Inteligencia Artificial - IAI84

Instituto Tecnológico Metropolitano

Pedro Atencio Ortiz - 2019

En este notebook se abordan los temas:

- Regresión lineal.
- Descenso del gradiente.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Un modelo lineal se define como: $Y = WX + b$

In [ ]:
```
np.random.seed(2) #aseguramos que el resultado del random es el mismo er

m = 100
X = 2 * np.random.rand(m,1)

W = np.random.randint(3,8)+np.random.rand()
b = np.random.randint(3,8)+np.random.rand()

Y = W * X + b  #regresor lineal

Y = Y + np.random.randn(m,1) #ruido
```

In [ ]:
```
print b
```

In [ ]:
```
plt.scatter(X, Y, color='red')
plt.xlabel("x", fontsize=15)
plt.ylabel("y", fontsize=15)
plt.show()
```

La regresión lineal consiste en hallar los valores de W y b tales que definan una recta

que se acerque lo mejor posible a los valores originales de Y dados los X de entrada.
En este punto, podemos adivinar dichos valores?

In [ ]:
```
W = None
b = None

Y_pred = W * X + b

plt.scatter(X, Y, color='red')
plt.plot(X, Y_pred)

plt.xlabel("x", fontsize=15)
plt.ylabel("y", fontsize=15)
plt.show()
```

La regresión lineal permite encontrar un valor a y un valor b tal que la linea que define la siguiente ecuación, minimiza la distancia de todos los datos respecto a la misma.

$$y = aX + b$$

La solución exacta o ecuación normal de la regresión lineal es la siguiente:

$$X = [1, X]$$
$$\theta = [b, a] = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$$

Es decir, encontrando r encontramos los valores de a y b que resuelven el problema de ajuste lineal.

In [ ]:
```
X_b = np.c_[np.ones((m,1)), X] #X with bias
#theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(Y)
theta = None
```

In [ ]:
```
print theta
```

In [ ]:
```
#Ejemplo de prediccion para dos datos del conjunto

x = np.array([[0], [2]])
y_solution = theta[1]*x + theta[0]


###En forma vectorizada
#x = np.array([[0], [2]])
#x_b = np.c_[np.ones((2, 1)), x]
#y_solution = x_b.dot(theta)

print y_solution
```

In [ ]:
```
plt.scatter(X, Y, color='red')
plt.plot(x, y_solution)
```

```
plt.plot(x, y_solution)
plt.show()
```

Implementemos el ejemplo anterior mediante descenso del gradiente

El algoritmo del descenso del gradiente se basa en minimizar una función de error $g(x)$ utilizando las derivadas de los parámetros del modelo respecto a dicha función. Sea θ un conjunto de parámetros, X un conjunto de elementos de entrada y Y un conjunto de elementos de salida:

$$
\begin{aligned}
&repeat\{ \\
&\quad Y_{pred} = f(x) = \theta X \\
&\quad e = g(Y, Y_{pred}) \\
&\quad d\theta = \frac{de}{d\theta} \\
&\quad \theta = \theta - d\theta \\
&\}
\end{aligned}
$$

Analicemos

Que pasa con la linea $\theta = \alpha - d\theta$ cuando $d\theta$ es muy grande?

Algoritmo del gradiente con tasa de apredizaje:

$$
\begin{aligned}
&repeat\{ \\
&\quad Y_{pred} = f(x) = \theta X \\
&\quad e = g(Y, Y_{pred}) \\
&\quad d\theta = \frac{de}{d\theta} \\
&\quad \theta = \theta - \alpha d\theta \\
&\}
\end{aligned}
$$

In [ ]:
```
X_new = X.T
Y_new = Y.T

print X.shape
```

Implementación per-data

In [ ]:
```
np.random.seed(2)
```

```
W = np.random.random()
b = 0

print(W, b)
```

In [ ]:
```
alpha = 0.05
epochs = 1000

cost_list = []

m = X_new.shape[1] #numero de datos en el dataset

for epoca in range(epochs): #numero de veces que se repite el descenso d
    dW = 0
    db = 0

    cost = 0

    for i in range(m):
        Ai = W * X_new[0,i] + b

        dz = Ai - Y_new[0,i]
        dW += dz * X_new[0, i]
        db += dz

        cost += (Ai - Y_new[0, i])**2

    #promedio de las derivadas
    dW = dW / m
    db = db / m

    #promedio del costo
    cost = cost / m

    #actualizacion
    W = W - alpha * dW
    b = b - alpha * db

    if(epoca % 100 == 0): #imprimir cada 100 epocas

        print "Cost: ", cost
        cost_list.append(cost)

plt.plot(cost_list)
plt.show()
```

In [ ]:
```
print(W, b)
```

In [ ]:
```
y_solution = W * X + b #prediccion
```

```
plt.scatter(X, Y, color='red')
plt.plot(X, y_solution)
plt.show()
```

Implementación Vectorizada

In [ ]:
```
#Funcion de error: error medio cuadrado
def mse(a, y):
    m = y.shape[1]
    return np.sum((a - y)**2) / m
```

In [ ]:
```
np.random.seed(2)

W = np.random.random([1, X.shape[1]])
b = np.array([[0]])

print(W, b)
```

In [ ]:
```
alpha = 0.05
epochs = 1000

cost_list = []

for epoca in range(epochs):
    A = W.T.dot(X_new) + b

    dz = A - Y_new
    dW = np.dot(X_new, dz.T) / m
    db = np.sum(dz) / m

    W = W - alpha * dW
    b = b - alpha * db

    if(epoca % 100 == 0):
        cost = mse(A, Y_new)
        print "Cost: ", cost
        cost_list.append(cost)

plt.plot(cost_list)
plt.show()
```

In [ ]:
```
print W,b
```

In [ ]:
```
y_solution = W.T.dot(X_new) + b

plt.scatter(X, Y, color='red')
plt.plot(X, y_solution.T)
plt.show()
```

In [ ]:
```

```