Tarea 1 Predecir la necesidad de una cesárea

@Author: Maite Sánchez Fornaris

@email: maite.sanchez@reduc.edu.cu

Los especialistas en gineco-obstretricia deciden hacerle una cesárea a una parturienta basados en distintas constantes físicas que describen el estado de la mujer y del bebé. En este caso, se se toman solamente algunas de las constantes vitales y padecimientos crónicos de la madre para entrenar un perceptrón simple y predecir resultados a partir del entrenamiento hecho.

Las variables que se utilizaron fueron:

> @attribute 'Age' { 22,26,28,27,32,36,33,23,20,29,25,37,24,18,30,40,31,19,21,35,17,38 }
>
> @attribute 'Delivery number' { 1,2,3,4 }
>
> @attribute 'Delivery time' { 0,1,2 } -> {0 = timely , 1 = premature , 2 = latecomer}
>
> @attribute 'Blood Pressure' { 2,1,0 } -> {0 = low , 1 = normal , 2 = high }
>
> @attribute 'Heart Problem' { 1,0 } -> {0 = apt, 1 = inept }
>
> @attribute 'Caesarian' { 0,1 } -> {0 = No, 1 = Yes }

Donde los rasgos de entrada son la edad, el número de partos, momento del parto, presión arterial, problemas del corazón previos. De estas variables, la edad y el número de partos son nominales; mientras que el momento del parto, la presión arterial y la variable problemas del corazón previos son ordinales.

Lo que se desea predecir es el atributo cesarea que se considera una variable ordinal.

```
import numpy as np
import pandas as pd

# Carga de los datos, use pandas por la flexibilidad y visibilidad que le da a los
dataframes
df = pd.read_csv('./caesarian.csv.arff', header=None, sep=',', comment='@',
skiprows=[0,9])
df.columns = ['age', 'delivery_number', 'delivery_time', 'blood_pressure',
'heart_problems', 'caesarian']
df
>>>
```

	age	delivery_number	delivery_time	blood_pressure	heart_problems	caesarian
0	22	1	0	2	0	0
1	26	2	0	1	0	1
2	26	2	1	1	0	0
3	28	1	0	2	0	0
4	22	2	0	1	0	1

	age	delivery_number	delivery_time	blood_pressure	heart_problems	caesarian
...	...	...	...	...	...	...
75	27	2	1	1	0	0
76	33	4	0	1	0	1
77	29	2	1	2	0	1
78	25	1	2	0	0	1
79	24	2	2	1	0	0

80 rows × 6 columns

```
# Se convierten los valores del dataframe a ndarray de Numpy
arr = np.array(df.values, dtype=float)
arr

# Se declara una clase Perceptron que contiene toda la logica
class Perceptron:
    """
        La funcion fit permite entrenar un perceptron simple, el bias se incluyo
dentro del arreglo
        de los pesos (denominado weights).
        @params:
        X: arreglo numpy bidimensional
        y: arreglo numpy unidimensional
        n_iter: el numero de veces que se ejecutara el perceptron para ajustar los
pesos
    """
    def fit(self, X, y, n_iter=100):
        n_samples = X.shape[0]
        n_features = X.shape[1]

        # Add 1 for the bias term
        self.weights = np.zeros((n_features+1,))

        # Add column of 1s
        X = np.concatenate([X, np.ones((n_samples, 1))], axis=1)
        for i in range(n_iter):
            for j in range(n_samples):
                if y[j]*np.dot(self.weights, X[j, :]) <= 0:
                    self.weights += y[j]*X[j, :]
    """
        Esta funcion permite predecir etiquetas para nuevos conjuntos de datos de
entrada.
        El entrenamiento se realiza mediante una multiplicacion de matrices entre
la X de entrada
        y los pesos previamente calculados y se mapean a 1 o -1, segun corresponda.
Verificando
        primeramente si ya existe un arreglo de pesos, que se genera mediante
        el metodo fit. El arreglo numpy que recibe el metodo debe tener exactamente
la misma
        cantidad de columnas que el que se uso en el metodo fit.
        @params:
        X: arreglo numpy bidimensional
    """
    def predict(self, X):
        if not hasattr(self, 'weights'):
            print('The model is not trained yet!')
            return
```

```
        n_samples = X.shape[0]
        # Add column of 1s
        X = np.concatenate([X, np.ones((n_samples, 1))], axis=1)
        y = np.matmul(X, self.weights)
        y = np.vectorize(lambda val: 1 if val > 0 else -1)(y)
        return y

    """
        Esta funcion permite calcular la media de las veces en que el algoritmo
acierta
        con la etiqueta que le asigna el perceptron, internamente llama al metodo
predict
        de la clase Perceptron.
        @params:
        X: arreglo numpy bidimensional
        y: arreglo numpy unidimensional
    """
    def score(self, X, y):
        pred_y = self.predict(X)
        return np.mean(y == pred_y)
# Crear los intervalos train/test
print(len(arr))
train = arr[:57]
# print("Training dataset")
# print(train)
test = arr[57:]
# print("Testing dataset")
# print(test)
>>> 80


# Probar el perceptron para 100 iteraciones
# Crear los conjuntos de entrada y de salida
X = train[:,0:6]
# print(X)
y = train[:,5:]
# print(y)
# Declarar un objeto perceptron
model = Perceptron()
# Entrenar el perceptron
model.fit(X, y, 100)
X_test = test[:,0:6]
y_test = test[:,5:]
y_hat = model.predict(X_test)
print('Number of epochs:', 100)
for i in range(len(y_hat)):
    print('original_y:', y_test[i], 'y_hat:', y_hat[i])
print(model.score(X_test, y_test))

>>>Number of epochs: 100
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
```

```
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
0.6521739130434783

# Probar el perceptron para 500 iteraciones
# Crear los conjuntos de entrada y de salida
X = train[:,0:6]
# print(X)
y = train[:,5:]
# print(y)
# Declarar un objeto perceptron
model = Perceptron()
# Entrenar el perceptron
model.fit(X, y, 500)
X_test = test[:,0:6]
y_test = test[:,5:]
y_hat = model.predict(X_test)
print('Number of epochs:', 500)
for i in range(len(y_hat)):
    print('original_y:', y_test[i], 'y_hat:', y_hat[i])
print(model.score(X_test, y_test))

>>>Number of epochs: 500
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
0.6521739130434783
```

```
# Probar el perceptron para 1000 iteraciones
# Crear los conjuntos de entrada y de salida
X = train[:,0:6]
# print(X)
y = train[:,5:]
# print(y)
# Declarar un objeto perceptron
model = Perceptron()
# Entrenar el perceptron
model.fit(X, y, 1000)
X_test = test[:,0:6]
y_test = test[:,5:]
y_hat = model.predict(X_test)
print('Number of epochs:', 1000)
for i in range(len(y_hat)):
    print('original_y:', y_test[i], 'y_hat:', y_hat[i])
print(model.score(X_test, y_test))

>>>Number of epochs: 1000
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [1.] y_hat: 1
original_y: [0.] y_hat: 1
0.6521739130434783
```

Resultados del perceptrón simple

Los pesos iniciales y el bias del perceptrón se inicializaron arbitrariamente en cero y luego se fueron actualizando por el método estudiado en clase. Se probó el algoritmo para 100, 500 y 1000 ejecuciones y siempre da una media de acierto de aproximadamente el 0.65, aunque no se probó a hacer predicciones aleatorias.