UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

UNIDAD DIDÁCTICA IV

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo I – Nivel Inicial I

Unidad IV – Scikit-learn.

Presentación:

En el aprendizaje de máquinas contamos con varias herramientas, una de las más utilizadas es la librería scikit-learn, escrita principalmente en python, la cual cuenta con varios algoritmos de clasificación, regresión y análisis de grupos. Esta librería se integra muy bien con la librería NumPy y Matplotlib que ya hemos utilizado y es de código abierto, es de fácil implementación y puede ser utilizada en diferentes contextos.

En esta y las próximas unidades veremos cómo utilizar scikit-learn para resolver diferentes problemáticas de estudio.

Objetivos:

Que los participantes:

Comiencen a analizar casos básicos de aprendizaje supervisado.

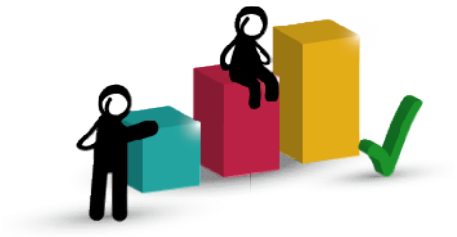
Aprendan los pasos a seguir para la generación de modelos simples.

Bloques temáticos:

1.- Aprendizaje supervisado.

2.- Regresión Lineal.

3.- Clasificación – Regresión logística.



Consignas para el aprendizaje colaborativo

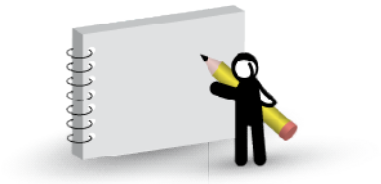
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Aprendizaje supervisado

Cuando trabajamos dentro de lo que es el aprendizaje supervisado, contamos con datos que poseen atributos adicionales, los cuales queremos predecir. Básicamente podemos contar con dos tipos de casos, que podemos nombrar como clasificación y regresión:

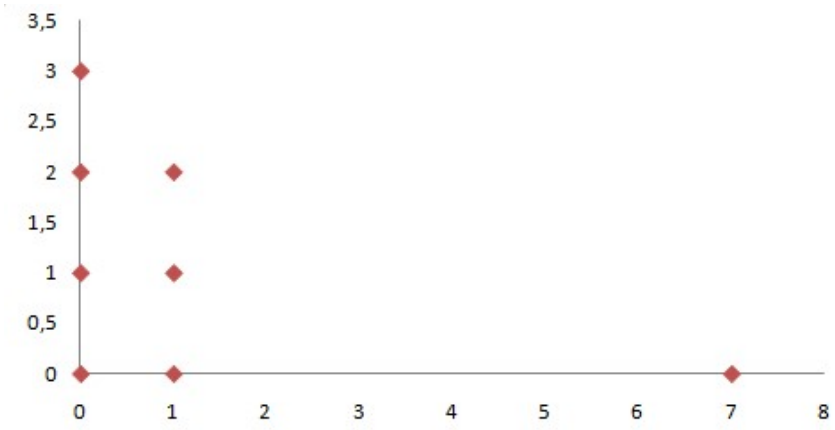
- **Regresión**: si la salida deseada consiste en una o más variables continuas, la tarea se llama regresión. Un ejemplo de un problema de regresión sería la predicción de la longitud de un salmón en función de su edad y peso.
- **Clasificación**: En este caso las muestras pertenecen a dos o más clases y queremos aprender de los datos ya etiquetados cómo predecir la clase de datos sin etiquetar. Un ejemplo de un problema de clasificación sería el reconocimiento de dígitos a mano, en el que el objetivo es asignar cada vector de entrada a una de un número finito de categorías discretas. Otra forma de pensar en la clasificación es como una forma discreta (en lugar de continua) de aprendizaje supervisado donde uno tiene un número limitado de categorías y para cada una de las n muestras proporcionadas, se trata de etiquetarlas con la categoría o clase correcta.

Uno de los puntos más difíciles en la creación de algoritmos es obtener datos que podamos utilizar confiadamente. El preprocesamiento de datos es un punto fundamental que debemos tomar muy en serio. Antes de utilizar los datos dentro de un algoritmo, debemos verificar la calidad y precisión de los mismos. En esta unidad comenzaremos a utilizar el análisis exploratorio de datos AED, el cual de caracteriza por su sencillez, debida principalmente a:

1. El empleo de gráficas y la reducción del uso de fórmulas algebraicas.
2. A que requiere muy pocas consideraciones previas sobre los datos.

El AED enfatiza la importancia de organizar y desplegar los datos gráficamente, de modo tal que se puedan percibir valores distintivos, como valores extremos, atípicos, conglomerados (grupos de datos muy cercanos entre sí) y brechas (separaciones de magnitudes considerables entre conglomerados), así como la identificación de patrones globales.

El AED nos lleva a plantearnos preguntas sobre los datos, por ejemplo tomemos la siguiente recopilación de datos a partir de números enteros:



Independientemente de lo que representan los datos, se puede apreciar en el diagrama que hay un valor considerablemente mayor que todos (7) que puede verse como atípico y que se tienen dos conglomerados, un grupo de cuatro valores en cero y un grupo de tres valores en uno.

2. Regresión lineal

El método de regresión lineal consiste en identificar con precisión una línea que sea capaz de representar la distribución de puntos en un plano bidimensional. Dada la siguiente ecuación de la recta:

$$y = \frac{y_2 - y_1}{x_2 - x_1} * (x - x_1) + y_1 = \frac{y_2 - y_1}{x_2 - x_1} * x + \left(y_1 - \frac{y_2 - y_1}{x_2 - x_1} * x_1\right)$$

$$y = \alpha * x + \beta$$

El coeficiente alfa nos da la pendiente de la recta, mientras que el coeficiente beta nos da el punto de corte con el eje **y**.

Vamos a ver cómo realizar una regresión lineal a partir de los datos almacenados en el archivo datos.csv, importando los datos mediante **pandas** y convirtiendo las columnas en dos arrays de **numpy**.

```
import pandas as pd
import numpy as np
dat_csv  = pd.read_csv('datos.csv', encoding = "ISO-8859-1")
datos_x = dat_csv.x
datos_y = dat_csv.y
x = []
y = []
for i in dat_csv.x:
    x.append(i)
for j in dat_csv.y:
    y.append(j)
print(x)
print(y)

X = np.array(x)
Y = np.array(y)
```

Creamos datos de entrenamiento

Cuando construimos un modelo de aprendizaje automático, necesitamos una manera de validar nuestro modelo y verificar si está funcionando a un nivel satisfactorio. Para hacer esto, debemos separar nuestros datos en dos grupos: un conjunto de datos de entrenamiento y un conjunto de datos de prueba. El conjunto de datos de entrenamiento se usará para construir el modelo, y el conjunto de datos de prueba se usará para ver cómo este modelo entrenado se desempeña en datos desconocidos. Entonces, sigamos adelante y dividamos estos datos en conjuntos de datos de entrenamiento y prueba:

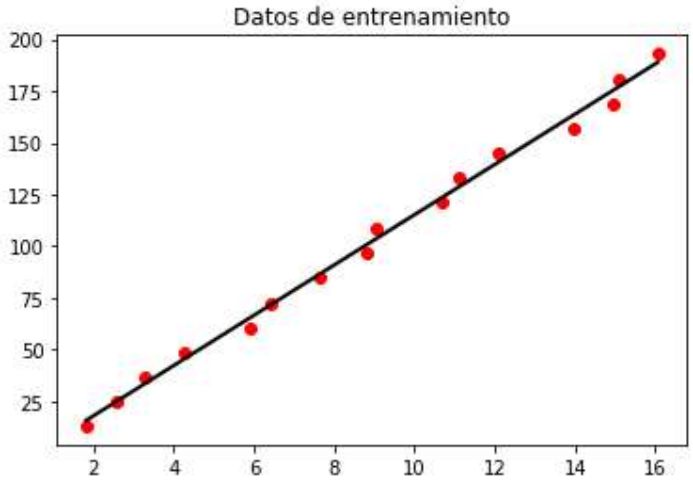
```
X,Y
datos_entrenamiento = int(0.8 * len(X))
datos_prueba = len(X) - datos_entrenamiento

#ENTRENAMIENTO
X_entrenamiento = X[:datos_entrenamiento].reshape((datos_entrenamiento,1))
print(X_entrenamiento)
Y_entrenamiento = Y[:datos_entrenamiento].reshape((datos_entrenamiento,1))
print(Y_entrenamiento)
#PRUEBA
X_prueba = X[datos_entrenamiento:].reshape((datos_prueba,1))
Y_prueba = Y[datos_entrenamiento:].reshape((datos_prueba,1))
```

Realizamos regresión lineal.

Para realizar la regresión lineal, haremos uso del objeto **linear_model.Regression()** de scikit-learn, el cual hace uso del método fit() para tomar los dos datos a ajustar y obtener la regresión, luego a partir de los datos de regresión encontrados podemos observar la curva de ajuste de entrenamiento junto con la dispersión de datos de entrenamiento.

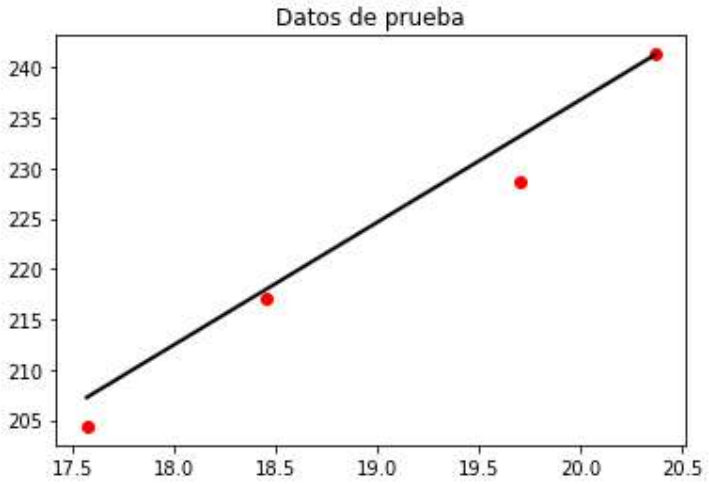
```
from sklearn import linear_model

# Creamos un objeto de regresión lineal
linear_regressor = linear_model.LinearRegression()

# Ajustamos la regresión a los datos de entrenamiento
linear_regressor.fit(X_entrenamiento, Y_entrenamiento)

# Y a partir de datos de regresión encontrados
Y_predicha_de_entrenamiento = linear_regressor.predict(X_entrenamiento)
```

```
import matplotlib.pyplot as plt

plt.figure()
plt.scatter(X_entrenamiento, Y_entrenamiento, color='red')
plt.plot(X_entrenamiento, Y_predicha_de_entrenamiento, color='black', linewidth=2)
plt.title('Datos de entrenamiento')
plt.show
```



Una vez que visualizamos el ajuste podemos evaluar la correspondencia de la regresión con relación a los datos de prueba.

```
Y_predicha_de_prueba = linear_regressor.predict(X_prueba)
plt.figure()
plt.scatter(X_prueba, Y_prueba, color='red')
plt.plot(X_prueba, Y_predicha_de_prueba, color='black', linewidth=2)
plt.title('Datos de prueba')
plt.show
```



Existen varias formas de evaluar una regresión lineal, y podemos utilizar nuevamente **scikit-learn** para realizar esta tarea, mediante el módulo **metrics** según se muestra a continuación:

```
import sklearn.metrics as sm
print("Error absoluto medio =", round(sm.mean_absolute_error(Y_prueba,
Y_predicha_de_prueba), 2))
print("Error cuadratico medio =", round(sm.mean_squared_error(Y_prueba,
Y_predicha_de_prueba), 2))
print("Error absoluto mediano =", round(sm.median_absolute_error(Y_prueba,
Y_predicha_de_prueba), 2))
print("Puntuación de varianza explicada =",
round(sm.explained_variance_score(Y_prueba, Y_predicha_de_prueba), 2))
print("Puntuación R2 =", round(sm.r2_score(Y_prueba, Y_predicha_de_prueba), 2))
```

- **Error absoluto medio:** este es el promedio de errores absolutos de todos los puntos de datos en el conjunto de datos dado.
- **Error cuadrático medio**: Este es el promedio de los cuadrados de los errores de todos los puntos de datos en el conjunto de datos dado. ¡Es una de las métricas más populares que hay!
- **Error absoluto medio**: Esta es la mediana de todos los errores en el conjunto de datos dado. La principal ventaja de esta métrica es que es robusta a los valores atípicos. Un único punto negativo en el conjunto de datos de prueba no

distorsionara la métrica de error completa, en lugar de una métrica de error promedio.

- **Puntuación de varianza explicada**: esta puntuación mide qué tan bien nuestro modelo puede explicar la variación en nuestro conjunto de datos. Una puntuación de 1.0 indica que nuestro modelo es perfecto.
- **Puntuación R2**: Se nombra como R cuadrado, y esta puntuación se refiere al coeficiente de determinación. Esto nos dice qué tan bien serán predichas las muestras desconocidas por nuestro modelo. La mejor puntuación posible es 1.0, pero la puntuación también puede ser negativa.

Los valores obtenidos son:

- Error absoluto medio = 2.02
- Error cuadrático medio = 7.01
- Error absoluto medio = 1.83
- Puntuación de varianza explicada = 0.98
- Puntuación R2 = 0.96

Regresión contraída

En ocasiones cuando vamos a realizar una regresión lineal, nos encontramos que pueden existir algunos valores aislados, quizás producto de una mala toma de datos que se alejan de la dispersión y modifican el valor de los coeficiente encontrados. Cuando realizamos la regresión lineal por el método de mínimos cuadrados, los coeficientes se estiman determinando valores numéricos que minimizan la suma de las desviaciones al cuadrado entre las respuestas observadas y las respuestas ajustadas, de acuerdo con la siguiente ecuación:

$$RSS = \sum_{i=1}^{n} (y_i - \beta_1 * x_i + \beta_2)^2$$

La regresión de cresta, para estimar los coeficientes β, comienza a partir de la fórmula básica de la suma residual de cuadrados (RSS) y agrega el término de penalización. λ (≥ 0) el cual se define como el parámetro de ajuste, que se multiplica por la suma de los coeficientes de β al cuadrado (excluyendo la intersección) para definir el período de penalización, como se muestra en la siguiente ecuación:

$$RSS + \lambda * \beta_1^2 = \sum_{i=1}^{n} (y_i - \beta_1 * x_i + \beta_2)^2 + \lambda * \beta_1^2$$

```
from sklearn import linear_model
regresion_ridge = linear_model.Ridge(alpha=0.01, fit_intercept=True,
max_iter=10000)
regresion_ridge.fit(X_prueba, Y_prueba)

Y_predicha_de_prueba = regresion_ridge.predict(X_prueba)


print("Error absoluto medio =", round(sm.mean_absolute_error(Y_prueba,
Y_predicha_de_prueba), 2))
print("Error cuadratico medio =", round(sm.mean_squared_error(Y_prueba,
Y_predicha_de_prueba), 2))
print("Error absoluto mediano =", round(sm.median_absolute_error(Y_prueba,
Y_predicha_de_prueba), 2))
print("Puntuación de varianza explicada =",
round(sm.explained_variance_score(Y_prueba, Y_predicha_de_prueba), 2))
print("Puntuación R2 =", round(sm.r2_score(Y_prueba, Y_predicha_de_prueba), 2))
```

En este caso los valores obtenidos son:

- Error absoluto medio = 1.49
- Error cuadrático medio = 2.86
- Error absoluto mediano = 1.49
- Puntuación de varianza explicada = 0.98
- Puntuación R2 = 0.98

3. Clasificación - Regresión logística.

La regresión logística (Logistic Regression), a pesar de su nombre, es un modelo lineal para la clasificación en lugar de la regresión. La regresión logística también se conoce en

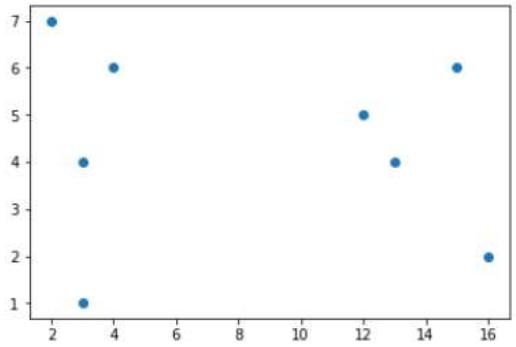
la literatura como regresión logit, clasificación de máxima entropía (MaxEnt) o el clasificador log-lineal. En este modelo, las probabilidades que describen los posibles resultados de un solo ensayo se modelan utilizando una función logística.

Veamos cómo aplicar la regresión logística aplicando el proceso a la clasificación de una serie de puntos, es un caso abstracto para generalizar el procedimiento, pero podríamos aplicarlo a algo más concreto como la clasificación de mails para determinar si es spam o no.

Paso 1

Comencemos por tomar una serie de puntos según se muestra a continuación y graficarlos como hemos aprendido a hacer hasta aquí.

```
import numpy as np

from sklearn import linear_model

from sklearn import model_selection

from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt


X = np.array([[16,2], [3,1], [2,7], [13,4], [3,4], [12,5], [15,6], [4,6]])

plt.figure()

plt.scatter(X[:,0],X[:,1])

plt.show()
```



Paso 2

Dados los puntos de dispersión parecen existir dos grupos de datos, uno que se agrupa en torno al valor x = 3 y otro que se agrupa en torno al valor x = 14, lo que haremos es en base a esta hipótesis asignar a los valores cercanos a 3 el valor de cero y a los cercanos a catorce el valor de uno y agregaremos los valores en el mismo orden dentro del array Y.

```
Y = [1, 0, 0, 1, 0, 1, 1, 0]
```

En base a esta clasificación separaremos los datos en dos clases

```
clase0 = np.array([X[i] for i in range(len(X)) if Y[i]==0])

clase1 = np.array([X[i] for i in range(len(X)) if Y[i]==1])
```

Paso 3

Ahora utilizaremos el objeto clasificador “LogisticRegresion” y le pasaremos los datos que deseamos clasificar de forma de crear un modelo, haciendo que se ajuste mediante el uso de (fit)

```
clasificador = linear_model.LogisticRegression(solver='lbfgs', C=100)

clasificador.fit(X, Y)
```

Paso 4

Con la clasificación realizada, utilizaremos el método predict() de forma de predecir como el modelo predice los propios valores pasados como datos. Este es un caso muy simple en el cual la predicción retorna exactamente los mismos valores, sin embargo en otros casos más complejos el ajuste no es del 100% y el modelo debe ser ajustado a medida que conocemos más la problemática del tema tratado.

```
prediccion = clasificador.predict(X)

print(prediccion)
```

Retorna: [1 0 0 1 0 1 1 0]

```
print(clasificador.score(X,Y))
```

Retorna: 1.0

Paso 5

Ahora en base a otros datos recopilados podemos ver como el modelo predice al grupo al cual deberían pertenecer los datos.

```
Xn = np.array([[6,4], [20,7], [4,17]])

Yn = clasificador.predict(Xn)

print(Yn)
```

Nos retorna: [0 1 0]

Paso 6

Siempre es mejor tratar de visualizar los datos para tener una mejor comprensión del tema, por lo que vamos a graficar los datos asignando como colores las opciones de ceros o unos de la clasificación (c=Y).

Tomaremos los datos de que utilizamos para crear el modelo y los que hemos utilizados para testear el modelo y los concatenamos para formar un único array en x e y, y luego determinaremos el rango de visualización en el cual veremos los datos.

```
Xs = np.concatenate((X, Xn), axis=0)

Ys = np.append(Y , Yn)



x_min, x_max = min(Xs[:, 0]) - 1.0, max(Xs[:, 0]) + 1.0

y_min, y_max = min(Xs[:, 1]) - 1.0, max(Xs[:, 1]) + 1.0
```

Dado ahora los puntos de los ejes x e y crearemos a partir de ellos una malla de puntos lo cual es en la practica una cuadricula rectangular creada a partir de los datos.

```
 step_size = 0.01

valor_x, valor_y = np.meshgrid(np.arange(x_min, x_max, step_size), np.arange(y_min, y_max, step_size))

malla_de_puntos = clasificador.predict(np.c_[valor_x.ravel(), valor_y.ravel()])
```

Aquí estamos utilizando el método ravel() de forma de crear una copia de los valores originales. También podríamos haber utilizado el método flatten(), la diferencia entre ambos son:

- **ravel ():** Retorna una referencia a la matriz original, si se modifica la matriz, el valor de la matriz original también cambia. **ravel()** es más rápido que **flatten()**.

- **flatten (): Retorna una** copia de la matriz original, si sei modifica algún valor de esta matriz, el valor de la matriz original no se verá afectado.

También notar que estamos utilizando np.c_, para obtener los datos por pares x, y ya que dada una matriz nos retorna sus valores por columnas.

Paso 7

Creamos un mapa de color a partir de los datos anteriores mediante **pcolormesh()** :

```
malla_de_puntos = malla_de_puntos.reshape(valor_x.shape)

plt.figure()

plt.pcolormesh(valor_x, valor_y, malla_de_puntos, cmap=plt.cm.gray)
```

Para comprender cómo funciona reshape() consideremos que tenemos el siguiente ejemplo en el cual tenemos dos arrays

```
x = np.array([1, 5, 10])

y = np.array([1, 4, 8])

z = np.linspace(0, 255, 9).reshape(3, 3)

print(z)

plt.pcolormesh(x, y, z, cmap = "PuRd");

plt.colorbar();
```

Pcolormesh() crea una gráfica en pseudo-colores con un grid no rectangular.

pcolormesh([x, y,] c, **kwargs)

Aquí:

- c es un array 2D de escalares, valores que serán "mapeados" a un conjunto de colores determinado por el argumento cmap,
- x e y, son argumentos opcionales, definen las coordenadas de las esquinas de los rectángulos que se mostrarán del color correspondiente según se muestra a continuación.

```
(X[i+1, j], Y[i+1, j])         (X[i+1, j+1], Y[i+1, j+1])
                       +--------+
                       | C[i,j] |
                       +--------+
   (X[i, j], Y[i, j])          (X[i, j+1], Y[i, j+1]),
```

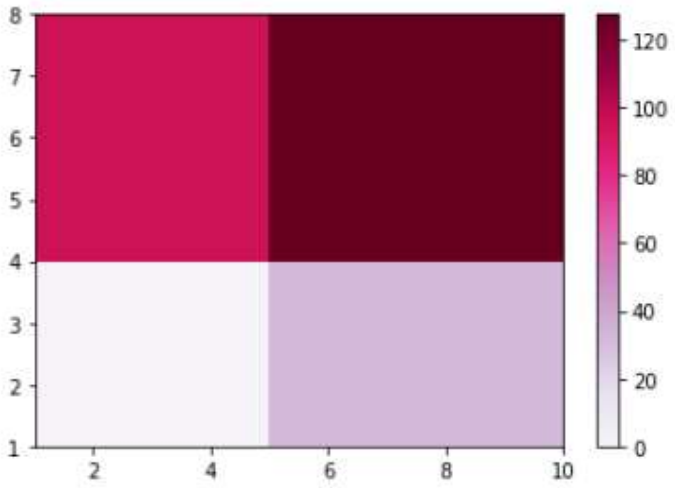
fuente: https://scikit-learn.org/stable/

De hecho, los "rectángulos" en cuestión están limitados por los valores de x e y. En el este ejemplo los valores de x 1, 5 y 10 definen dos rectángulos, uno que va a tener como límites en una dimensión los valores 1 y 5, y otro que va a tener como límites los valores 3 y 10.

Los valores de x y de y determinan los límites de los rectángulos, lo que significa el array z no necesita tener más que x-1 elementos horizontales e y-1 elementos verticales. O dicho con otras palabras, de nuestro array z de tamaño 3x3 se está ignorando una fila y una columna como se muestra a continuación.

```
[[  0.     31.875  63.75 ]
 [ 95.625 127.5   159.375]
 [191.25  223.125 255.   ]]
```

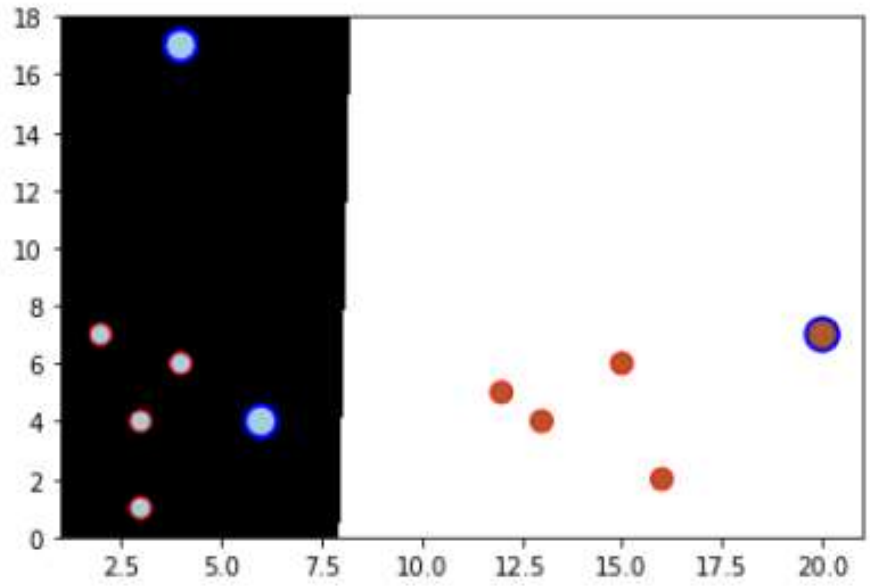
Al graficar los datos anteriores nos quedan los cuatro rectángulos:



Paso 8

Finalmente podemos poner sobre un gráfico todos los puntos tanto los de entrenamiento como los de testeo y comprobar que el modelo a separar correctamente los valores:

```
plt.scatter(X[:, 0], X[:, 1], c=Y, s=80, edgecolors='red', linewidth=1, cmap=plt.cm.Paired)

plt.scatter(Xn[:, 0], Xn[:, 1], c=clasificador.predict(Xn), s=180, edgecolors='blue', linewidth=2, cmap=plt.cm.Paired)


plt.show()
```


18
16
14
12
10
8
6
4
2
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Bibliografía utilizada y sugerida

Libros

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Mastering Exploratory Analysis with pandas - By Harish Garg - September 2018

Manual online

(2019, 01). Obtenido 01, 2019, de https://pandas.pydata.org/

(2019, 01). Obtenido 01, 2019, de https://matplotlib.org/

(2019, 01). Obtenido 01, 2019, de https://scikit-learn.org/stable/

Lo que vimos

En esta unidad hemos comenzado a trabajar con modelos de aprendizaje supervisado tanto de regresión lineal como de clasificación.

Lo que viene:

En la siguiente unidad seguiremos trabajando con casos de aprendizaje supervisado.