LogisticRegression

March 19, 2021

Nicholas Paisley

[ ]:
```
#Import libraries
import autograd.numpy as np
from autograd import grad
import matplotlib.pyplot as plt
```

[ ]:
```
#Logistic Regression is basically a predictive model analysis technique where␣
↪the target variables (output) are discrete values for a given set of␣
↪features or input (X).
#Example: 1->Pass 2->Fail (Discrete variables)

def logistic(x,a,b): #creating a definition named "logistic" with "x" , "a" ,␣
↪and "b" variables
    return 1/(1.0+np.exp(-a*x+b)) #returns the equation y(x) = 1/(1+e^(-mx+b)))␣
↪or in this case y(x) = 1/(1+e^(-ax+b)))
    #This function is the sigmoid function

def loss(x,y_obs,a,b): #creating a definition named "loss" with "x" , "y_obs"␣
↪,"a" and "b" variables
    y_model = logistic(x,a,b) #creating a variable "y_model" that returns the␣
↪logistic definition (the sigmoid function/formula)
    return np.sum( (y_model-y_obs)**2 ) #This is the SSR function that is␣
↪returned in the "loss" definition

x = np.array( [0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,2.50,2.75,3.00,3.
↪25,3.50,4.00,4.25,4.50,4.75,5.00,5.50] )  #Array of numbers for the x␣
↪coordinates (in an array)
y_obs = np.array( [0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1] ) #Array of numbers␣
↪for the y_obs coordinates (in an array) --> The discrete variables

a,b = 5.0,3.5 #Creating stagnant "a" and "b" variables (guesses)
y_model = logistic(x,a,b) #Creating a "y_model" variable using the linear␣
↪defintion with the already defined "x" , "a" , and "b" variables

print('First guess at a, b ',a,b) #prints the a and b variables
```

```python
print('First loss function is ',loss(x,y_obs,a,b)) #prints the SSR function using the defined "a", "b" and the "x" and "y_obs" arrays

d_by_da = grad(loss,2) #create the derivative/gradient function of "loss" --> called d_by_da, (equation(loss), index=2 (a))
d_by_db = grad(loss,3) #create the derivative/gradient function of "loss" --> called d_by_da, (equation(loss), index=3 (b))

#Learning rate gives the rate of speed where the gradient moves during gradient descent. Setting it too high would make your path instable, too low would make convergence slow.
#Put it to zero means your model isn't learning anything from the gradients.
learning_rate = 0.001 #controls the magnitude of the vector update (positive number that moves the starting point by a very small number)
maximum_number_of_iterations = 50000 #number of iterations given

ssr = [] #Sum of Square Residuals -> The lower it is the better it fits the linear model

for iter in range(maximum_number_of_iterations):
    #We are finding new "a" (slope) and "b" (intercept) variables for 50000 iterations.
    a -= learning_rate*d_by_da(x,y_obs,a,b) #(-=) subtracts the value of the expression on the right-hand side from the value on the left-hand side and then assigns the result to the left hand side variable.
    #We are changing the slope for every iteration and saving the new slope off as the new "a".
    b -= learning_rate*d_by_db(x,y_obs,a,b) #(-=) subtracts the value of the expression on the right-hand side from the value on the left-hand side and then assigns the result to the left hand side variable.
    #We are changing the intercept for every iteration and saving the new slope off as the new "b".
    y_model = logistic(x,a,b) #Creates our y_model with our "a" and "b" and the array of "x" values
    ssr.append(loss(x,y_obs,a,b)) #adds to the empty array list above of all of the SSR generated from the current "a" and "b".

print('Best a and b are ',a,b) #outputs the "a" and "b" variable that gives the SSR value
print('Best loss function is ',loss(x,y_obs,a,b)) #outputs the best SSR value

plt.subplot(1,2,1) #creates 2 plots with this plot being the 1st one shown
plt.scatter(x,y_obs) #puts the "x" and "y_obs" values from the arrays on the graphs
plt.plot(x,y_model) #plots the best y_model for the linear regression
```

```
plt.subplot(1,2,2) #creates 2 plots with this plot being the 1st one shown
plt.plot(ssr) #creates a graph of all the calculated SSR values

plt.show() #outputs the graphs
```

```
[ ]: plt.semilogy(ssr) #plots ssr function
```