

Práctica 1: regresión lineal

Aprendizaje Automático y Big Data



11 de octubre de 2018

Felix Villar Y víctor ramos

Universidad Complutense de Madrid

1. ***Regresión lineal con una variable***

***Código:***

import pandas as pd #Biblioteca para análisis de datos

import numpy as np #Biblioteca para operaciones numéricas

import matplotlib.pyplot as plt #Biblioteca para representar funciones

archivo = open("ex1data1.csv") #Abrimos el archivo csv

df = pd.read\_csv(archivo, header=None, names=['poblacion', 'ingresos']) #Genera un dataframe con los datos

df.head(100) #Coge los 5 primeros datos del dataframe por defecto

def calc\_costo(df, th0, th1):

poblacion = df['poblacion']

prediccion = poblacion \* th1 + th0

ingresos = df['ingresos']

return np.sum(np.square((prediccion - ingresos)) / len(df) / 2.0)

def grad\_desc(df, th0, th1, alpha):

length = len(df)

df['prediccion'] = df['poblacion'] \* th1 + th0

th0 = th0 - alpha / length \* np.sum((df['prediccion'] - df['ingresos']))

th1 = th1 - alpha / length \* np.sum(((df['prediccion'] - df['ingresos']) \* df['poblacion']))

return th0, th1

def test\_graph(df, iteraciones):

theta0, theta1 = 0, 0

alpha = 0.01

costo = 0

for elem in range(iteraciones):

theta0, theta1 = grad\_desc(df, theta0, theta1, alpha)

costo = calc\_costo(df, theta0, theta1)

plt.plot(df['poblacion'], df['ingresos'], 'b.', df['poblacion'], df['poblacion']\*theta1 + theta0, 'r-')

plt.title('Método del gradiente')

plt.xlabel('Población de la ciudad en 10.000s')

plt.ylabel('Ingresos en $10000s')

plt.text(5, 23, 'Costo: {}'.format(costo))

plt.show()

test\_graph(df, 1500)

![Imagen que contiene mapa, texto

Descripción generada con confianza muy alta]()

1. ***Regresión lineal con varias variables***

***Código:***

import numpy as np

def normalizar(x):

media = np.mean(x, axis=0)

desv = np.std(x, axis=0)

normalizar = (x - media) /desv

return normalizar, media, desv

def calc\_costo(entradas, salidas, th):

prediccion = np.sum(entradas \* th,axis=1)

fun = prediccion -salidas

return np.matmul(np.atleast\_2d(fun).T,np.atleast\_2d(fun))/2\*len(entradas)

def grad\_desc(entradas,salidas,th,alpha):

length = len(entradas)

prediccion = np.sum(entradas \* th,axis=1)

fun=np.multiply((prediccion - salidas)[:,np.newaxis],entradas)

r= alpha/length\*np.sum(fun,axis=0)

th= th-r

return th

def normec(entradas,salidas,unos):

pob = np.concatenate((np.atleast\_2d(unos).T,entradas),axis=1)

a=np.matmul(np.transpose(pob),pob)

b=np.linalg.pinv(a)

c=np.matmul(b,np.transpose(pob))

d=np.matmul(c,salidas)

return d

def mulvar(arch, iteraciones):

x =np.genfromtxt(arch, delimiter = ',')

numparametros = x[0].size - 1

unos = np.ones(int(x.size/(numparametros+1)))

solucion = normec(x[:,:-1],x[:,-1],unos)

entradas,media,desv = normalizar(x[:,:-1])

entradas = np.concatenate((np.atleast\_2d(unos).T,entradas),axis=1)

salidas = x[:, -1]

th = np.zeros(numparametros+1)

alpha = 0.03856

costo = 0

for elem in range(iteraciones):

th = grad\_desc(entradas, salidas, th, alpha)

costo = calc\_costo(entradas, salidas, th)

print('El coste de manera vectorizada')

print(costo)

#Ejemplo para demostrar que funciona el metodo de descenso

#Piso con 1650 metros cuadrados y 3 habitaciones, hay que normalizar los datos

print('La solucion de descenso de gradiente')

print(th)

print(th[0]+th[1]\*((1650-media[0])/desv[0])+th[2]\*((3-media[1])/desv[1]))

print('La solucion de la ecuacion normal')

print(solucion)

print(solucion[0]+solucion[1]\*1650+solucion[2]\*3)

mulvar('ex1data2.csv',1500)

![Imagen que contiene texto, periódico

Descripción generada con confianza muy alta]()