Assignment$_4$

Chaitra , Ganesh , Avik

May 24 2019

1 Question 1

Let

$$f(a_1, a_2, ..., a_+n) = ln(e^{a_1} + e^{a_2} + ... + e^{a_n})$$

Show that f is convex.

Proof. Let

$$u_i = e^{a_i}, v_i = e^{b_i}$$

where

$$a = a_i, b = b_i$$

because

$$f(a) = ln \sum_{i=1}^{n} e^{a_i}, f(b) = ln \sum_{i=1}^{n} e^{b_i}$$

$$a = (a_1, a_2, ..., a_n) = a_i, b = (b_1, b_2, ..., b_n) = b_i$$

By definition,

$$f[(1-\theta)b + \theta a] = ln[\sum_{i=1}^{n} e^{(1-\theta)b_i + \theta a_i}]$$

$$= ln[\sum_{i=1}^{n} e^{(1-\theta)b_i} e^{\theta a_i}]$$

$$= ln[\sum_{i=1}^{n} v_i^{1-\theta} u_i^{\theta}] ①$$

From Hölder's Inequality,

$$\sum_{i=1}^{n} a_i b_i \leq (\sum_{i=1}^{n} |a_i|^p)^{1/p} (\sum_{i=1}^{n} |b_i|^q)^{1/q}$$

Applying the inequality to ①,

$$f[(1-\theta)b + \theta a] = ln[\sum_{i=1}^{n} v_i^{1-\theta} u_i^{\theta}] \leq ln[(\sum_{i=1}^{n} v_i^{1-\theta.(1/1-\theta)})^{1-\theta} (\sum_{i=1}^{n} u_i^{\theta.(1/\theta)})^{\theta}]$$

$$= ln[(\sum_{i=1}^{n} v_i^{1-\theta})(\sum_{i=1}^{n} u_i^{\theta})]$$

$$= ln(\sum_{i=1}^{n} v_i^{1-\theta}) + ln(\sum_{i=1}^{n} u_i^{\theta})]$$

$$= (1-\theta) ln \sum_{i=1}^{n} v_i + \theta ln \sum_{i=1}^{n} u_i$$

$$= (1-\theta) ln \sum_{i=1}^{n} e^{b_i} + \theta ln \sum_{i=1}^{n} e^{a_i}$$

$$= (1-\theta) f(b) + \theta f(a)$$

Thus,

$$f[(1-\theta)b + \theta a] \leq (1-\theta) f(b) + \theta f(a)$$

Hence f is convex. □

2 Question 2

Suppose that k(.,.) and k'(.,.) are kernels. Prove that l(.,.) where l(x,y) = k(x,y) + k'(x,y) is also a kernel.

Proof. By definition we have:

$$K: R^d \times R^d \to R$$

$$\phi: R^d \to H$$

$$\forall x, \tilde{x} \in R^d: K(x, \tilde{x}) =< \phi(x), \phi(\tilde{x}) > \textcircled{1}$$

Let H' be the Hilbert space for Kernel K & let H" be the Hilbert space for K'.

We get feature maps,

$$\phi': X' \to H'$$

$$\phi'': X'' \to H''$$

$$\Rightarrow \phi'(x) = (\phi_{'}^{'}(x), ..., \phi_{m1}^{'}(x))$$

$$\phi''(x) = (\phi_{'}^{''}(x), ..., \phi_{m2}^{''}(x))$$

By property of direct sum, define H by concatenating feature maps or alternate features if spaces are infinite.

$$H = H' \oplus H''$$

$$\Rightarrow \phi(x) = (\phi_{'}^{'}(x), ..., \phi_{m1}^{'}(x), \phi_{'}^{''}(x), ..., \phi_{m2}^{''}(x))$$

$$\phi(y) = (\phi_{'}^{'}(y), ..., \phi_{m1}^{'}(y), \phi_{'}^{''}(y), ..., \phi_{m2}^{''}(y))$$

The mapping satisfies,

$$\phi(x).\phi(y) = \phi^{'}(x).\phi^{'}(y) + \phi^{''}(x).\phi^{''}(y)$$

By ①

$$l(x, y) = k(x, y) + k'(x, y)$$

□

3 Question 3

Let's implement SVM via the stochastic subgradient descent algorithm (Slide 32 - Lecture 3).
a) Calculate the gradient vector for the SVM (relative to w and b).
Gradient vector relative to w,

$$\delta f/\delta w = 2/2|w| - C\sum y_i x_i - 0$$

$$= |w| - C\sum y_i x_i$$

Gradient vector relative to b,

$$\delta f/\delta b = 0 - 0 - C\sum y_i$$

$$= -C\sum y_i$$

b)

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import random
import math
from bokeh.plotting import figure, show, Figure
from bokeh.models import ColumnDataSource, Label
from bokeh.models.glyphs import Text
from bokeh.palettes import Spectral3, Spectral11
from bokeh.layouts import row, column, gridplot
```

```
import numpy as np
from bokeh.io import output_notebook
from bokeh.palettes import Dark2_5 as palette
# itertools handles the cycling
import itertools
output_notebook()
from enum import Enum


# In[2]:


def getDataFromCSVFile():

    return pd.read_csv('./data/DWH_Training.csv')



# In[3]:


numberOfIteration = 10000
learningRate = 1/numberOfIteration
dataFrame = getDataFromCSVFile()


# In[4]:


def classify(w, b, dataFrameTraining):

    classificationResult = []

    for index, row in dataFrameTraining.iterrows():

        result = ((w[0]*row['height(cm)'] + w[1]*row['weight(kg)'])+b)
        if result > 0:
            classificationResult.append(1)
        else:
            classificationResult.append(-1)

    return classificationResult



# In[5]:
```

```
def splitData(index,fold_size,nFolds,data):
    current_index=index*fold_size
    if(index==nFolds-1):
        training_data=data[0:current_index]
        test_data=data[current_index:]
    else:
        training_data=data[0:current_index].append(data[current_index+fold_size:
        test_data=data[current_index:current_index+fold_size]
    return training_data,test_data



# In[6]:


def getFoldSize(data,nFolds):
    len_data=len(data)
    fold_size=len_data//nFolds
    return fold_size


# In[7]:


def computeGradientAndUpdateHyperParam():

    batchSizes = [1,10,50]
    c = [0.1,1.0,10.0]

    biasMap = dict()
    cMap = dict()
    accuracy = []
    print(dataFrame.head())
    foldSize=getFoldSize(dataFrame,10)
    for m in range (10):
        for index in range(0,numberOfIteration):

            for cElement in c:

                for bElement in batchSizes:

                    updatedW = [-11.07,-16.61]
                    updatedB = 2976.81
                    w = [-11.07,-16.61]
```

```
                    randomDataPointIndicesList = [random.randint(0,len(dataFrame

                    magnitudeOfW = math.sqrt(math.pow(w[0],2)+math.pow(w[1],2))
                    label = 0
                    for element in randomDataPointIndicesList:

                        if (dataFrame['gender'][element] *
                        ((w[0]*dataFrame['height(cm)'][element] + w[1] * dataFra

                            label += dataFrame['gender'][element]
                            w[0] += label * dataFrame['height(cm)'][element]
                            w[1] += label * dataFrame['weight(kg)'][element]

                    w[0] *= cElement
                    w[1] *= cElement

                    w[0] += magnitudeOfW
                    w[1] += magnitudeOfW

                    w[0] *=learningRate
                    w[1] *=learningRate


                    updatedW[0] -= w[0]
                    updatedW[1] -= w[1]

                    updatedB -= ((-cElement)*(label))
                    biasMap[bElement]=(updatedW,updatedB)
                cMap[cElement]=biasMap

        return cMap


# In[8]:



def performPlot(hyperPlaneMap):
    palette = ["SpringGreen", "Crimson"]
    dataFrame['color'] = np.where(dataFrame['gender']==1,"SpringGreen", "Crimson
    dataFrame['legend_values'] = np.where(dataFrame['gender']==1,"Male", "Female
    #dataFrame['line_color'] = ["SpringGreen", "Crimson", "blue","green","black"

#aff a figure
    p=figure(x_axis_label='height(cm)',y_axis_label='weight(kg)', width=800, hei
        title='Relation between Male and Female',
```

```
        )

#render the graph
    p.circle('height(cm)','weight(kg)',source=dataFrame,size=5,alpha=0.8,color='
    p.y_range.start = dataFrame['weight(kg)'].min()
    p.y_range.end = dataFrame['weight(kg)'].max()
    p.x_range.start = dataFrame['height(cm)'].min()
    p.x_range.end = dataFrame['height(cm)'].max()

    hyperPlaneValues = []
    xvalues = []


    accuracy, cI, bi = crossValidation(dataFrame,10,hyperPlaneMap)
    tupleValues = hyperPlaneMap[cI][bi]
    weight = tupleValues[0]
    bias = tupleValues[1]
    print(tupleValues)
    print(cI)
    print(bi)
    print(accuracy)
    for index, row in dataFrame.iterrows():
        x = row['height(cm)']
        elements = []
        elements.append((-(weight[0]/weight[1])*x)-(bias/weight[1]))
        hyperPlaneValues.append(elements)
        xvalues.append(row['height(cm)'])
        p.line(x=xvalues , y=hyperPlaneValues, line_width=1, color='blue')
    show(p)



# In[9]:


def crossValidation(data,nFolds,hyperPlaneMap):

    maxb = 0
    maxIndexb = 0
    acc = []
    foldSize=getFoldSize(data,nFolds)
    count = 0
    countC = 0
    maxc = 0
    maxcIndex = 0
    maxcbIndex = 0
```

```
    for c,b in hyperPlaneMap.items():
        maxb = 0
        maxIndexb = 0
        count = 0
        for tupleInfo in b.values():
            weight = tupleInfo[0]
            bias = tupleInfo[1]
            accuracy = []
            for index in range(nFolds):
                training_data,test_data=splitData(index,foldSize,nFolds,data)
                testResults = classify(weight,bias,test_data)
                correctClassificationCount = 0
                test=len(test_data)
                for j in range(test):
                    test_row=test_data.iloc[j]
                    if testResults[j] == int(test_row['gender']):
                        correctClassificationCount +=1
                    else:
                        pass
                acc=correctClassificationCount/test*100
                accuracy.append(acc)
            avgAcc = np.sum(accuracy)/len(accuracy)
            if  avgAcc > maxb:
                maxb = avgAcc
                maxIndexb = list(b.keys())[0]
        if maxb > maxc:
            maxc = maxb
            maxIndexc = c
            maxcbIndex = maxIndexb
    return maxc, maxIndexc, maxcbIndex


# In[10]:


def main():

    hyperPlaneMap = computeGradientAndUpdateHyperParam()
    performPlot(hyperPlaneMap)
    print('\n')
    print("Hyperplane Map",hyperPlaneMap)


# In[11]:
```

```
if __name__ == '__main__':
    main()
```

c) Make a scatter plot where the x-axis is the height of the citizens and the y-axis is the weight of the citizens. The color of the points need to be different for males and females. In the same figure, plot the linear classifier calculated.

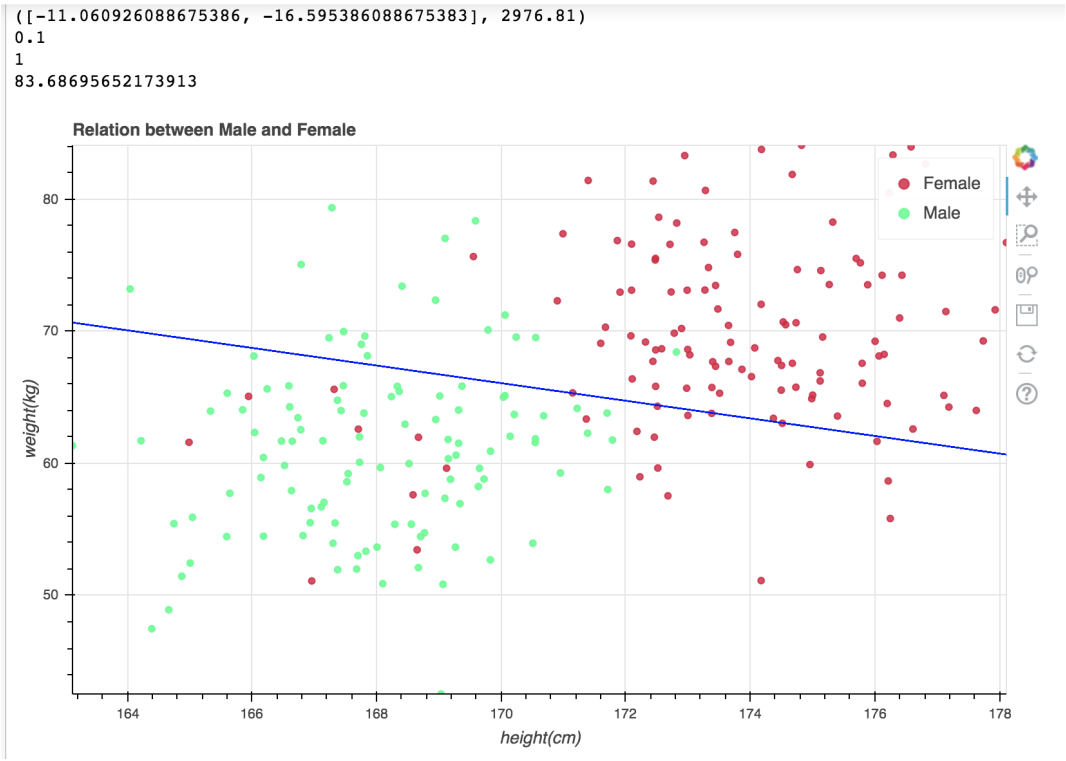

Figure 1: Scatterplot of Question 4b

d) Compare the accuracy with the SVM via LIBLINEAR (in the test set).
Accuracy using SVM Stochastic Subgradient Descent = 83.6869 %
Accuracy using LIBLINEAR SVM = 88.6486 %

e) Now let's use a new dataset BIG-DWH-Training.csv. It has 2,000,000 rows. Execute 10 times SVM via LIBLINEAR and 10 times SVM via stochastic subgradient descent algorithm. Compare the average execution time.
Time considering without plotting the scatterplots.

```
Accuracy = 72.5172% (689946/951424) (classification)

real    6m36.564s
user    6m29.017s
sys     0m7.781s
```

Figure 2: SVM VIA LIBLINEAR

```
([-11.071885388675385, -16.611829988675385], 2976.81)
0.1
1
85.18094999999998


Hyperplane Map {0.1: {1: ([-11.071885388675385, -16.611829988675385], 2976.81)}}

real    6m18.034s
user    6m19.389s
sys     0m1.329s
```

Figure 3: SVM VIA Stochastic Subgradient Descent