

[5.8, 2.7, 5.1, 1.9],

[6.8, 3.2, 5.9, 2.3],

[6.7, 3.3, 5.7, 2.5],

[6.7, 3. , 5.2, 2.3],

[6.3, 2.5, 5. , 1.9],

[6.5, 3. , 5.2, 2. ],

[6.2, 3.4, 5.4, 2.3],

[5.9, 3. , 5.1, 1.8]]), 'target': array([

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'target\_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '.. \_iris\_dataset:\n\nIris plants dataset\n--------------------\n\n\*\*Data Set Characteristics:\*\*\n\n :Number of Instances: 150 (50 in each of three classes)\n :Number of Attributes: 4 numeric, predictive attributes and the class\n :Attribute Information:\n - sepal length in cm\n - sepal width in cm\n - petal length in cm\n - petal width in cm\n - class:\n - Iris-Setosa\n - Iris-Versicolour\n - Iris-Virginica\n \n :Summary Statistics:\n\n ============== ==== ==== ======= ===== ====================\n Min Max Mean SD Class Correlation\n ============== ==== ==== ======= ===== ====================\n sepal length: 4.3 7.9 5.84 0.83 0.7826\n sepal width: 2.0 4.4 3.05 0.43 -0.4194\n petal length: 1.0 6.9 3.76 1.76 0.9490 (high!)\n petal width: 0.1 2.5 1.20 0.76 0.9565 (high!)\n ============== ==== ==== ======= ===== ====================\n\n :Missing Attribute Values: None\n :Class Distribution: 33.3% for each of 3 classes.\n :Creator: R.A. Fisher\n :Donor: Michael Marshall ([MARSHALL%PLU@io.arc.nasa.gov](mailto:MARSHALL%25PLU@io.arc.nasa.gov))\n :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature. Fisher\'s paper is a classic in the field and\nis referenced frequently to this day. (See Duda & Hart, for example.) The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant. One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n Mathematical Statistics" (John Wiley, NY, 1950).\n - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n Structure and Classification Rule for Recognition in Partially Exposed\n Environments". IEEE Transactions on Pattern Analysis and Machine\n Intelligence, Vol. PAMI-2, No. 1, 67-71.\n - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions\n on Information Theory, May 1972, 431-433.\n - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II\n conceptual clustering system finds 3 classes in the data.\n - Many, many more ...', 'feature\_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename': '/usr/local/lib/python3.7/dist-packages/sklearn/datasets/data/iris.csv'}

1: Familiarizarse con tu conjunto de datos

Están en la forma de array, típicamente denotado N×M donde N representa el número de filas (150), y M el número de columnas (4).

La clave feature\_names indica la información de las características: largo del sépalo, ancho del sépalo, largo del pétalo y ancho del pétalo.

1 TARGET (0,1) POR CADA FILA…. 150

El término "target", salida u objetivo es típicamente la meta que queremos modelar. La clave target\_names indica 3 especies de esta flor: "setosa", "versicolor" y "virginica".

The term "target" is typically the goal we are trying to model. The key "target\_names" indicates 3 species of this flower, "setosa", "versicolor", and "virginica".

Nuestra meta es usar estos features y descripciones de las flores para poder predecir la especie de la planta.

# Creamos el DataFrame con los feature names.  
data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)  
  
# Creamos el DataFrame con los targets (las especies de la flor).  
target = pd.DataFrame(data=iris.target, columns=['species'])  
  
# Unimos ambos DF con concat; agregamos una nueva columna.  
data = pd.concat([data, target], axis=1)  
  
# Mezclar en orden aleatorio.  
data = data.sample(frac=1, random\_state=1234)  
  
# Imprimimos los primeros registros del nuevo DataFrame.  
data.head()

1a: Separando nuestros datos - training y testing

Antes de comenzar a crear modelos, es importante dividir nuestro conjunto de datos en **Entrenamiento (Training)** y **Pruebas (Testing)**.

Una buena regla general es que la mayoría de los datos deben incluirse en **entrenamiento**. Los datos de entrenamiento se utilizan para ayudar a nuestros modelos a aprender las "reglas" o patrones subyacentes dentro de los datos.

El conjunto de **pruebas** debe separarse del conjunto de entrenamiento. Su objetivo es evaluar el modelo que hizo, lo que significa que verificamos si los patrones detectados en el modelo reflejan lo que estamos tratando de modelar.

También recordemos que existe el término **Validación (Validation)**. Esta separación del dataset se usa típicamente para modelos más complicados donde se requieren parámetros de ajuste (o los elementos clave) de un modelo. En el siguiente caso estos modelos son lo suficientemente simples como para que datos de validación no sean necesaria.

Una división del 80/20 por ciento de entrenamiento/pruebas suele ser razonable de manera elemental. Realizamos esto:

# Fracción de entrenamiento 0.8.  
Ntrain = int(data.shape[0] \* 0.8)  
  
train = data.iloc[:Ntrain, :]  
test = data.iloc[Ntrain:, :]

2: Regresión lineal

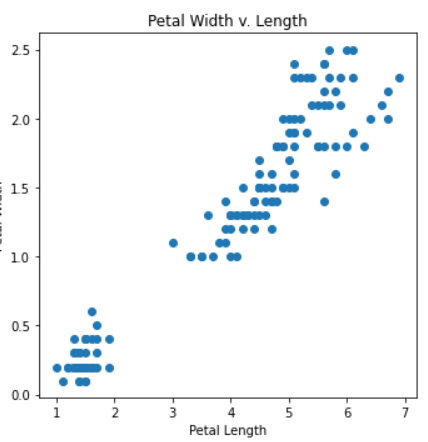
La regresión lineal intenta predecir una salida de valor numérico. La especie de la flor es una etiqueta o label. Así que intentemos pensar en un modelo en el que podamos predecir la salida de un número.

Para este ejemplo, supongamos que queremos predecir **petal width** o ancho del pétalo (Index = 3 en el DataFrame).

Para comenzar, construyamos un modelo con solo 1 feature, el **petal length** o largo del pétalo (Index = 2 en el DataFrame).

Primero, observemos la relación entre **petal length** y **petal width** en los datos:

# Definimos variables para acceder de forma más simple a las columnas de nuestros features.  
plength = data["petal length (cm)"]  
pwidth = data["petal width (cm)"]  
  
# Creamos la gráfica comparanco Petal lenght (eje x) contra Petal width (eje y)  
f = plt.figure(figsize=(5,5))  
ax = f.add\_subplot(1,1,1)  
ax.scatter(plength, pwidth)  
ax.set\_xlabel("Petal Length")  
ax.set\_ylabel("Petal Width")  
ax.set\_title("Petal Width v. Length")  
f.tight\_layout()



De lo anterior, podemos ver una relación **lineal**. Queremos cuantificar esta relación.

En la gráfica anterior x es el Petal Lenght y y es el Petal Width. Queremos ajustar un modelo a la forma θ1∗x+θ0=y, la ecuación de la recta para generar nuestra regresión lineal.

Podemos utilizar scikit-earn para construir un modelo de regresión lineal como tenemos a continuación:

**NOTA:** aparecerá un molesto mensaje de error ya que scikit-learn espera un input 2D (N x 1) por eso llamamos "DataFrame" en el x-input para ajustar el comando.

# Importar un objeto de regresión lineal de sklearn.  
model\_1 = linear\_model.LinearRegression()  
  
# Ajustar el modelo a tus datos.  
model\_1.fit(pd.DataFrame(train.iloc[:, 2]), train.iloc[:, 3])  
  
# Imprimir los coeficientes.  
print("Coef\n", model\_1.coef\_)  
  
# Imprimir el sesgo o bias.  
print("\n\nBias\n", model\_1.intercept\_)

Coef

[0.41490802]

Bias

-0.357589314248546

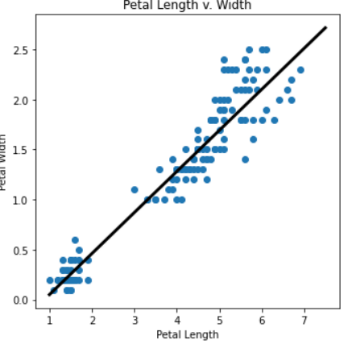
El resultado anterior nos dice que θ1, la relación con el largo del pétalo, es 0.41, y que el bias (también llamado interceptar) es −0.36

The above tells us that θ1, the relationship with petal length, is 0.41, and that the bias (también llamado interceptar) is −0.36. En conjunto, esperamos que la fórmula se vea así:

θ1∗x+θ0=0.41∗x−0.36=y

Grafiquemos esto para observarlo:

# Definimos variables para acceder de forma más simple a las columnas de nuestros features.  
plength = data["petal length (cm)"]  
pwidth = data["petal width (cm)"]  
  
# Definimos un arreglo desde el valor mímimo del petal length hasta el máximo del petal lenght + 1, en pasos de 0.5.  
xvals = np.arange(plength.min(), plength.max()+1, 0.5)  
  
# Ecuación de nuestra regresión lineal.  
yvals = 0.41\*xvals - 0.36  
  
# Gráficamos nuestra recta con los datos y nuestra ecuación.  
f = plt.figure(figsize=(5,5))  
ax = f.add\_subplot(1,1,1)  
ax.scatter(plength, pwidth)  
ax.plot(xvals, yvals, 'k', linewidth=3)  
ax.set\_xlabel("Petal Length")  
ax.set\_ylabel("Petal Width")  
ax.set\_title("Petal Length v. Width")  
f.tight\_layout()



¡No es una mala relación! Podemos dar un paso más para cuantificar "qué tan bueno es el ajuste" utilizando dos métricas de rendimiento: el error cuadrático medio (MSE) y R2.

El valor de R2 busca la correlación entre x y y. Para este caso pregunta "qué tan bien se ajusta la línea a los puntos de nuestros datos".

Siempre queremos un MSE bajo y R2 cercano a 1. R2 está limitado de 0 (efectivamente no correlacionado) a 1 (relación perfecta).

Podemos calcular nuestras métricas de rendimiento prediciendo primero en nuestro **conjunto de datos de pruebas** y luego viendo qué tan buena es la estimación de nuestro modelo del ancho del pétalo cuando se usa la longitud del pétalo, en comparación con los valores reales del ancho del pétalo.

[ ]

# Predicción con model\_1 = linear\_model.LinearRegression() utilizando los datos de prueba.  
ypredict = model\_1.predict(pd.DataFrame(test.iloc[:, 2]))  
  
# Calculamos el MSE con los valores predichos en y.  
print('MSE: %.2f'  
      % mean\_squared\_error(pd.DataFrame(test.iloc[:, 3]), ypredict))  
  
# Calculamos R^2 con los valores predichos en y.  
print('R2: %.2f'  
      % r2\_score(pd.DataFrame(test.iloc[:, 3]), ypredict))

MSE: 0.04

R2: 0.94

¡Excelente! Podemos ver que existe una relación decente entre la longitud y el ancho de los pétalos.

3: Regresión logística

Similar a la regresión lineal, la regresión logística ajusta una "línea" con los coeficientes. Sin embargo, a diferencia de la regresión lineal, este modelo tiene como objetivo **clasificar datos**.

A partir del ancho de los pétalos, podemos ver que hay 3 comportamientos distintos de cada una de las especies de flores. Usando el largo/ancho del sépalo y el largo/ancho del pétalo, nos gustaría clasificar a qué especie de flor pertenece cada punto de datos.

Para hacer esto podemos emplear regresión logística. En la regresión logística binaria, estamos tratando de ajustar un logaritmo de relación impar; es decir, una probabilidad de una clase particular en relación con la otra clase.

En este problema hay 3 especies, por lo que la fórmula está ligeramente adaptada. Afortunadamente, scikit-learn puede manejar problemas de etiquetas binarias o de etiquetas múltiples con mucha facilidad.

Para construir un modelo de regresión logística se puede utilizar lo siguiente:

[ ]

# Preparamos los datos de entrenamiento y prueba en x y y.  
# La última columna es la etiqueta de la especie..  
xtrain = train.iloc[:, :-1] # input, training  
ytrain = train.iloc[:, -1] # output, training  
xtest = test.iloc[:, :-1] # input, test  
ytest = test.iloc[:, -1] # output, test  
  
# Configuramos la regresión logística con el método de regresión logística de linear\_model.  
model\_2 = linear\_model.LogisticRegression()  
  
# Ajustamos el modelo con los datos de entrenamiento.  
model\_2.fit(xtrain, ytrain)

/usr/local/lib/python3.7/dist-packages/sklearn/linear\_model/\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

<https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression>

extra\_warning\_msg=\_LOGISTIC\_SOLVER\_CONVERGENCE\_MSG)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

La evaluación de la regresión logística utilizará **precisión (accuracy)**, o un recuento de la cantidad de veces que el modelo etiqueta correctamente un ejemplo.

Lo podemos hacer de la siguiente manera:

# Hacer predicciones con el conjunto de prueba.  
ypred = model\_2.predict(xtest)  
  
# Probamos el accuracy del modelo generado  
print("Testing accuracy =", accuracy\_score(ytest, ypred))

Testing accuracy = 1.0

¡Increíblemente, nuestro modelo pudo obtener todos los ejemplos correctamente!

Tal hazaña solo puede ocurrir si el conjunto de prueba de los datos es *separable linealmente*, una condición especial en la que es realmente fácil agrupar los datos, o una línea (o plano) dibujada puede separar completamente entre todas las muestras de diferentes clases.

En problemas más complicados, es posible que necesites verificar la *[matriz de confusión](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html" \t "_blank)* preguntando cuáles son los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Puedes hacer esto usando la matriz de confusión de scikit-learn de la siguiente forma:

[ ]

# Un ejemplo de la matriz de confusión con sklearn  
#Importamos el método de matriz de confusión desde sklearn  
from sklearn.metrics import confusion\_matrix  
  
#Aplicamos la matriz a nuestros valores de testing y los valores de predicción generados.  
confusion\_matrix(ytest, ypred)

array([[11, 0, 0],

[ 0, 7, 0],

[ 0, 0, 12]])

Ten en cuenta que, dado que hay 3 clases, cada columna y fila individual corresponde a una clase. Las columnas corresponden a las etiquetas predichas y las filas a las etiquetas verdaderas.

3a: Regresión logística - regularización (bonus)

Vale la pena mirar los [argumentos predeterminados](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html" \t "_blank) de LogisticRegression de scikit-learn. A saber, una cosa a buscar son los términos de **regularización** y **penalización**.

La **regularización**, aunque no se trata en el curso, tiene la intención de ayudar a los modelos asegurándose de que no se ajusten demasiado. Al agregar un nuevo término a la función de pérdida, el modelo penalizará el tamaño de cada coeficiente en el modelo.

Overfitting means that your model will perform fantastically on the training set, but horribly on the test (or validation) set. Regularization tries to make coefficients of redundant/not useful features as close to 0 as possible, meaning they play little role in the prediction.

El **sobreajuste** u **overfitting** significa que el modelo funcionará fantásticamente en el conjunto de entrenamiento, pero horriblemente en el conjunto de prueba (o el de validación). La regularización intenta hacer que los coeficientes de las características redundantes/no útiles sean lo más cercanos a 0 como sea posible, lo que significa que juegan un papel pequeño en la predicción.

En scikit-learn, penalty o penalización se refiere al tipo de regularización, normalmente **L1** or **L2**. Ambos métodos fomentan la escasez (lo que significa que muchos de los coeficientes delante de las características se acercan a 0). Además, la fuerza de la penalización inversa (regularización) es C. Una C grande indica una menor fuerza de penalización, así que disminuye gradualmente este valor mientras intentas encontrar los features más importantes.

4. Bosque aleatorio (árboles de decisión)

El último enfoque que analizaremos es el bosque aleatorio. Como estamos tratando de etiquetar entre especies, usaremos un Clasificador de bosque aleatorio como se ve [aquí](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=randomforest" \l "sklearn.ensemble.RandomForestClassifier" \t "_blank). También hay un Random Forest [regressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html?highlight=randomforest#sklearn.ensemble.RandomForestRegressor) o regresor de bosque aleatorio.

Bosque aleatorio tiene varios argumentos de entrada. Por simplicidad solo agregamos los estimadores n\_estimators. Corresponden al número de árboles en el bosque.

# Creamos el clasificador  
clf = RandomForestClassifier(n\_estimators=3)  
  
# Entrenamos el modelo  
clf.fit(xtrain, ytrain)  
  
# Generamos predicción   
y\_pred=clf.predict(xtest)  
  
# Probamos el modelo con accuracy  
print("Testing accuracy =", accuracy\_score(ytest, ypred))

Testing accuracy = 1.0

El modelo es capaz de conseguir perfección en la prueba de accuracy (100% o 1).

5. K-Means clustering

En el último enfoque, intentaremos ver si podemos ver patrones dentro de los datos usando KMeans. KMeans es un algoritmo de aprendizaje no supervisado, lo que significa que no se requieren etiquetas para entrenar.

Aquí, ya sabemos que hay 3 especies diferentes de flores, y hemos tenido muchos modelos de entrenamiento exitosos para reconocer estas especies. Sin embargo, en muchos problemas, es posible que no sepamos de antemano qué grupos pueden existir ya en los datos.

Para ello, KMeans sobresale. A continuación, usaremos un modelo de K-medias e intentaremos encontrar cuántos grupos hay.

Primero, inicializamos nuestro modelo KMeans. Existen muchos argumentos en la implementación de scikit-learn [aquí](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html" \t "_blank). Mientras que usaremos los por defecto, el principal a tomar en cuenta es n\_clusters o cuántos grupos pensamos de antemano que hay en los datos. También usaremos random\_state, pero eso es para asegurar la reproducibilidad.

Ten en cuenta que usaremos el conjunto de datos completo aquí, pero es posible encajar en un conjunto de entrenamiento y usar un conjunto de pruebas por separado para ver qué tan "robustos" son los clústeres.

Para simplificar la visualización, entrenaremos con petal length and petal width.

# Inicializamos el modelo KMeans.  
k2model = KMeans(n\_clusters=2, random\_state=42)  
  
# Ajustar todos los puntos de datos EXCEPTO para la especie.  
k2model.fit(data[["petal length (cm)", "petal width (cm)"]])  
  
# Calculamos e imprimimos la inercia desde el modelo de Kmeans.  
print("Inertia =", k2model.inertia\_)  
  
# Cálculamos e imprimimos la posición de los clústers.  
print("Centroids (x, y) =\n", k2model.cluster\_centers\_)

Inertia = 86.39021984551395

Centroids (x, y) =

[[4.92525253 1.68181818]

[1.49215686 0.2627451 ]]

Primero, visualicemos estos centros de clústeres en un diagrama de dispersión de los puntos de datos.

Antes de trazar los centros de los clústers primero trazaremos los puntos de datos.Para mayor claridad, también los colorearé de acuerdo con las etiquetas. ***NOTA, los modelos entrenaron SIN conocer estas etiquetas***.

# Graficamos los datos con la línea para setosa.  
f = plt.figure(figsize=(5,5))  
ax = f.add\_subplot(1,1,1)  
  
# Setosa  
ax.scatter(data[data.iloc[:, -1]==0]["petal length (cm)"],  
           data[data.iloc[:, -1]==0]["petal width (cm)"],  
           c='k')  
  
# Versicolor  
ax.scatter(data[data.iloc[:, -1]==1]["petal length (cm)"],  
           data[data.iloc[:, -1]==1]["petal width (cm)"],  
           c='r')  
  
# Virginica  
ax.scatter(data[data.iloc[:, -1]==2]["petal length (cm)"],  
           data[data.iloc[:, -1]==2]["petal width (cm)"],  
           c='b')  
  
ax.legend(["Setosa", "Versicolor", "Virginica"])  
  
# Graficamos los centroides de los clústers (output en Petal Length x Petal Width)  
ax.plot(k2model.cluster\_centers\_[:,0],  
        k2model.cluster\_centers\_[:,1],  
        "g\*", markersize=30)  
  
ax.set\_xlabel("Petal Length")  
ax.set\_ylabel("Petal Width")  
ax.set\_title("Petal Length v. Width")  
f.tight\_layout()

Nuestro modelo pareció captar el clúster inferior, pero debido a que solo especificamos 2 clústers, parece que perdemos la diferencia entre Versicolor y Virginica. El modelo coloca apropiadamente un centroide entre ellos.

En la práctica, es posible que no tengas las etiquetas disponibles como las tenemos aquí. Entonces, para poder identificar cómo elegir el mejor "*k*", usaremos un **diagrama de codo** tratando de ajustar un modelo para muchos *k*s diferentes en un panel pequeño. Trazamos un diagrama de codo típicamente con inercia en el eje y, y el número de grupos en el eje x. La inercia puede ser recuperada por el modelo como se ve en las 2 celdas de código arriba; mide qué tan lejos están los puntos de datos de sus centros de conglomerados más cercanos. Idealmente, los modelos con menor inercia son mejores; sin embargo, con más grupos, la inercia a menudo seguirá disminuyendo marginalmente. Por lo tanto, el diagrama de codo nos dice el mejor número de conglomerados que debemos tener para aprovechar al máximo la inercia.

6: Conclusiones

En esta notebook hemos cubierto lo esencial de diferentes modelos de machine learning usando scikit-learn:

* Regresión lineal - (supervisado, regresión).
* Regresión logística - (supervisado, clasificación).
* Bosques aleatorios (Random Forest) - (supervised, clasificación).
* K-Means - (no supervisado, clustering)

En los datos con los que puedes trabajar en el futuro es posible que las respuestas no sean tan claras. Es importante probar muchos modelos y sondear realmente tu conjunto de datos para preguntar si las métricas de rendimiento realmente indican si los modelos están aprendiendo como se esperaría.