the matrix not semidefinite

=

If = 0 and =2 then would be < 0 so it would not be positive semidefinite

Consider A to be then =

which would be >0 for all values of so our new A would be PSD

* 1. =

A taking the 2d derivative where p!=q we can see that taking the derivative of any of the above with any theta value not already used in the simplification of the problem would result in 0



≤ 0 => g(≥ 0

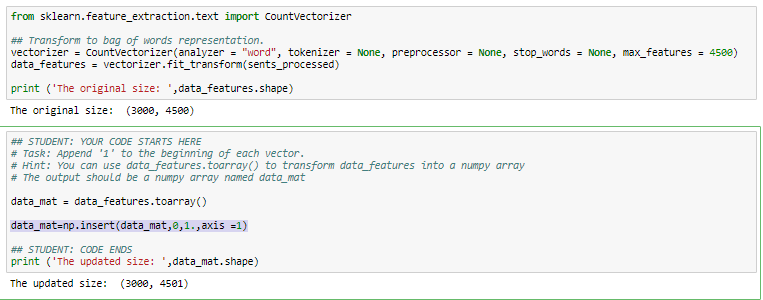
≤ 0 => g(≥ 0

≤ 0 => g(≥ 0

1. 1. = = ->
   2. = =||x||^2 which is greater than or equal to 0 for all x so the function is psd so it is convex

#####################################################################################

1. data\_mat=np.insert(data\_mat,0,1.,axis =1)





## STUDENT: Start of code ###

# - yx

# \_\_\_\_\_\_\_\_\_\_\_\_

# ((e^yθ^Tx)+1)

#initialize ld(theta)

derivatives=np.zeros(weights.size)

for i in range(labels.size):

#dot product of weigths with each column of feature matrix

column\_product=np.dot(weights,feature\_matrix[i,:])

temp=labels[i]\*column\_product

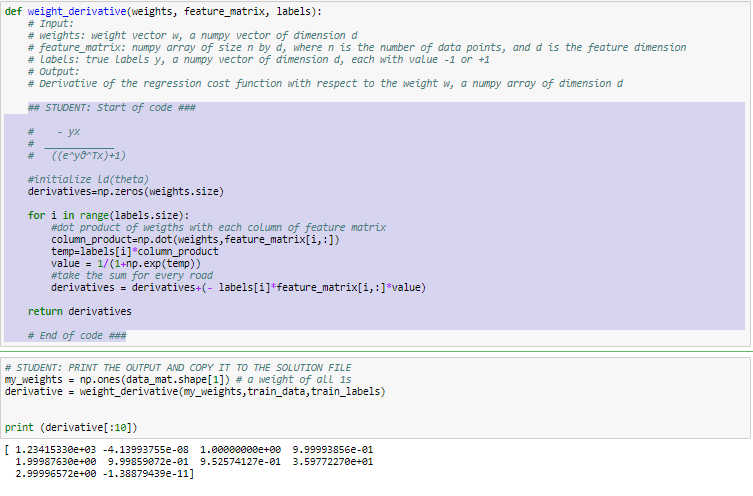
value = 1/(1+np.exp(temp))

#take the sum for every road

derivatives = derivatives+(- labels[i]\*feature\_matrix[i,:]\*value)

return derivatives

# End of code ###





temp\_weight = [0.5]\*(len(train\_data[0]))

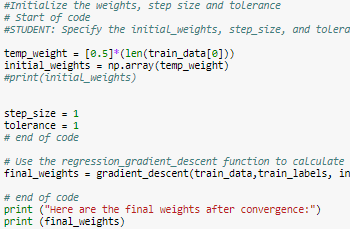
initial\_weights = np.array(temp\_weight)

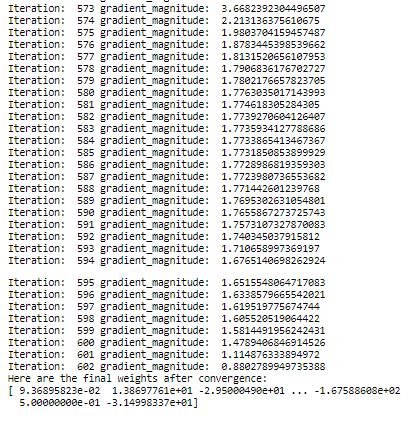
#print(initial\_weights)

step\_size = 5

tolerance = 2

# end of code







## STUDENT: CODE STARTS HERE

## Pull out the parameters (theta\_0, theta) of the logistic regression model

theta = final\_weights #gradient\_descent(train\_data,train\_labels, initial\_weights, step\_size, tolerance)

theta0 = theta[0]

theta = np.delete(theta,0)

## STUDENT: CODE ENDS HERE



## STUDENT: YOUR CODE HERE

predict\_array = []

for i in range(len(feature\_matrix)):

dot = np.dot(feature\_matrix[i],weights)

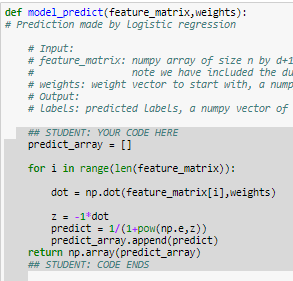
z = -1\*dot

predict = 1/(1+pow(np.e,z))

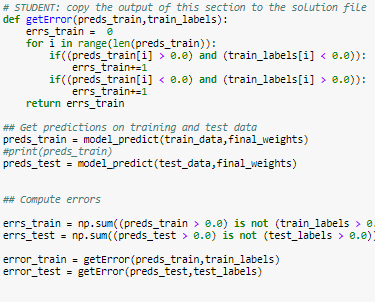
predict\_array.append(predict)

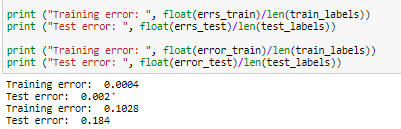
return np.array(predict\_array)

## STUDENT: CODE ENDS



#not test error did not work as written so I wrote my own





## STUDENT: YOUR CODE HERE

model\_predict\_arr = model\_predict(feature\_matrix, weights)

count =0;

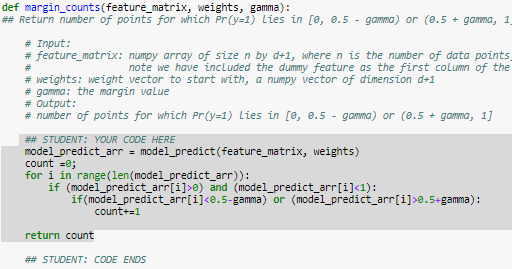
for i in range(len(model\_predict\_arr)):

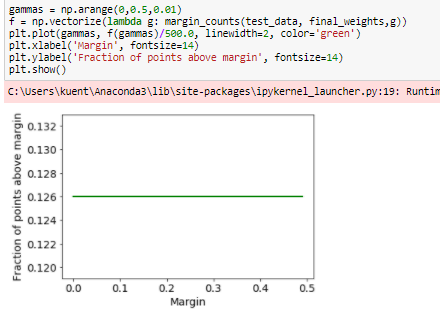
if (model\_predict\_arr[i]>0) and (model\_predict\_arr[i]<1):

if(model\_predict\_arr[i]<0.5-gamma) or (model\_predict\_arr[i]>0.5+gamma):

count+=1

return count







## STUDENT: YOUR CODE HERE

model\_predict\_arr = model\_predict(feature\_matrix, weights)

denominator=len(feature\_matrix)

numerator = 1

for i in range(len(model\_predict\_arr)):

if (model\_predict\_arr[i]>0) and (model\_predict\_arr[i]<1):

if(model\_predict\_arr[i]<(0.5-gamma)) or (model\_predict\_arr[i]>(0.5+gamma)):

denominator+=1

if(labels[i]<(0.5-gamma)) and (model\_predict\_arr[i]<(0.5-gamma)):

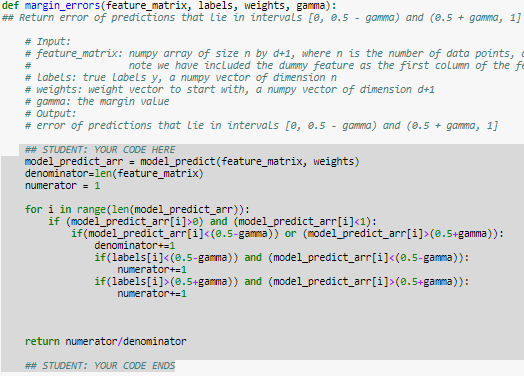
numerator+=1

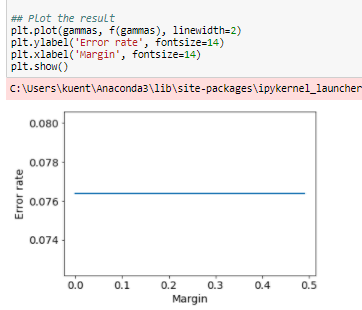
if(labels[i]>(0.5+gamma)) and (model\_predict\_arr[i]>(0.5+gamma)):

numerator+=1

return numerator/denominator

## STUDENT: YOUR CODE ENDS







## STUDENT: YOUR CODE HERE

#the first index is the larges value and the last index is the smallest value

#the value that is going to be excluded firrst from the array is in the back

weights= np.delete(final\_weights,0)

print(len(weights))

large\_positive = []

small\_negative = []

for i in range(len(weights)):

#initialize array

if (len(large\_positive)<10):

large\_positive.append([vocab[i],weights[i]])

small\_negative.append([vocab[i],weights[i]])

else:

#larger than smallest value in array

if (final\_weights[i]>large\_positive[0][1]):

#remove small add large

large\_positive.pop(0)

large\_positive.append([vocab[i],weights[i]])

large\_positive.sort(key=lambda x: x[1])

#smaller than largest value in array

if (final\_weights[i]<small\_negative[9][1]):

#remove large add small

small\_negative.pop(9)

small\_negative.append([vocab[i],weights[i]])

small\_negative.sort(key=lambda x: x[1])

#remove weights from print statement

final\_positive = []

final\_negative = []

for i in range(10):

final\_positive.append(large\_positive[i][0])

final\_negative.append(small\_negative[i][0])

print('Top Ten')

print(large\_positive)

print('Bottom Ten')

print(small\_negative)

## STUDENT: CODE ENDS



Each is accompanied with their theta value

