$$Name \quad Sudhir \ \ Sharma$$
$$ID \quad 12041500$$
$$Branch \quad CSE$$
$$HomeWork \quad II \ \ CS550$$

Question 1

Deriving Decision Boundary for Learning with Prototypes LwP Classifier

Given training examples of +ve class

```
(2,3,4); (1,1,2); (0,2,0)
```

Given training examples of -ve class

```
(4,3,2); (2,1,1); (3,5,3)
```

Answer

To Compute the average feature vectors of +ve and -ve class examples. Let us call them μ+ i used

1 .

$$\boldsymbol{\mu}+ = \frac{1}{N_+}\Sigma(x_i)$$
$$\boldsymbol{\mu}+ = (\frac{(2+1+0),(3+1+2),(4+2+0))}{3})$$
$$\boldsymbol{\mu}+ = (1,2,2)$$

ii.)

Answer

To Compute the average feature vectors of +ve and -ve class examples. Let us call them μ− i used

$$\boldsymbol{\mu}- = \frac{1}{N_-}\Sigma(x_i)$$
$$\boldsymbol{\mu}- = (\frac{(4+2+3),(3+1+5),(2+1+3)}{3})$$
$$\boldsymbol{\mu}- = (3,3,2)$$

2. Determine the vector $\boldsymbol{w}$ which joins μ+ and $\mu-$: $\boldsymbol{w}$ = $\boldsymbol{\mu}$+ − $\boldsymbol{\mu}$−

$\boldsymbol{\mu}$+ − $\boldsymbol{\mu}$−=(1,2,2)-(3,3,2)

$\boldsymbol{w}$ = (-2,-1,0)

3. Given decision boundary passes through the mid-point of the prototypes.

Given

$$midpoint = \frac{1}{2}(\mu_+ + \mu_-)$$

$$equation => w^T + b = 0$$

$$midpoint = (\frac{(1+3),(2+3),(2+2))}{2})$$

$$midpoint = (2, 2.5, 2)$$

$$b = -w^T x$$

$$b = -\begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 2.5 \\ 2 \end{bmatrix}$$

The value of b come out to be

$$b = 6.5$$

Equation of the Decision boundary is (-2,-1,0)x +6.5=0

4. Compute the distances of the training examples from the decision boundary. Are there any errors in classification using the LwP classifier?

   Distances calculated below part 1 question 5

Boundary Equations came out is

Equation of the Decision boundary is

$$(-2, -1, 0)x + 6.5 = 0$$

Let

$$x = (x, y, z)$$

then boundary equation became

$$-2x - y + 6.5 = 0$$

Distance of training examples from decision boundary

$$d = (\frac{-2x - y + 6.5}{\sqrt{5}})$$

positive class

$$(2,3,4) => \begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + 6.5 = -7 + 6.5 = -0.5 => d1$$

$$(1,1,2) => \begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} + 6.5 = -3 + 6.5 = 2.5 => d2$$

$$(0,2,0) => \begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} + 6.5 = -2 + 6.5 = 4.5 => d3$$

negative class

$$(4,3,2) => \begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix} + 6.5 = -11 + 6.5 = -4.5 => d4$$

$$(2,1,1) => \begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} + 6.5 = -5 + 6.5 = 1.5 => d5$$

$$(3,5,3) => \begin{bmatrix} -2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \\ 3 \end{bmatrix} + 6.5 = -11 + 6.5 = -4.5 => d6$$

Yes there are errors in classification using LwP , we can see that d1 for +ve class give -ve distance and d5 for -ve class give +ve distance. But According to LwP classifier +ve class should have +ve distance and -ve class should have -ve distance. In sort some pos points are classified as neg while some neg points are classified as pos.

5. Find the equation of a linear classifier which separates the data without any error? If no such classifier exists, give a justification.

```
import numpy as np
pos=[[2,3,4],[1,1,2],[0,2,0]]
neg=[[4,3,2],[2,1,1],[3,5,3]]
predicted_w=np.matrix([-2,-1,0])
predicted_b=6.5
```

```
while (True):
  y=0 # flag to check if there is error
  c=0 # count to check at final if the error persists
  # for pos
  for i in range(3):
    np_mat=np.matrix(pos[i]).T
    mat=np.dot(predicted_w,np_mat)
```

```
      to_list=mat.tolist()
      num=to_list[0][0]+predicted_b
      # print(num,"pos")
      if (num<0):
        predicted_w=predicted_w+pos[i]
        predicted_b+=1
        # print(predicted_w,predicted_b,"pos")
        c+=1
    # for neg
    for i in range(3):
      np_mat=np.matrix(neg[i]).T
      mat=np.dot(predicted_w,np_mat)
      to_list=mat.tolist()
      num=to_list[0][0]+predicted_b

      if (num>0):
        predicted_w=predicted_w-neg[i]
        predicted_b-=1

        c+=1

    if (c==0):
      break

print(f'Finalscore for w is {predicted_w} ')
print (f'and final bias is {predicted_b}')
```

```
 Finalscore for w is [[-4 -1  2]]
 and final bias is 6.5
```

```
import numpy as np
# Part 1  4th question compute the distances
pos=[[2,3,4],[1,1,2],[0,2,0]]
neg=[[4,3,2],[2,1,1],[3,5,3]]
w=[-2,-1,0]
```

```
# Distance from pos points
for i in range(3):
  ans=np.dot(w,pos[i])/np.linalg.norm(w)
  print(f'Distance of +ve class {str(pos[i])} from boundry is {ans}')
```

```
 Distance of +ve class [2, 3, 4] from boundry is -3.1304951684997055
 Distance of +ve class [1, 1, 2] from boundry is -1.3416407864998738
 Distance of +ve class [0, 2, 0] from boundry is -0.8944271909999159
```

```
# Distance from neg points
for i in range(3):
```

```
ans=np.dot(w,neg[i])/np.linalg.norm(w)
print(f'Distance of -ve class {str(neg[i])} from boundry is {ans}')
```

```
Distance of -ve class [4, 3, 2] from boundry is -4.919349550499537
Distance of -ve class [2, 1, 1] from boundry is -2.23606797749979
Distance of -ve class [3, 5, 3] from boundry is -4.919349550499537
```

Double-click (or enter) to edit

Question 2

i.) Explain the importance of scaling features for training Large Margin Classifiers?

Scaling is the process of normalizing data, which sets the mean of data to 0 and the standard deviation as 1.

It makes the model easy to learn because if there exists a variable with a large spread it may cause a large loss in training.

ii.)Explain the effect of changing the value of C from very small to very large in a Soft margin ?

For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, we should get misclassified examples, often even if our training data is linearly separable.

Here in

$$C\Sigma_{i=1}^{N} z_i$$

, z_i is the slack, and C mutiplied with it increases the influence of the equation. Therefore, for high levels of C, the model prioritises minimising those slacks over lowering w. For small values of C, $\Sigma_{i=1}^{N} z_i$ is also small, and has less significance in the objective function, and so, allows more violations of margin.

iii.)Consider the 2-D array, arr. What is the output of np.sum(arr, axis=1)?

$$Let \quad arr = [[0, 1, 13], [31, 43, 25]]$$
$$after \quad applying \quad np.sum(arr, axis = 1) => [14, 99]$$

Double-click (or enter) to edit

```
import numpy as np
arr=[[0,1,13],[31,43,25]]
np.sum(arr, axis=1)       # so it gives sum horizontally across the columns.
```

```
array([14, 99])
```

iv.) Which of the following are methods for indexing into a DataFrame?

```
d.) All of the above

data.iloc[[0, 2, 4, 7]]
data.loc[2:5]
data.set_index('column_name')
```

v.) What is an indicator feature

b.)A feature that represents categorical data, using 1's and 0's to denote which categories are present.

An indicator feature is a function which maps to 1 if the datapoint belongs to certain subset or else it maps rest of the points to 0.

The expectation of the indicator function is equal to the probability of the event.

The indicator function is typically used to define the loss function used in classification. The property related to its expectation then implies that the classification risk is the probability of misclassification.

In pictures...

vi.) What is the main purpose of standardizing data?

```
a.) It lets us view data in terms of standard deviations from the average case (i.e. mean)
```

vii.) For the given confusion matrix, please calculate accuracy, precision, recall and F1 score

	Actual	
Predicted	Yes	No
Yes	12	3
No	1	9

$$TP = 12$$
$$TN = 9$$
$$FP = 3$$
$$FN = 1$$

$$1.)Precision = \frac{TP}{FP+TP}$$

$$=> \frac{12}{3+12} => 0.8$$

RECALL

$$2.)Recall = \frac{TP}{FN+TP}$$

$$=> \frac{12}{1+12} => 0.92$$

F1Score

↳ *1 cell hidden*

Accuracy

↳ *2 cells hidden*

ix.) Justify the cost function $L(\boldsymbol{w})$ used by logistic regression for binary classification and

compute its gradient

$$\frac{\partial L}{\partial w}$$

with respect to parameters**

$$L(w) = \Sigma_{i=1}^{N}(y_i(p) + (1-y_i)log(1-p))$$

$$\frac{\partial L(w)}{\partial w} = \frac{\partial(\Sigma_{i=1}^{N}(y_i(p_i) + (1-y_i)log(1-p_i))}{\partial w})$$

so

$$\frac{\partial L(w)}{\partial w} = \Sigma_{i=1}^{N}(\frac{(y_i-p_i)\partial p_i}{p_i(1-p_i)\partial w})$$

$$p = \sigma(w^T x_i) = \frac{1}{1+e^{-w^T x_i}}$$

$$\frac{\partial L(w)}{\partial w} = \frac{dp}{dw} = \frac{x_i e^{-w^T x_i}}{(1+e^{-w^T x_i})^2}$$

$$\frac{\partial L(w)}{\partial w} = \frac{dp}{dw} = p(1-p)x_i \qquad Eq-1$$

$$\frac{\partial L(w)}{\partial w} => \Sigma_{i=1}^{N}(y_i(\frac{p(1-p)x_i}{p}) - (1-y_i)(\frac{p(1-p)x_i}{(1-p)}))$$

Substituting in equation 1 we get

$$\frac{\partial L(w)}{\partial w} => \Sigma_{i=1}^{N} x_i(y_i - p_i)$$

x.) It takes 10 minutes to train a SVM algorithm on a dataset with 100K data points.How much time do you think it might take to train the same algorithm on 1 M data points?

If you are referring to standard SVM it has O(N^3) time and O(N^2) space complexity where N is training set size using quadratic programming formulation.

$$O(n^3)$$

```
For 100K = 10^5 points =10 minutes
For 1M = 10^6 points = 10^5*10 = Data is increased by 10 times and time complexity is n^3 so time
So total time is 10*1000 = 10000 minutes  which is quite huge time
```

Question 3

A car insurance company is building a risk assessment prediction system based on the age of the driver and the type of car. Can you help them by building a decision tree using information gain as the split point evaluation measure?

```
import pandas as pd
data = {
    'Age of Driver' : [25,20,25,45,20,25],
    'Car Type' : ['Sports','Vintage','Sports','SUV','Sports','SUV'],
    'Risk' : ['L','H','L','H','H','H'],
}
df=pd.DataFrame(data)
df  # s the sample data:
```

	Age of Driver	Car Type	Risk
0	25	Sports	H

```
# References https://stackoverflow.com/questions/15450192/fastest-way-to-compute-entropy-i
# Function for calculating entropy
def entropy(label1,label2):
  tot_le=len(label1)+len(label2)
  lb1=len(label1)
  lb2=len(label2)
  H_pro=lb1/tot_le
  L_pro=lb2/tot_le
  try:
    ent=-(H_pro*math.log2(H_pro)+L_pro*math.log2(L_pro))
    return ent
  except:
    return 0
```

a. (1 mark) Entropy of the dataset is:

```
# https://medium.com/analytics-vidhya/entropy-calculation-information-gain-decision-tree-l
import math
# Entropy of dataset
H_df=df[df["Risk"]=="H"]
L_df=df[df["Risk"]=="L"]
df_ent=entropy(H_df,L_df)
print('a. Entropy of dataset is %.3f' %df_ent)
```

```
a. Entropy of dataset is 0.918
```

b. (1 mark) Calculate the information gain if we split the dataset by the following rule: Age<=22.5

```
def information_gain(sp1,sp2):
  tot_len=len(sp1)+len(sp2)
  sp1_len=len(sp1)
  sp2_len=len(sp2)
  H_df=sp1[sp1["Risk"]=="H"]
  L_df=sp1[sp1["Risk"]=="L"]
  en1=entropy(H_df,L_df)
  H_df=sp2[sp2["Risk"]=="H"]
  L_df=sp2[sp2["Risk"]=="L"]
  en2=entropy(H_df,L_df)
  frames=[sp1,sp2]
  main_df=pd.concat(frames)
  main_L=main_df[main_df["Risk"]=="H"]
  main_H=main_df[main_df["Risk"]=="L"]
  ans=entropy(main_H,main_L)-((sp1_len)/tot_len)*en1-((sp2_len)/tot_len)*en2
  return ans
```

```
# information gain for data split on age
df_age_less=df[df["Age of Driver"]<=22.5]
df_age_more=df[df["Age of Driver"]>22.5]
IG_age=information_gain(df_age_less,df_age_more)
# print(IG_age)
print('b. Information gain for data split on age is %.3f' %IG_age)
```

```
b. Information gain for data split on age is 0.252
```

```
# information gain for data split on car ty
df_car_sports=df[df["Car Type"]=="Sports"]
df_not_sports=df[df["Car Type"]!="Sports"]
IG_age=information_gain(df_car_sports,df_not_sports)
# print(IG_age)
print('c.Information gain for data split on car ty is %.3f' %IG_age)
```

```
c.Information gain for data split on car ty is 0.459
```

```
# We will take the base case where entropy is 0 or the number of variables we are left wit
H_df=df_car_sports[df_car_sports["Risk"]=="H"]
L_df=df_car_sports[df_car_sports["Risk"]=="L"]
ent_sports=entropy(H_df,L_df)

H_df=df_not_sports[df_not_sports["Risk"]=="H"]
L_df=df_not_sports[df_not_sports["Risk"]=="L"]
ent_non_sports=entropy(H_df,L_df)

# non sports car have zero entropy so we don't have to deal with it
```

c. (1 mark) Calculate the information gain if we split the dataset by the following rule: Car Type = Sports

```
print('if Car Type = Sports %.3f' %ent_sports)
print('if Car Type != Sports %.3f' %ent_non_sports)
```

```
if Car Type = Sports 0.918
if Car Type != Sports 0.000
```

Double-click (or enter) to edit

```
# splitting sports on Age of Driver basis i.e. age >=22.5 and sports car basis

df_age_less=df_car_sports[df_car_sports["Age of Driver"]<22.5]
df_age_more=df_car_sports[df_car_sports["Age of Driver"]>=22.5]
```

```
IG_sports_age=information_gain(df_age_less,df_age_more)
print(IG_sports_age)

# Here as the IG is very large and the entropy of this dataset is 0 for sports type
# splitted on basis of age . We can use the age function in our decision tree.
```

```
0.9182958340544896
```

d. (2 mark) Which of the above two rules should we use for the root-node of the decision tree? Assuming that this is the best choice, can you complete the decision tree?

```
# Decision Tree

def predict(age,car_type):
  if car_type!="Sports":
    return "H"
  else:
    if age>=22.5:
      return "L"
    else:
      return "H"
```

```
predicted_df=pd.DataFrame(df.copy())
predicted=[]
for ind in predicted_df.index:
  predicted_risk=predict(predicted_df["Age of Driver"][ind],predicted_df["Car Type"][ind])
  predicted.append(predicted_risk)
predicted_df["Predicted Risk"]=predicted
predicted_df
```

	Age of Driver	Car Type	Risk	Predicted Risk
0	25	Sports	L	L
1	20	Vintage	H	H
2	25	Sports	L	L
3	45	SUV	H	H
4	20	Sports	H	H
5	25	SUV	H	H

Colab paid products - Cancel contracts here

✓ 0s completed at 6:35 AM