Dheeraj

2020194

Assignment 2

ML REPORT

Q1) done in different pdf. name "Q1"

2. (15 points) **Section B (Scratch Implementation)**

   1. (2 marks) Create a dataset (with 10,000 points) using the circle equation

   $$(x-h)^2+(y-k)^2=r^2$$

   such that: h=0, k=0, r=1, with the label **0** and h=0, k=3, r=1, with the label **1**.

      - Write a class named dataset which takes number of points as input.
      - The class should have a function named get(add_noise=False), which should give a set of pre-defined number of points. Every call to this function returns random points, given it satisfies the conditions above. DO NOT implement this class in the main .py/.ipynb file instead create a separate utils.py and import the functions you need in main file.
      - Given, add_noise=True, as an input to the get function, it should also add Gaussian noise with mean 0 and standard deviation 0.1.

   2. (1 mark) Plot the dataset on a 2-D plot such that all the information related to the dataset i.e., x, y, and labels can be inferred from the 2d plot itself with add_noise argument set to True and False.

   3. (5 marks) Train a classifier using the Perceptron training algorithm (PTA), taught in class, on the data you just created (with and without noise) and plot the decision boundary if there exists one. Otherwise explain why not a decision boundary exists. NOTE: You have implement the PTA algorithm as a python class yourself from scratch using only numpy and python. DO NOT implement this class in the main .py/.ipynb file instead implement this in the utils.py and import the class/functions you need in main file.

Page 2

   4. (3 marks) Train another classifier using the perceptron training algorithm (PTA) on the data you just created (without noise) but with a fixed bias equal to "0" and plot the decision boundary if there exists one. Compare the results with question 2.3 and write a brief report of at least 150 words.

   5. (3 marks) Create a dataset (with 4 points) using the XOR, AND, and OR property. Plot decision boundary, if there exists one, using the PTA such that the bias is learnable, and fixed (equals to "0")

   6. (1 mark) Given a hyperplane equation and a point how would you predict which class (0 or 1) it belongs to? Also write any assumption you made, any equations you use for explanation.

- <u>Dataset class</u>

According to given question with given feature

1. With f(n) get(add_noise = false)
2. With fn() get(add-noise = true)
   i. With gaussian noise which have mean = 0 and standard deviation = 0.1

```python
class dataset:
    N_points =0
    df = pd.DataFrame()
    data0 =[]
    data1 =[]

    def helperl0_y(self,x,r):
        y1 = (r**2 -x**2)**0.5
        y2 = -(r**2 -x**2)**0.5
        return y1, y2

    def helperl1_y(self, x, h, k, r):
        y1 = (r**2 - (x-h)**2 )**0.5 + k
        y2 = -((r**2 - (x-h)**2 )**0.5) + k
        # print(y2)
        return y1,y2

    def __init__(self, N_points):
        self.N_points =N_points
        self.data0=(self.calculate_l0(self.N_points)).c

        self.data1 = self.calculate_l1(self.N_points).c
        # print(self.data0)

    def calculate_l0(self, N_points):
        h = 0
        k = 0
        r = 1
        print(r)

        data = []
        for i in range(N_points//4):
            x = random.uniform(-1,1)
            y1, y2 = self.helperl0_y(x,r)
            temp = [x, y1,0]  # adding x ,y , label
            data.append(temp)

            temp = [x, y2,0]  # adding x ,y , label
            data.append(temp)

        # print(data)
        return data
```

```python
def calculate_l1(self, N_points):
    h = 0
    k = 3
    r = 1
    i = 0
    data = []
    for i in range(N_points//4):
        x = random.uniform(-1,1)
        y1 ,y2 = self.helperl1_y(x, h, k, r)
        temp = [x,y1,1]  # adding x ,y , label = 1
        temp1 =[x,y2,1]
        data.append(temp)
        data.append(temp1)
        # print(data)
        # print("")


    return data
# get function will return  df according to question  which we will call in main
def get(self,add_nose = False):
    if(add_nose == False):
        # print("data0",self.data1)
        # print("new rand",self.data0)
        random.shuffle(self.data0)
        random.shuffle(self.data1)
        self.df = pd.DataFrame(self.data0,columns=['X','Y','Label'])
        df1 = pd.DataFrame(self.data1,columns=['X','Y','Label'])
        self.df=self.df.append(df1,ignore_index = True)
        # print("data0",self.data0)
        # print(df1.head())
    else:
        random.shuffle(self.data0)
        random.shuffle(self.data1)
        self.df = pd.DataFrame(self.data0,columns=['X','Y','Label'])
        df1 = pd.DataFrame(self.data1,columns=['X','Y','Label'])
        self.df=self.df.append(df1,ignore_index = True)
        lable = self.df.iloc[:,-1]

        # print(lable)
        mean = 0
        sigma = 0.1
        noise = pd.DataFrame(np.random.normal(mean,sigma,[len(self.df),2]),column
        # print(noise.head())
        # print("------------------")
        # print(self.df.head())
        # print("------------------")


        self.df=self.df.loc[:,self.df.columns != 'Label'].add(noise)
        self.df["Label"] = lable
        # print(self.df.head())
    return self.df
```

The Dataset class here creates a set of points (random points) which satisfied the given equation :

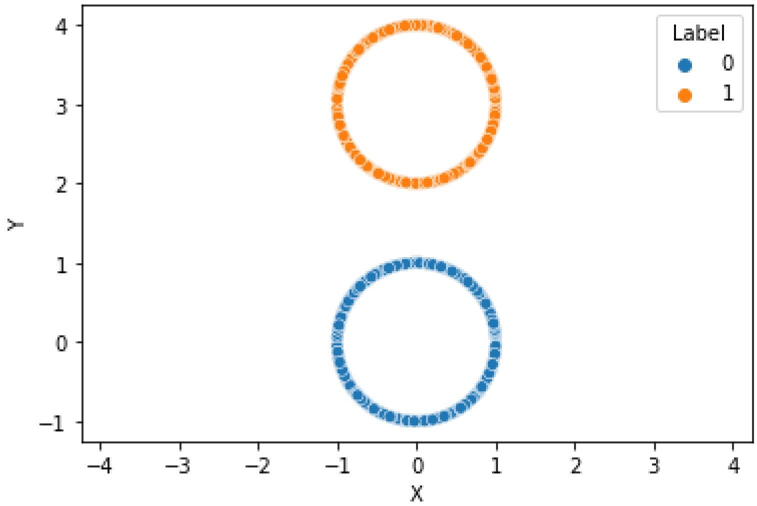
$$(x-h)^2 + (y-k)^2 = r^2$$

1) when h =0, k=0, r=1 with label 0

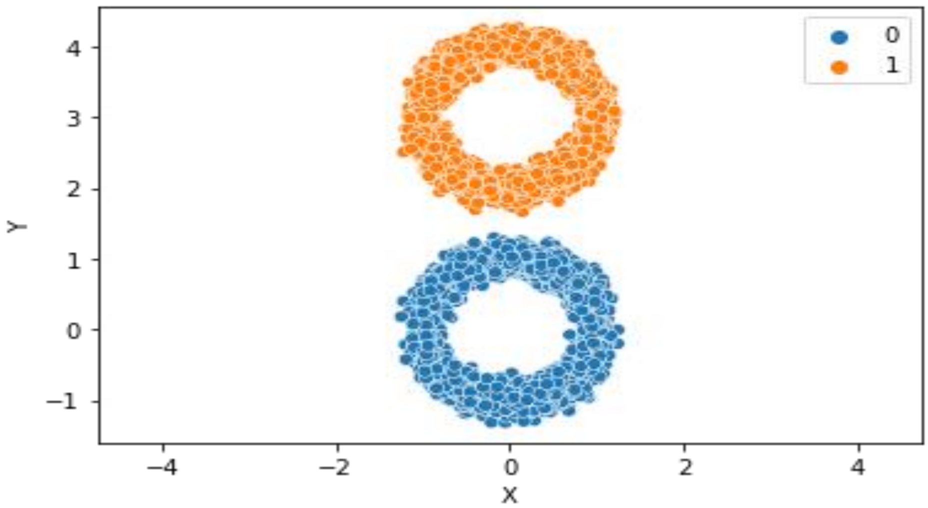
2) when h =0 , k =3, r=1 with label1

2)

Plot without noise :



Plot with noise:



3)

Code:

Here class perceptron contains all functions are required to train the model

We are calling perceptron class inside main code then pass perceptron class learning function to PTA in main which plot the graph from weights we have again from learning.

Class perceptron:

```python
class Perceptron:

    def __init__(self,X_actual,Y_actutal, learning_rate=0.01, epochs=1000):
        # declearing varibles
        self.lr = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None
        self.X_actual = X_actual
        self.Y_actutal = Y_actutal



    def learning(self, X, Y,flag=0):
        row, n_features = X.shape
        # x is here your feature matirx n*m

        # init weight parameters with zeros and bias = 0
        self.weights = np.zeros(n_features)
        self.bias = 0

        # y_ = np.array([1 if i > 0 else 0 for i in y])
        Y = np.array(Y)
        # print(Y)
        for _ in range(self.epochs):
            # taking loop i for index and value at that index
            for j in range(len(X)):
                # print(type(val), val)

                # print("----")

                # print(self.bias)
                temp_y_predic = np.dot(X[j], self.weights) + self.bias
                y_pred = self.step_func(temp_y_predic)

                #  update rule
                update = self.lr * (Y[j] - y_pred)

                # updating weights
                self.weights += update * X[j]
                #according to question if falg = 1 then bais = 0 everytime
                if(flag == 1):
                    self.bias =0
                else:
                    self.bias += update

        return self.weights, self.bias
```

Perceptron Training Algorithm:

1)Initially we have assume weights and bias $= 0$

2)Run until we find the decision boundary or for fix no. of time

For each loop

For each sample:

↳ Calculate $\hat{y} = g(f(x)) = g(w^T x + b)$

update rule 1) $\Delta w = \alpha \cdot (y_i - \hat{y}_i) \cdot x_i$

2) $\Delta b = \alpha \cdot (y_i - \hat{y}_i)$

3)update rule as mention above where alpha is learning rate

Then using update rule update weights and bais

4) update rule Explaination

y	$\hat{y}$	$y-\hat{y}$
1	1	0
1	0	1
0	0	0
0	1	-1

if both = true [hey =1] so No update $\underbrace{y-\hat{y}}_{\text{error term}} = 0$ (1)

$y-\hat{y}$: $(1-0) = 1$ so update is needed in +ve direction so update is + (2)

$y-\hat{y}$: $(0-0) = 0$ so no update again zero like (1) (3)

$y-\hat{y}$: $(0-1) = -1$ so update is needed in -ve direction (4)

so it will handel +ve & -ve case Automatically

Code for step function:

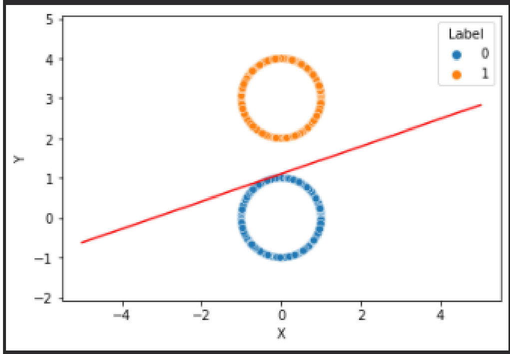
```
def step_func(self, x):
    if x>=0:
        return 1
    else:
        return 0
```

Calling in main file:

```python
# df = df.drop('index',axis=1)
percep = utils.Perceptron(df.iloc[:,:2].values , df.iloc[:,-1].values, epochs= 50)
PTA(percep,df)
```

✓ 4.1s

```
[-0.0093742   0.02714905]
```

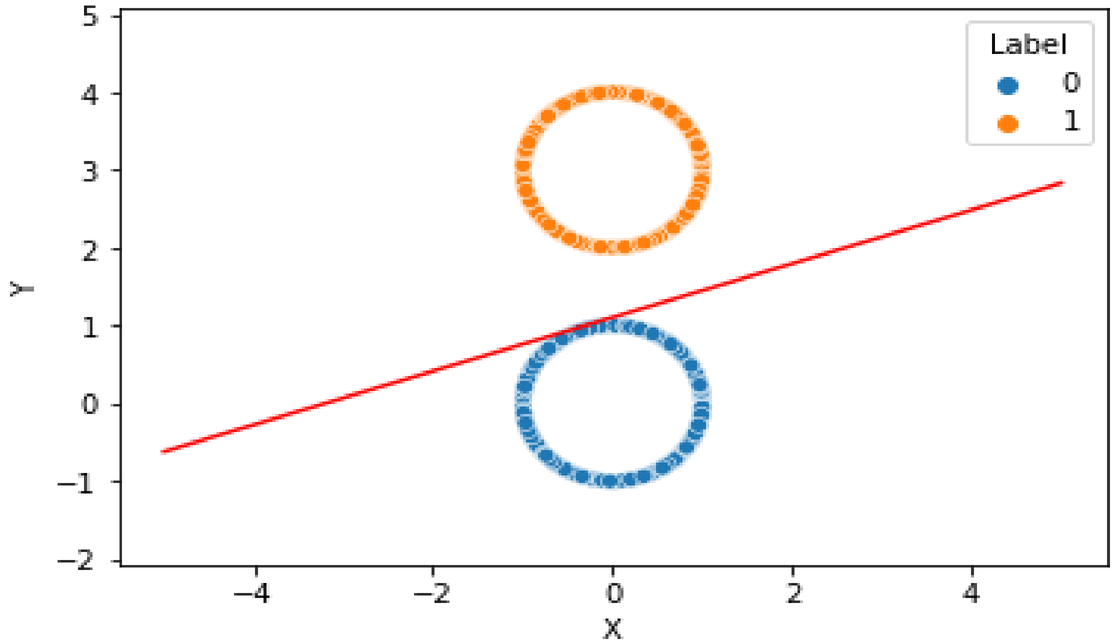


PTA function code:

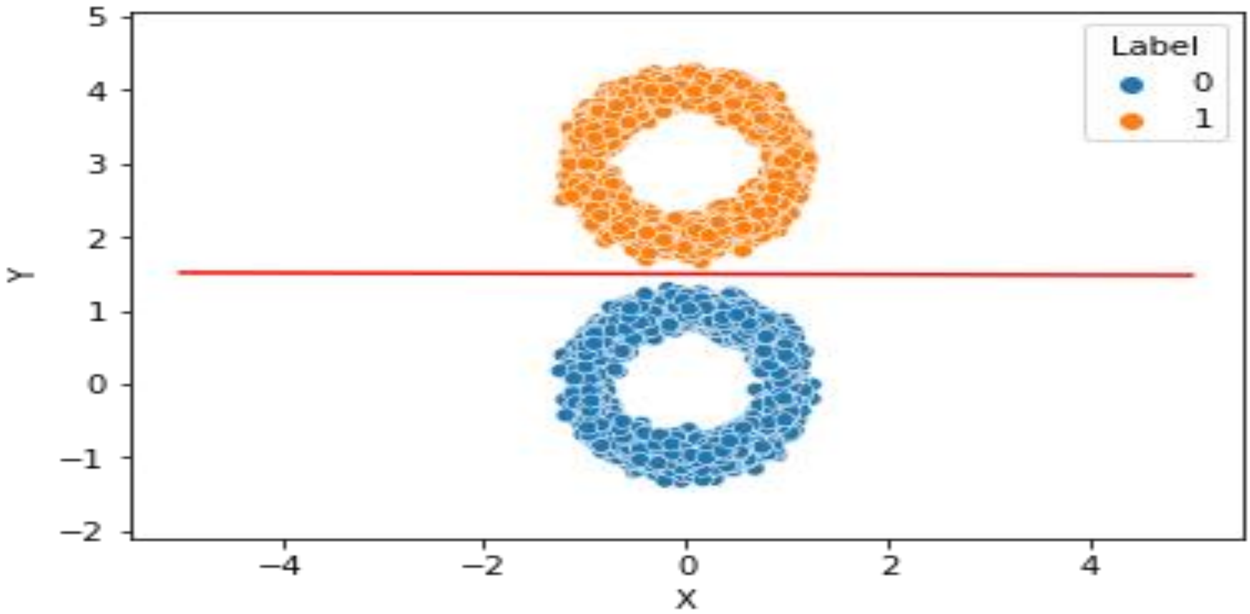
```python
def PTA (p,data,flag =0):

        weight, biased = p.learning(p.X_actual,p.Y_actutal,flag)
        print(weight)
        # weights[0]*x + weight[1]*y + b = 0;

        x = np.linspace(-5,5,100)

        y = (-weight[0]*x-biased)/weight[1]

        plt.plot(x,y,'-r')
        sea.scatterplot(x='X',y='Y',data = data, hue = 'Label')
        plt.axis("equal")

        plt.show()
```

<u>Plot of Decision Boundary</u>

1) <u>with noise</u>



2) with Noise



4)

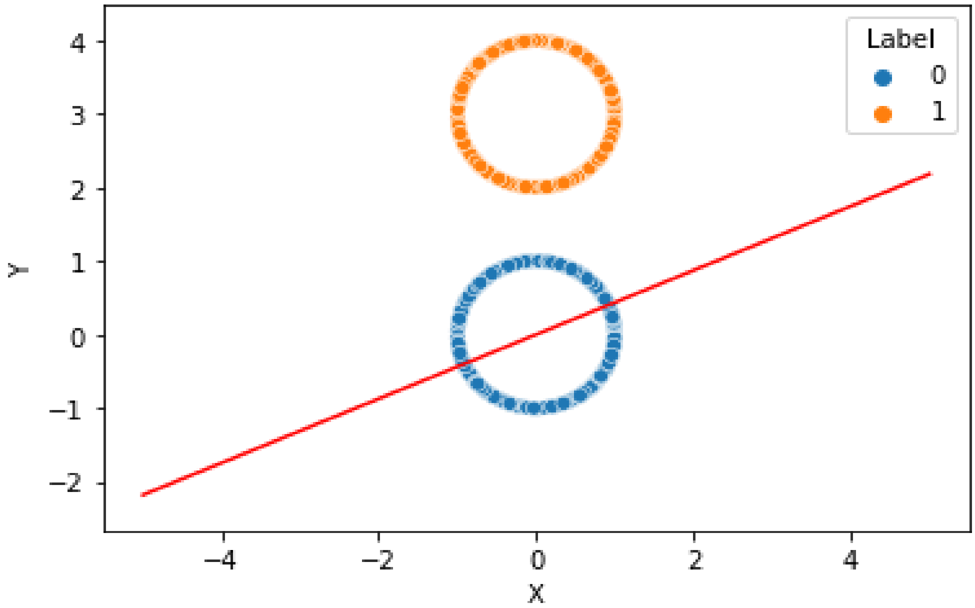
Algorithm Code and algorithm both are same as above path

According to question bias need to be zero hence passing our flag parameter =1

Without noise: No decision boundary exist

```
percep = utils.Perceptron(df.iloc[:,:2].values , df.iloc[:,-1].values, epochs= 52)
PTA(percep,df,1)
```

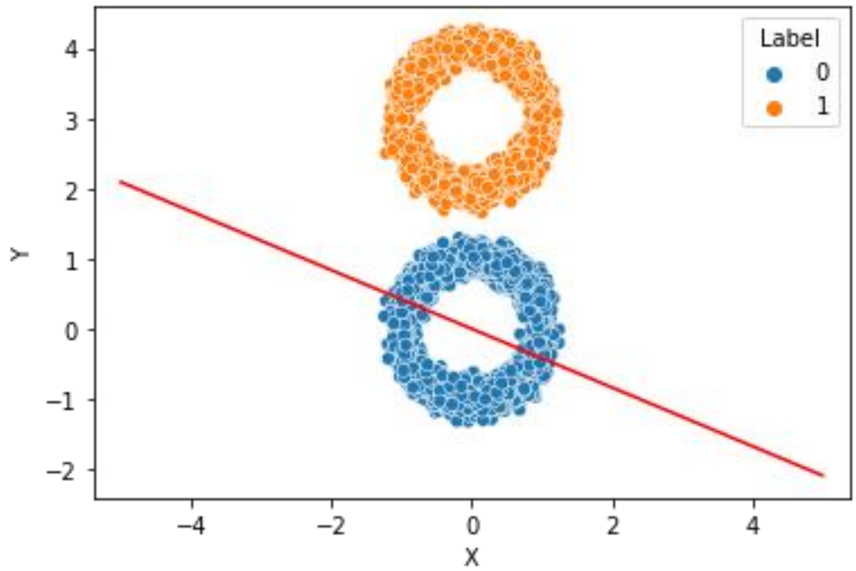
✓ 4.7s



With noise : No decision boundary exist

```
percep = utils.Perceptron(df1.iloc[:,:2].values , df1.iloc[:,-1].values, epochs= 52)
PTA(percep,df1,1)
```

✓ 4.1s



Comparison of the result of 2.3 and 2.4

- In question 2.3 we were able to find the decision boundary for both with and without noise datasets.
- Whereas in 2.4, no decision boundary was found for both cases
- In 2.3, without noise weights were w1 = -0.010367, w2 = 0.01674 bias = -0.03
  With noise w1 =0.00323697 ,w2 = 0.02512334 , bias = -0.04
- In 2.4 only change is Bias was equal to 0 in every case

Bias is equivalent to the intercept in a linear equation. It is an extra parameter in the Neural Network that is used to change the output in addition to the weighted sum of the neuron's inputs. As a result, Bias is a constant that helps the model fit best for the supplied data.

Due to the lack of Bias, the model will train over points only passing through the origin, which is inconsistent with real-world circumstances. The model will also become more flexible when Bias is introduced.

When Bias becomes 0, the Decision boundary w0+w1.x1+w2.x2=0 takes the form of y=m.x, as w0=0.

So, DB will now pass through the origin as per the St. line eq. of y=mx.

So, hence it cannot act as a classifier for this two-circle data.

5)

Dataset for XOR

```
#making points for data x and y
x =[0,0,1,1]
y =[0,1,0,1]
```

0.6s

OR_database

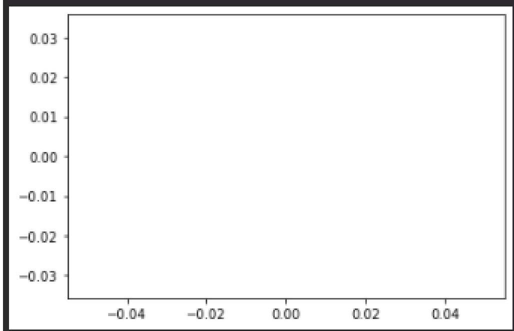
```
xor_r = [0,1,1,0]
df_xor = pd.DataFrame(list(zip(x,y,xor_r)) , columns=['X','Y','Label'])
```

Code + graph:

```
percep_xor = utils.Perceptron(df_xor.iloc[:,:2].values , df_xor.iloc[:,-1].values, epochs= 200)
PTA(percep_xor,0,df_xor)
```

0.3s

```
Truth value of a DataFrame is ambiguous.
```



Dataset for And

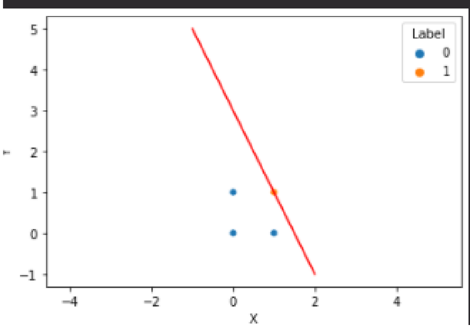
X and y values are same as in above part XOR

Code + graph:

```
percep_and = utils.Perceptron(df_and.iloc[:,:2].values , df_and.iloc[:,-1].values, epochs= 200)
PTA(percep_and,df_and,0)
```

0.1s

```
0.02 0.01]
```



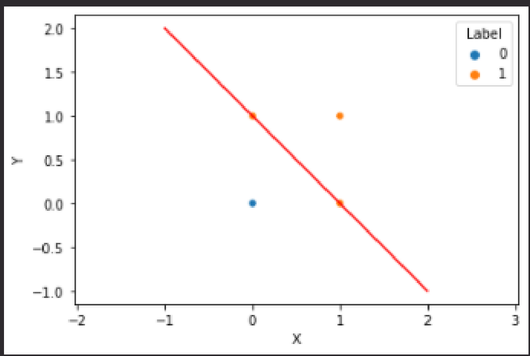
Dataset for OR

OR gate

```
or_r = [0,1,1,1]
df_or = pd.DataFrame(list(zip(x,y,or_r)) , columns=['X','Y','Label'])
percep_and = utils.Perceptron(df_or.iloc[:,:2].values , df_or.iloc[:,-1].values, epochs= 1000)
PTA(percep_and,df_or)
```

0.2s

```
[0.01 0.01]
```



Result:

For XOR : NO decision boundary exists for both unfixed Bias and fixed Bias

For AND : Decision boundary exist for unfixed Bias but does not exist for fixed Bias

For OR: Decision boundary exist for unfixed Bias but does not exist for fixed Bias

6)if we add points in the line equations if we get positive it is class 1 other wise it is class zero.

SECTION _C

3. (15 points) **Section C (Algorithm implementation using packages)**

In this question, you are expected to understand and run Random Forests and various boosting algorithms. You can use sklearn implementation of decision tree and Adaboost but you need to perform ensembling on your own.

Dataset: **Bitcoin Heist Ransomware Address Dataset**
Target Variable: Label
You will have to handle null values in the data. Split the data into a training, validation, and testing set (70:15:15 ratio) using the custom-designed train test split method. Use the same training set for training the following models. (You can not use sklearn for splitting the dataset.)

(a) (5 marks) Train a decision tree using both the Gini index and the Entropy by changing the max-depth [4, 8, 10, 15, 20]. Don't change any of the other default values of the classifier. In the following model, use the criteria which give better accuracy on the test set with the chosen depth

(b) (5 marks) Ensembling is a method to combine multiple not-so-good models to get a better performing model. Create 100 different decision stumps (max depth 3). For each stump, train it on randomly selected 50% of the training data, i.e., select data for each stump separately. Now, predict the test samples' labels by taking a majority vote of the output of the stumps. How is the performance affected as compared to parts (a)

(c) (5 marks) Another popular boosting technique is Adaboost. Use the sklearn Adaboost algorithm on the above dataset and report the testing accuracy. Use the Decision tree as the base estimator and with a number of estimators as [4, 8, 10, 15, 20]. Compare RF and Adaboost results.

Data pre-processing:

checking for null value:

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2916697 entries, 0 to 2916696
Data columns (total 10 columns):
 #   Column     Dtype  
---  ------     -----  
 0   address    object 
 1   year       int64  
 2   day        int64  
 3   length     int64  
 4   weight     float64
 5   count      int64  
 6   looped     int64  
 7   neighbors  int64  
 8   income     float64
 9   label      object 
dtypes: float64(2), int64(6), object(2)
memory usage: 222.5+ MB
```

checking for null vlaue any

```
[ ]  print(df.isnull().any().any())
     print(df.isnull().sum().sum())
     # no null vlaue found

False
0
```

No null value found.

Data pre-processing:

```
[5] lE = LabelEncoder()
    df['label'] = lE.fit_transform(df['label'])
    df['address'] = lE.fit_transform(df['address'])
    df = df.sample(frac=1)
    #changing label and address type from string to int
```

Split the data into a training, validation, and testing set (70:15:15 ratio)

```
[ ] # Split the data into a training, validation,
    # and testing set (70:15:15 ratio)
    # total range 2916697
    # Now dividing 2916697 in 70:15:15 ratio

    def split_data(df):
      temp = df.to_numpy()
      X = temp[:,:-1]
      Y = temp[:,- 1]
      Y = temp[:,- 1].reshape(Y.shape[0], 1)

      x_train = X[: 2041689, :]
      x_test = X[2041689: 2479193, :]
      x_val = X[2479193: , :]
      y_train = Y[: 2041689, :]
      y_test = Y[2041689: 2479193, :]
      y_val = Y[2479193: , :]

      return x_train, x_test, x_val,y_train,y_test,y_val
```

```
[ ] x_train ,x_test,x_val,y_train ,y_test,y_val = split_data(df)
```

A) Decision Tree

Code:

```
[ ] def Decision_tree_Train(x_train,x_test,x_val,y_train,y_test,y_val,type):
      for i in [4, 8, 10, 15, 20]:
        tree = DecisionTreeClassifier(criterion=type, max_depth=i)
        y_pred = tree.fit(x_train, y_train).predict(x_test)
        accu = accuracy_score(y_test,y_pred, normalize = True)
        print(type," for depth-",i,"accuracy = ",accu)
```

```
[ ] Decision_tree_Train(x_train ,x_test,x_val,y_train ,y_test,y_val,"gini")
    #gini with depth 15 is giving us the best accuracy.
    #0.9882903927735518

    gini  for depth- 4 accuracy =  0.9858401294616735
    gini  for depth- 8 accuracy =  0.9865532657987127
    gini  for depth- 10 accuracy =  0.9876092561439438
    gini  for depth- 15 accuracy =  0.9888092451726156
    gini  for depth- 20 accuracy =  0.987645827238151
```

```
[ ] Decision_tree_Train(x_train ,x_test,x_val,y_train ,y_test,y_val,"entropy")
    #entrophy with depth 15 is giving us the best accuracy.
    #0.988877815974254

    entropy  for depth- 4 accuracy =  0.9857715586600351
    entropy  for depth- 8 accuracy =  0.9861486980690463
    entropy  for depth- 10 accuracy =  0.9874904000877707
    entropy  for depth- 15 accuracy =  0.9889143870684611
    entropy  for depth- 20 accuracy =  0.9875658279695728
```

Best accuracy for both GINI index and Entropy is at max depth = 15 with the value of 0.988809 and 0.988914 respectively

Result: For the max depth 10,15,20 entropy performance is better than GINI,

B)

Random forest is a method to combine multiple not-so-good models to get a better performing model.

According to given question:

CODE:

```python
def train_train(x_train,y_train,x_test,y_test, typ, number=100):
  store_tree = []
  data = np.concatenate([x_train,y_train], axis=1)
  # making list of 100 DT according to need of question and stroing
  #inside list of name store_tree
  for i in range(number):
    store_tree.append(DecisionTreeClassifier(criterion=typ, max_depth=3))

  for i in store_tree:
    # randomly select 50% data
    # limit have 50 percent mark of total size
    limit = int(data.shape[0] * 0.5)
    np.random.shuffle(data)
    x = data[:,:-1]
    y = data[:,-1]

    x_limit = x[:limit,:]
    y_limit = y[:limit]

    i=i.fit(x_limit,y_limit)

  # now finding acuracy
  y_pred =[]
  for tree in store_tree:
    y_pred.append(tree.predict(x_test))
  # changing y_pred into nparray to use in accuracy model
  y_pred = np.array(y_pred)
  temp = np.array(stats.mode(y_pred))
  f_y_pred = temp[0,0,:]

  accu = accuracy_score(y_test,f_y_pred, normalize = True)
  print(typ," accuracy = ",accu)
```

```
[ ] train_train(x_train,y_train,x_test,y_test, "entropy", number=100)

    entropy  accuracy =  0.9855041325336454
```

<u>Result</u> The performance of Random Forest is better than the Decision Tree as we can see at depth 3 random forest as similar accuracy that of Decision Tree at depth 15.

C)

Code:

Part-c

```
[9]  from sklearn.ensemble import AdaBoostClassifier
```

```
[10] def ada_b(x_train,x_test,y_train,y_test,typ):
       accu = []
       y_test = y_test.reshape(y_test.shape[0])
       y_train = y_train.reshape(y_train.shape[0])
       for i in [4, 8, 10, 15, 20]:
         clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=15  , criterion=typ), n_estimators=i)
         clf.fit(x_train, y_train)

         y_pred = clf.predict(x_test)
         accu.append(accuracy_score(y_test, y_pred))
       return accu
```

```
temp = ada_b(x_train,x_test,y_train,y_test,"entropy");
```

```
[12] print(temp)

     [0.9886835320362786, 0.9863589818607372, 0.985901843183148, 0.9869921189291984, 0.9880321094207138]
```

Result:

Adaboost is giving higher accuracy than Random Forest at high number of estimators and max-depth where as Random forest is giving higher accuracy even for low max-width