Machine Learning (CSE343/ECE343)

Assignment - 2
Report

Name: Khushdev Pandit
Roll no.: 2020211

Section - A

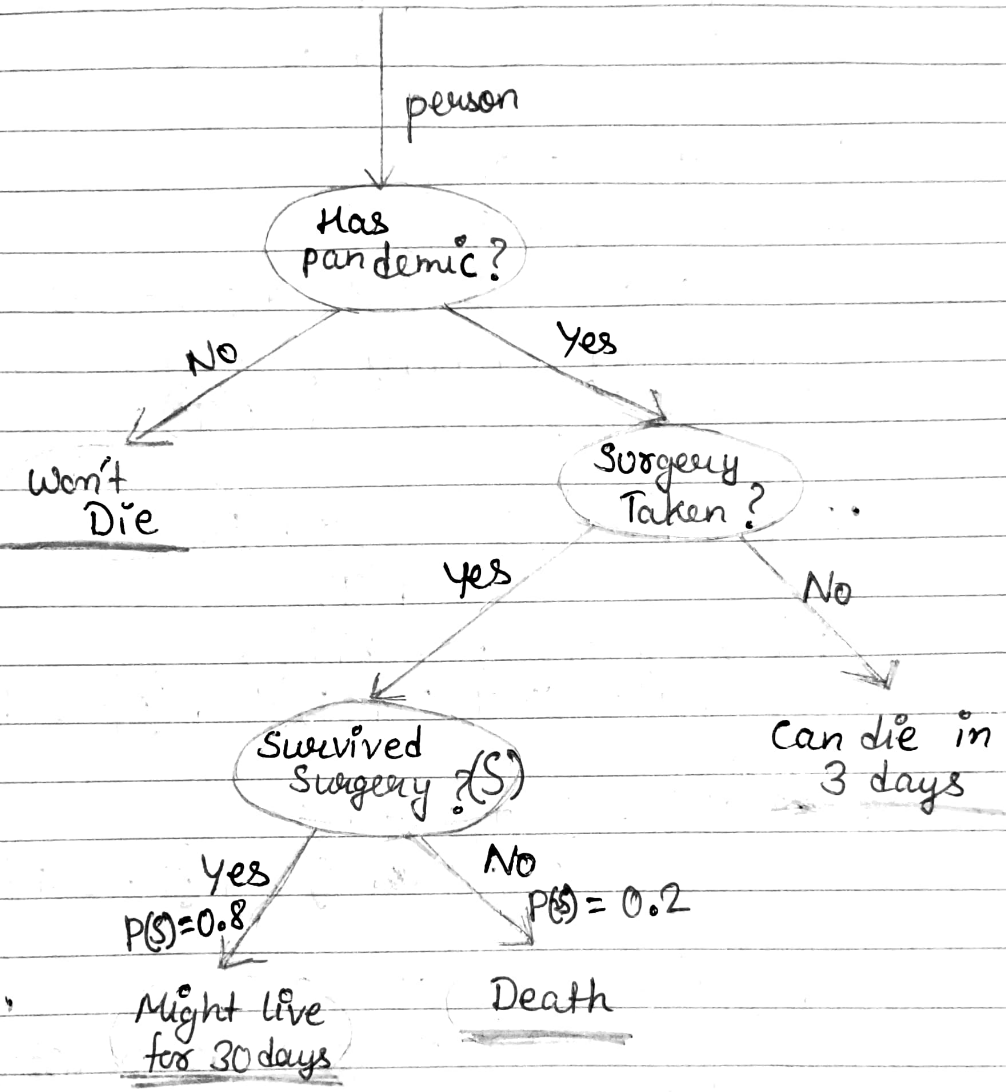
(Theoretical)

Q1

ML – Assignment – 2

Section – A

Answer 1(a)



This is the above Decision Tree for predicting whether the person will survive or not based on whether he has pandemic or not.

$P(S) = 0.8$

$P(\sim S) = 0.2$

Date / /
DELTA Page

Answer 1(c) S → patient surviving the surgery
~S → patient not surviving the surgery
T → test result is +ve
~T → test result is -ve

$\therefore P(T|S) = 0.95$
$\therefore P(T|\sim S) = 0.05$
$\therefore P(S) = 0.8 \Rightarrow P(\sim S) = 0.2$

→ We need to find Probability of having successful surgery if the test is positive $= P(S|T) = ?$

$\because P(S|T) = \dfrac{P(T|S) \times P(S)}{P(T)}$

By law of total probability,
$P(T) = P(T|S)\,P(S) + P(T|\sim S)\,P(\sim S)$

$\therefore P(S|T) = \dfrac{P(T|S) \times P(S)}{P(T|S).P(S) + P(T|\sim S).P(\sim S)}$

$\therefore P(S|T) = \dfrac{0.95 \times 0.8}{0.95 \times 0.8 + 0.05 \times 0.2}$

$\therefore P(S|T) = 0.987$

$\therefore$ Probability of having successful surgery if the test is positive is $= P(S|T) = 0.987$.

Answer 1(d) Yes, the surgery should be performed if the result of test is positive.
→ This is because the probability of having the successful surgery if the test is positive $(= P(S|T) = 0.987)$ is very high & close to 1.
→ Almost 98.7% of the positive test would result in successful surgery.

→ The value of True positive rate is very high (from part-c).

→ So, surgery should be performed.

Answer 1(b) → $L(x)$ → patient's living function for living x days.

$$\left.\begin{aligned} L(30) &= 1 \\ L(0) &= 0 \end{aligned}\right\}$$

∴ Minimum value of patient's utility for living three days $= L(3)\big|_{min} = \frac{1}{30} \times 3 = 0.1$

∴ When the surgery is performed, the Living function can be uniformly distributed over 30 days, with the utility value for living each day being '$\frac{1}{30}$'.

So, for 3 days, it would be '$3 \times \frac{1}{30}$'.

This would be the case of min. value of $L(3)$.

∴ $\cancel{L(3)\big|_{min} = 0.1}$

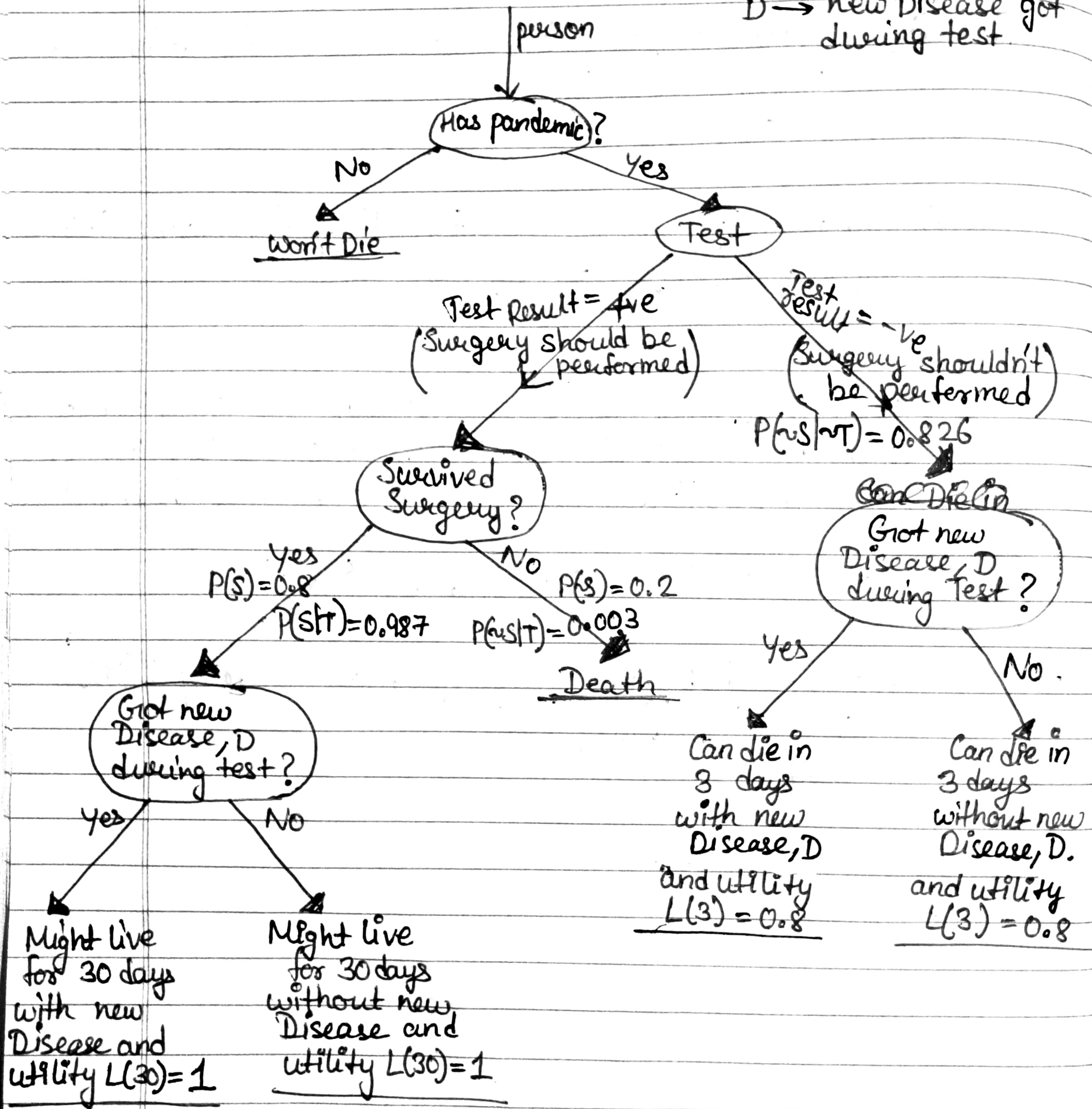
∴ $\min L(3) = 0.1$

Answer 1(e) → Following the same notations as in (c),

$P(S|T) = 0.987$, $P(\sim S|T) = 0.003$

$P(\sim S|\sim T) = 0.826$, $P(S|\sim T) = 0.173$

D → new Disease got during test.



→ When the test is +ve, surgery should be performed. as there is high chance that surgery will be successful, i.e, $P(S|T) = 0.987$

→ When the test is -ve, surgery should not be performed as there is high ~~ca~~ chance that surgery will be unsucessfully and person will die due to surgery.
i.e, $P(\sim S | \sim T) = 0.826$

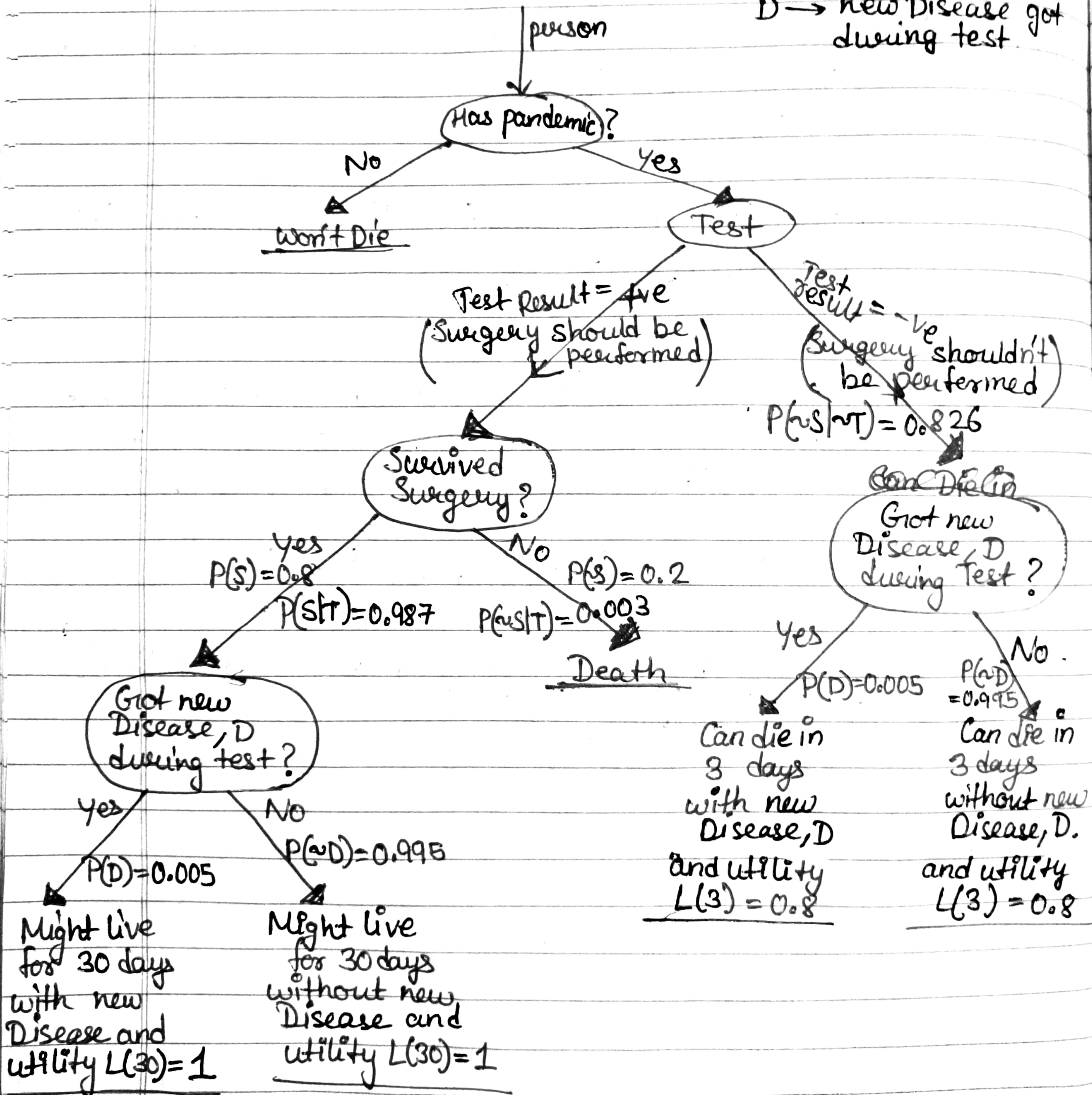
Answer 1(f) → The test should be conducted prior to operation, because the probability of contracting the new Disease during the test is $P(D)$ = 0.005, which is very low. So, YES! test should be conducted.

of part (e)
→ The Decision tree will be same as before just we have to add some Probability values.

$\therefore$ P(Contracting new Diseases when Test is performed) = 0.005.

$P(D) = 0.005$

D → new Disease got during test



Section - B

(Scratch Implementation)

Q2

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from utils import *
```

Q2 Part-1

Creating Dataset

Dataset without any noise

In [ ]:

```
df = CircleDataset(10000).get()
```

Dataset with Standara Normal noise

In [ ]:

```
df_noise = CircleDataset(10000).get(add_noise=True)
```

Check whether dataset are equal?

In [ ]:

```
(df_noise == df).sum()
```

Out[ ]:

```
x              0
y              0
center_x   10000
center_y    5040
radius     10000
label       5040
dtype: int64
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   x         10000 non-null  float64
 1   y         10000 non-null  float64
 2   center_x  10000 non-null  float64
 3   center_y  10000 non-null  float64
 4   radius    10000 non-null  float64
 5   label     10000 non-null  float64
dtypes: float64(6)
memory usage: 468.9 KB
```

In [ ]:

```
df_noise.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   x         10000 non-null  float64
 1   y         10000 non-null  float64
 2   center_x  10000 non-null  float64
 3   center_y  10000 non-null  float64
 4   radius    10000 non-null  float64
 5   label     10000 non-null  float64
dtypes: float64(6)
memory usage: 468.9 KB
```

Checking whether the Dataset satisfies Equation of circle

In [ ]:

```
(np.abs((df['x']-df['center_x'])**2 + (df['y']-df['center_y'])**2 - 1) > 1e-15).sum()
```

Out[ ]:

```
0
```

In [ ]:

```
df.head()
```

Out[ ]:

	x	y	center_x	center_y	radius	label
0	-0.979163	0.203076	0.0	0.0	1.0	0.0
1	-0.941643	0.336614	0.0	0.0	1.0	0.0
2	0.522993	-0.852337	0.0	0.0	1.0	0.0
3	-0.985146	0.171718	0.0	0.0	1.0	0.0
4	-0.634830	-0.772652	0.0	0.0	1.0	0.0

Q2 Part-2

Plotting the circle Dataset

In [ ]:

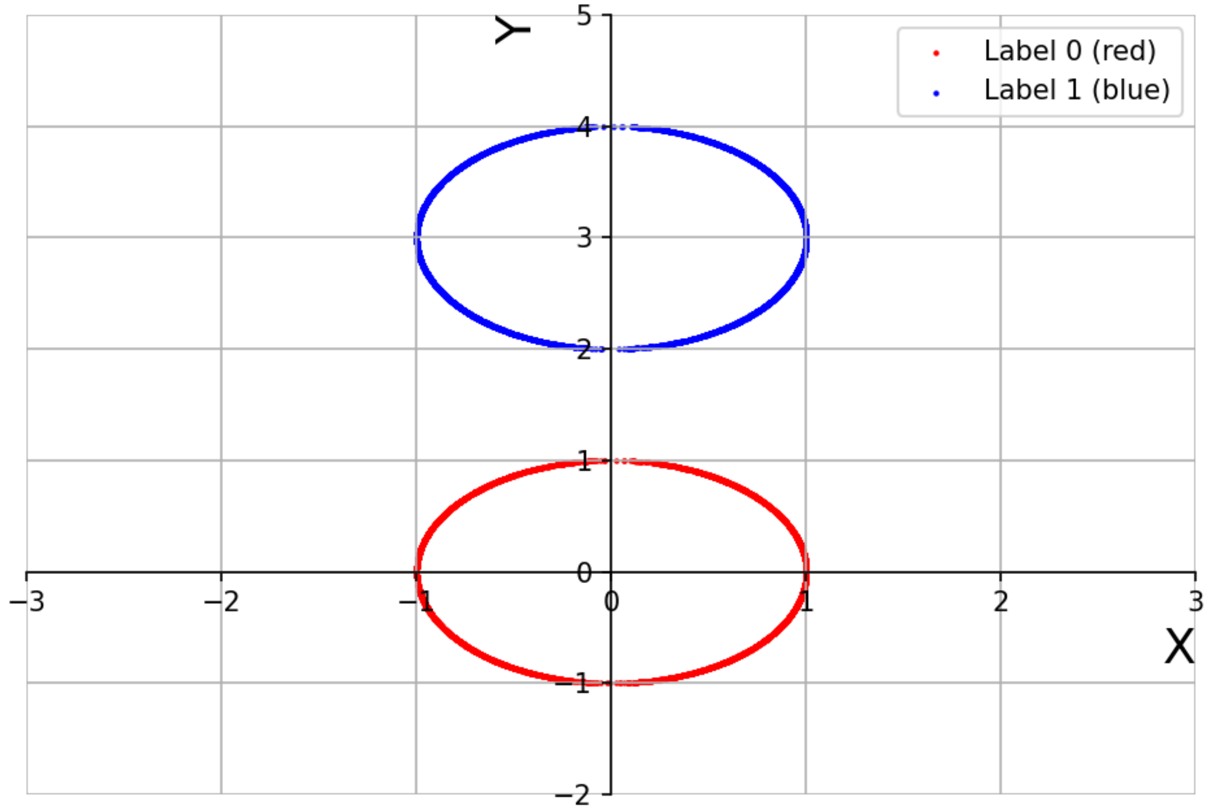
```
def plot_circle_dataset(data, xlim, ylim, plot_decision_boundary=False, model_weights=[]):
    fig = plt.figure(dpi=150)
    axes = fig.add_axes([0,0,1,1])

    df_label0 = data[data['label'] == 0]
    df_label1 = data[data['label'] == 1]

    axes.scatter(df_label0['x'], df_label0['y'], c='r', marker='.', label='Label 0 (red)', s=5);
    axes.scatter(df_label1['x'], df_label1['y'], c='b', marker='.', label='Label 1 (blue)', s=5);

    if plot_decision_boundary:
        x = np.random.uniform(low=xlim[0],high=xlim[1],size=1000)
        y = (-model_weights[2] - model_weights[0]*x) / model_weights[1]
        axes.plot(x,y,color='k',label='Decision Boundary')

    axes.grid(True)
    axes.legend()
```

```
    axes.spines['bottom'].set_position(('data',0)) # set position of x spine to x=0
    axes.spines['left'].set_position(('data',0))   # set position of y spine to y=0
    axes.spines['right'].set_color('none')
    axes.spines['top'].set_color('none')
    axes.set_xlim(xlim)
    axes.set_ylim(ylim)
    axes.set_xlabel('X', loc='right', fontsize=18)
    axes.set_ylabel('Y', loc='top', fontsize=18)
```

Plot of Circle Dataset without noise (with "add_noise = False")

In [ ]:

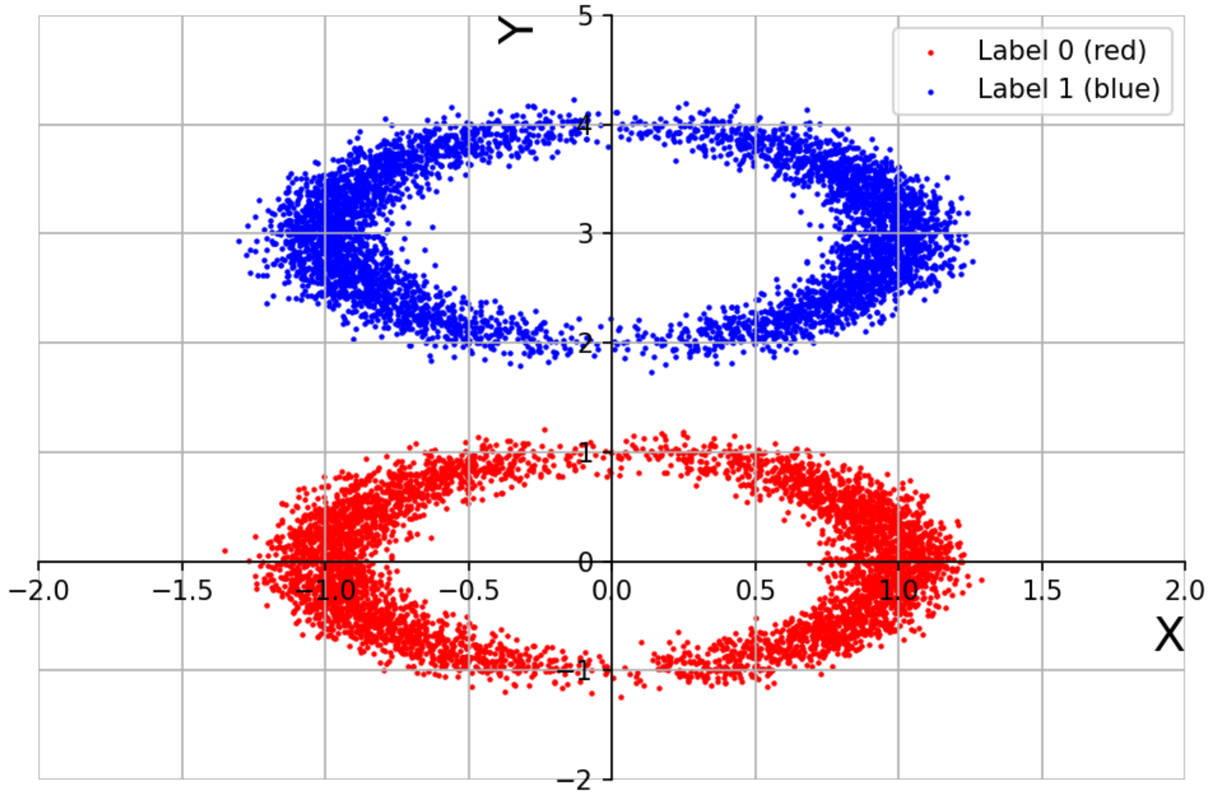
```
plot_circle_dataset(df, [-3,3], [-2,5])
```



Plot of Circle Dataset with noise (with "add_noise = True")

In [ ]:

```
plot_circle_dataset(df_noise, [-2,2], [-2,5])
```



Q2 Part-3

Applying Perceptron on the Dataset

In [ ]:

```
def apply_preceptron_on_circle_data(df_set: pd.DataFrame, partition_size, with_bias=True):
    x_train, y_train, x_test, y_test = split_circle_data_into_train_test(df_set, partition_size, with_bias)

    perceptron_model = Perceptron()
    perceptron_model.fit(x_train, y_train)

    y_pred = perceptron_model.predict(x_test)

    accuracy = perceptron_model.accuracy(x_test, y_test)
    # print(perceptron_model.weights)

    print(f'Accuracy in Prediction of Perceptron Learning Algorithm (on 20% Testing Set):', accuracy)
    plot_circle_dataset(df_set, [-3,3], [-2,5], plot_decision_boundary=True,
            model_weights=perceptron_model.weights)
```

Applying Perceptron on Datset without Noise and plotting Decision Boundary

with "add_noise = False"

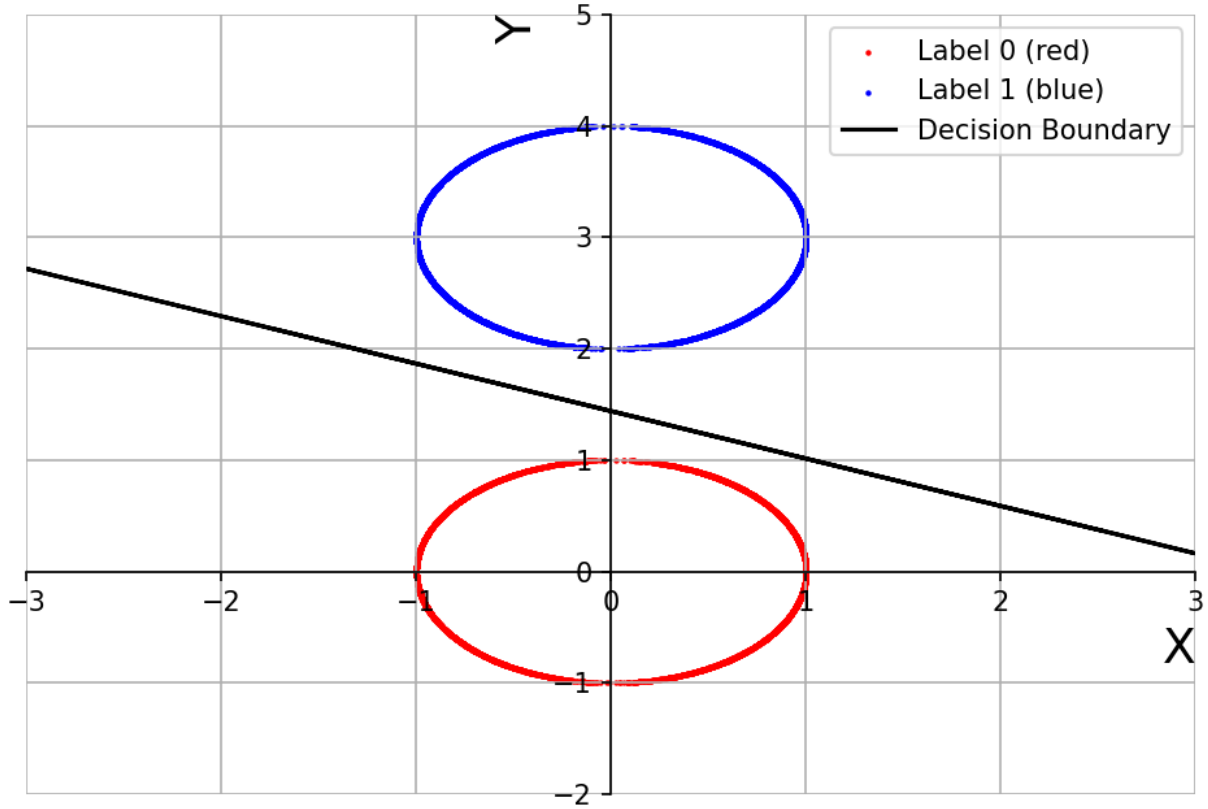
We can see that, linear Decision boundary is able to perfectly classify the two circular regions in DATASET WITHOUT NOISE.

As the two circular regions are linearly separable, so the Perceptron Training algorithm will converge according to Perceptron Convergence theorem. So, we can easily find a linear Decision boundary that can separate two circular regions.

In [ ]:

```
apply_preceptron_on_circle_data(df, partition_size=[80, 20])
```

Accuracy in Prediction of Perceptron Learning Algorithm (on 20% Testing Set): 1.0



Applying Perceptron on Datset with Noise and plotting Decision Boundary

with "add_noise = True"

We can see that, linear Decision boundary is able to perfectly classify the two circular regions in DATASET WITH NOISE.

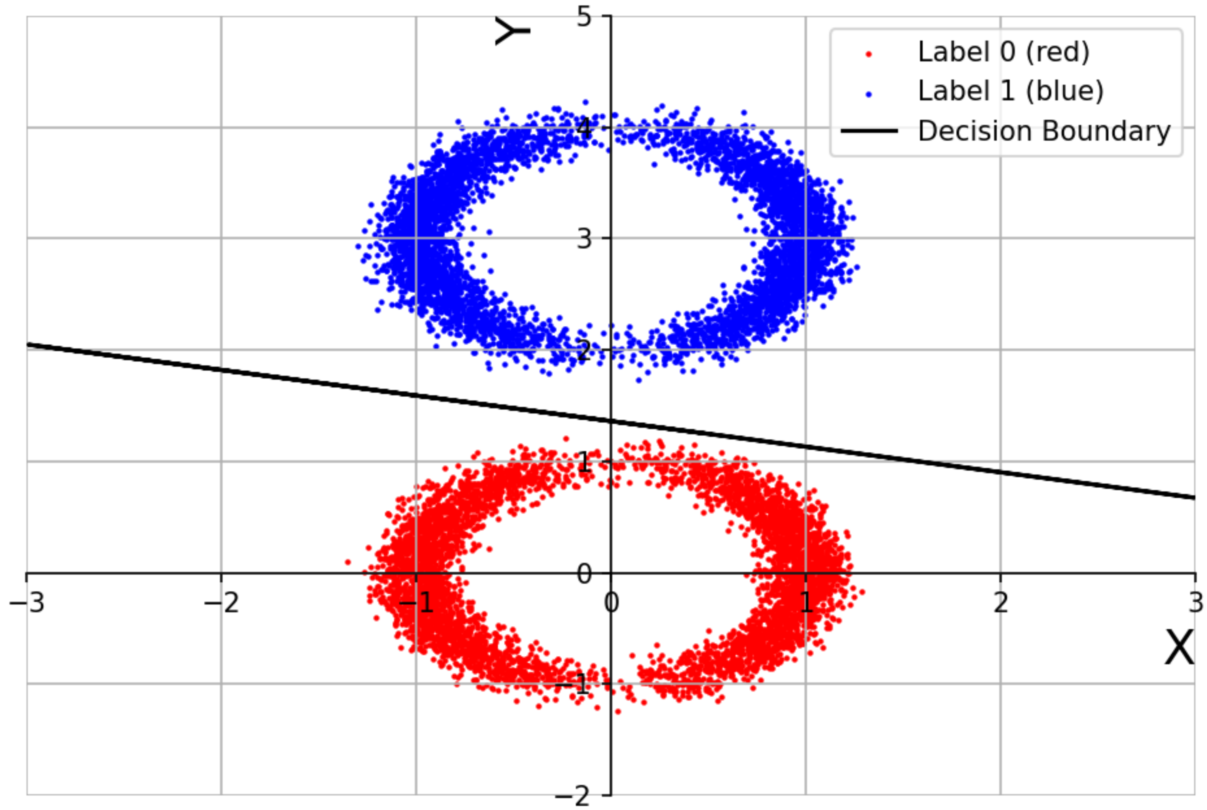
Since the random noise has standard deivation of 0.1 which is low, and still doesn't affects the linear separability of two circular regions.

Because even after adding the noise, the two circular regions are still linearly separable. So, the Perceptron Training algorithm will converge according to Perceptron Convergence theorem. So, we can easily find a linear Decision boundary that can separate two circular regions.

In [ ]:

```
apply_preceptron_on_circle_data(df_noise, partition_size=[80, 20])
```

Accuracy in Prediction of Perceptron Learning Algorithm (on 20% Testing Set): 1.0



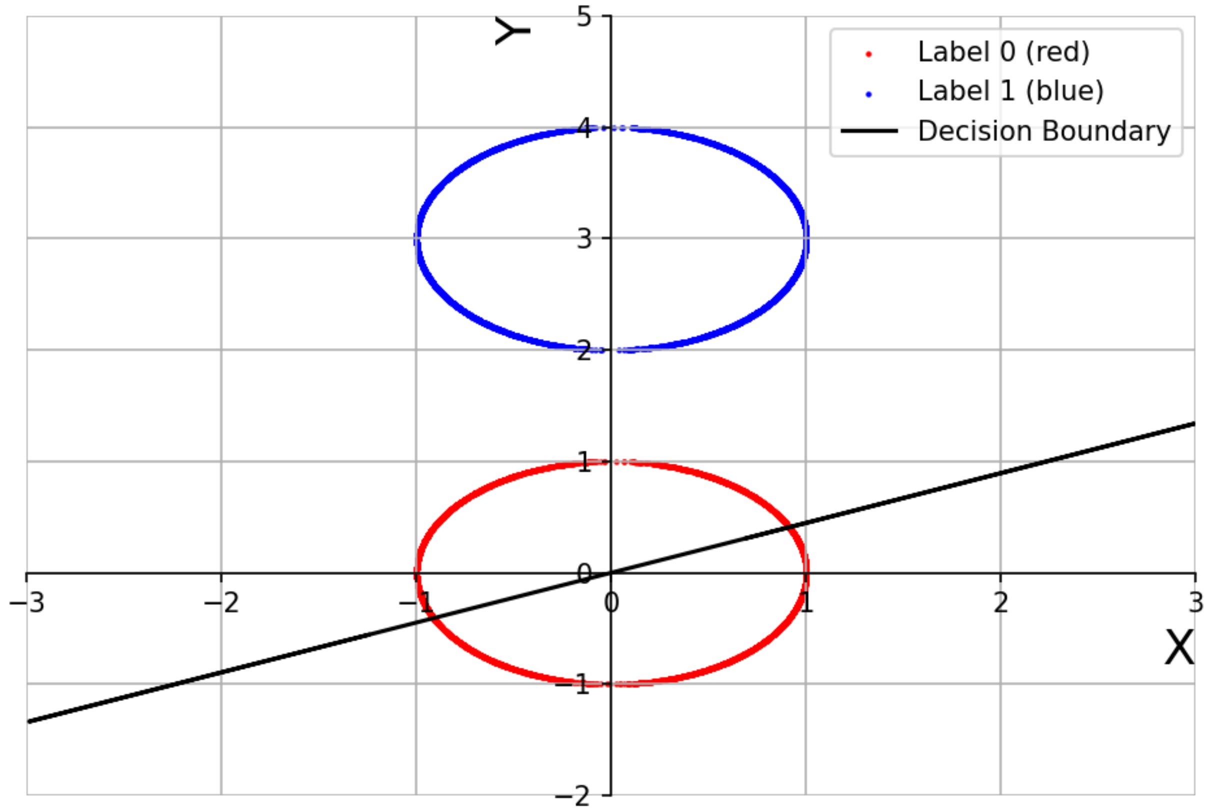
Decision Boundary exists in the case of Perceptron learning algorithm implemented, on both Dataset (with and without Noise Dataset)

Q2 Part-4

In [ ]:

```
apply_preceptron_on_circle_data(df, with_bias=False, partition_size=[80, 20])
```

Accuracy in Prediction of Perceptron Learning Algorithm (on 20% Testing Set): 0.7435



Explaination for why Decision Boundary doesn't exists in Circle dataset, when PTA is implemented without Bias.

We can clearly see that above Decision Boundary can not classify the two circular regions (in Circle Dataset without noise), when we set a fixed bias equal to "0".

This is beacuse the equation of our Decision boundary is "w0 + w1.x1 + w2.x2 + ... + wn.xn = 0".

In our case of Circle Dataset, we have only two features.

So, the equation of Decision Boundary reduces to "w0 + w1.x1 + w2.x2 = 0".

As the Input Bias is fixed to zero '0', this implies "w0 = 0".

So, the equation of Decision Boundary further reduces to "w1.x1 + w2.x2 = 0".

This Decision boundary can be represented in the form of "y = mx", once we know the values of models' weights by PTA. This is a equation of straight line passign through the origin.

So, when we apply apply PTA without Bias we get a Decision Boundary passing through the origin.

This Decision Boundary will just rotate thorugh the origin, whenever the values of models' weight (w1 & w2) are changed in the Perceptron Training Algorithm. So, a Decision Boundary rotating through the origin can not separate the two circular regions, as shown in the above the Plot of Decision Boundary.

By look at the above Dataset of two circular regions, we see that it is not separable by a linear Decision Boundary (st. line) passing through the origin. So, no Decision boundary exits when we train a model using PTA with fixed bias '0'.

Since, Bias acts as the model parameter which can be tuned to make the models' performance on training data accurate. Running PTA on Data without Bias leads to poor performance of Perceptron model.

Q2 Part-5

In [ ]:

```
bit_df = pd.DataFrame(np.array([[0, 0],
                               [0, 1],
                               [1, 0],
                               [1, 1]], dtype=np.int64), columns=['A', 'B'])
```

AND Dataset

In [ ]:

```
AND_df = bit_df.copy()
AND_df['AND'] = AND_df['A'] & AND_df['B']
AND_df
```

Out[ ]:

	A	B	AND
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

OR Dataset

In [ ]:

```
OR_df = bit_df.copy()
OR_df['OR'] = OR_df['A'] | OR_df['B']
OR_df
```

Out[ ]:

	A	B	OR
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

XOR Dataset

In [ ]:

```
XOR_df = bit_df.copy()
XOR_df['XOR'] = XOR_df['A'] ^ XOR_df['B']
XOR_df
```

Out[ ]:

	A	B	XOR
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

Plotting the AND, OR, XOR Dataset and their repective Decision Boundary

In [ ]:

```
def plot_bit_dataset(data:pd.DataFrame, model_weights, with_bias):
    fig = plt.figure(dpi=100)
    axes = fig.add_axes([0,0,1,1])

    df_bit_output_0 = data[data[data.columns[-1]] == 0]
    df_bit_output_1 = data[data[data.columns[-1]] == 1]

    axes.scatter(df_bit_output_0['A'], df_bit_output_0['B'], c='r', marker='o', label=f"{data.columns[-1]} = 0", s
    axes.scatter(df_bit_output_1['A'], df_bit_output_1['B'], c='b', marker='o', label=f"{data.columns[-1]} = 1",

    x = np.random.uniform(low=-1,high=3,size=100)
    y = -1
    y = (-model_weights[2] - model_weights[0]*x) / model_weights[1]
    axes.plot(x,y,color='k',label='Decision Boundary')

    axes.grid(True)
    axes.legend()
    axes.set_xlim([-0.5,3])
    axes.set_ylim([-1,3])
    axes.set_xticks(np.arange(0,4))
    axes.set_yticks(np.arange(-1,4))
    axes.set_xlabel('A', fontsize=18)
    axes.set_ylabel('B', fontsize=18)
    if not with_bias:
        axes.set_title("Plot of Decision Boundary in PTA (with fixed Bias = 0)")
    else:
        axes.set_title("Plot of Decision Boundary in PTA (with learnable Bias)")
    plt.show()
```

Function to apply PTA on AND, OR, XOR output Datasets

In [ ]:

```
def apply_perceptron_on_bit_dataset(dataset, with_bias):
    x_train, y_train = split_bit_dataset(dataset, with_bias=with_bias)

    perceptron_model = Perceptron()
    perceptron_model.fit(x_train, y_train)
    print("Perceptron Model Weights",perceptron_model.weights)

    plot_bit_dataset(dataset, with_bias=with_bias, model_weights=perceptron_model.weights)
```

Applying PTA on AND Dataset

We can see that, linear Decision boundary is able to perfectly classify the two classes (AND=0 and AND=1).

As the output labels (0 and 1) are linearly separable, so the Perceptron Training algorithm will converge according to Perceptron Convergence theorem. So, we can easily find a linear Decision boundary that can separate two output labels.

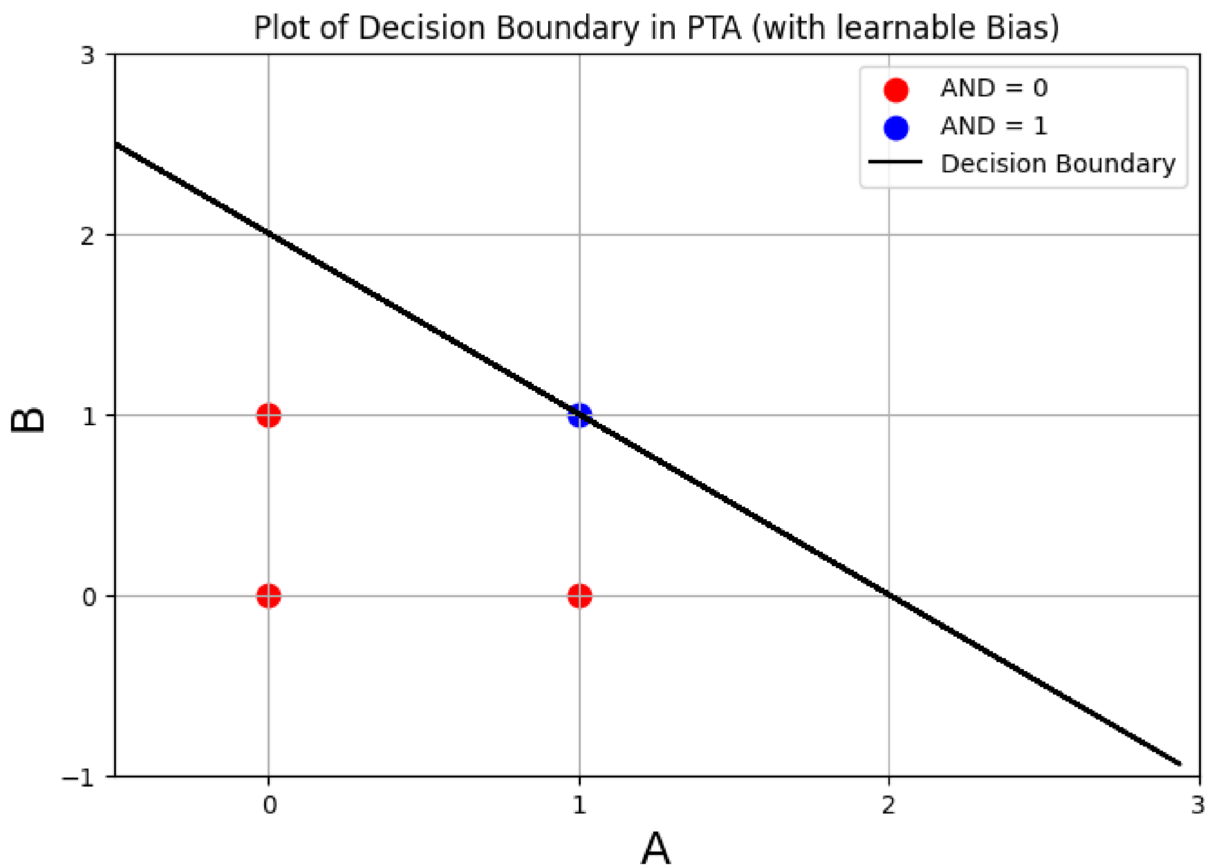
Assumption: Any input lying on the Decision boundary will be considered to have a output label as "1".

Because we used similar condition in Signum activation function (x>=0 implies y=1)

In [ ]:

```
print("Applying PCA on AND Dataset (with Learnable Bias):")
apply_perceptron_on_bit_dataset(dataset=AND_df, with_bias=True)
```

```
Applying PCA on AND Dataset (with Learnable Bias):
Perceptron Model Weights [ 1.  1. -2.]
```

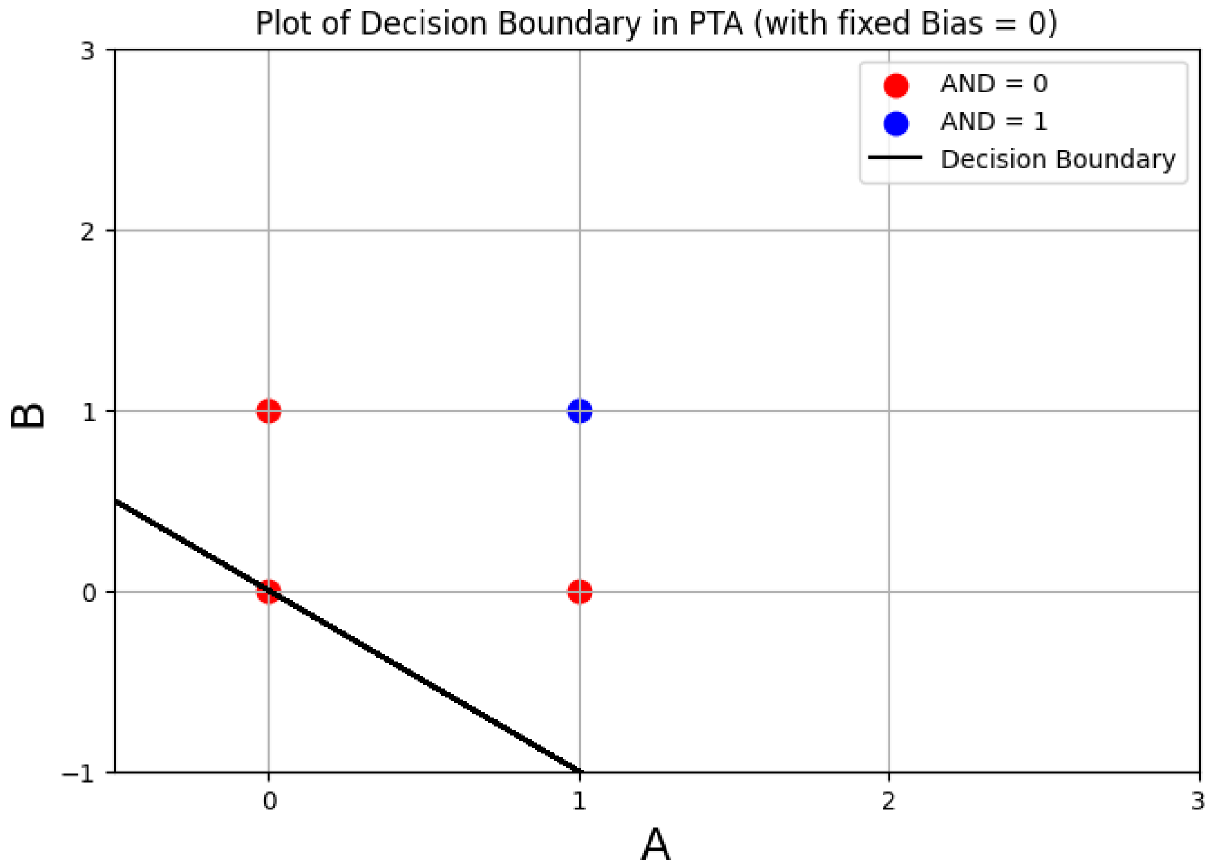


We can see that, linear Decision boundary is not able to perfectly classify the two classes (AND=0 and AND=1), when PTA is implemented with a fixed bias of 0. Reason being the same for Part 2-d.

In [ ]:

```
print("Applying PCA on AND Dataset (with Fixed Bias=0):")
apply_perceptron_on_bit_dataset(dataset=AND_df, with_bias=False)
```

```
Applying PCA on AND Dataset (with Fixed Bias=0):
Perceptron Model Weights [1. 1. 0.]
```



Applying PTA on OR Dataset

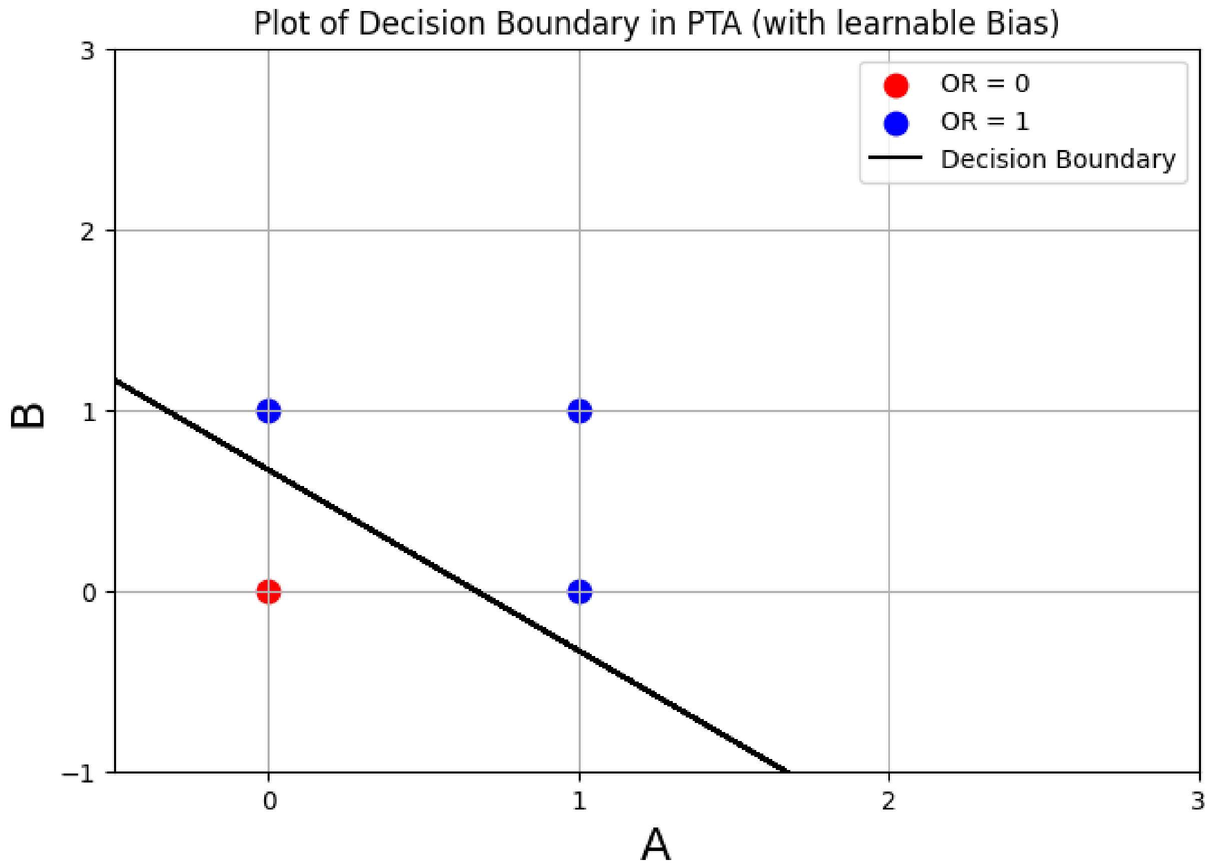
We can see that, linear Decision boundary is able to perfectly classify the two classes (OR=0 and OR=1).

As the output labels (0 and 1) are linearly separable, so the Perceptron Training algorithm will converge according to Perceptron Convergence theorem. So, we can easily find a linear Decision boundary that can separate two output labels.

In [ ]:

```
print("Applying PCA on OR Dataset (with Learnable Bias):")
apply_perceptron_on_bit_dataset(dataset=OR_df, with_bias=True)
```

```
Applying PCA on OR Dataset (with Learnable Bias):
Perceptron Model Weights [ 3.  3. -2.]
```



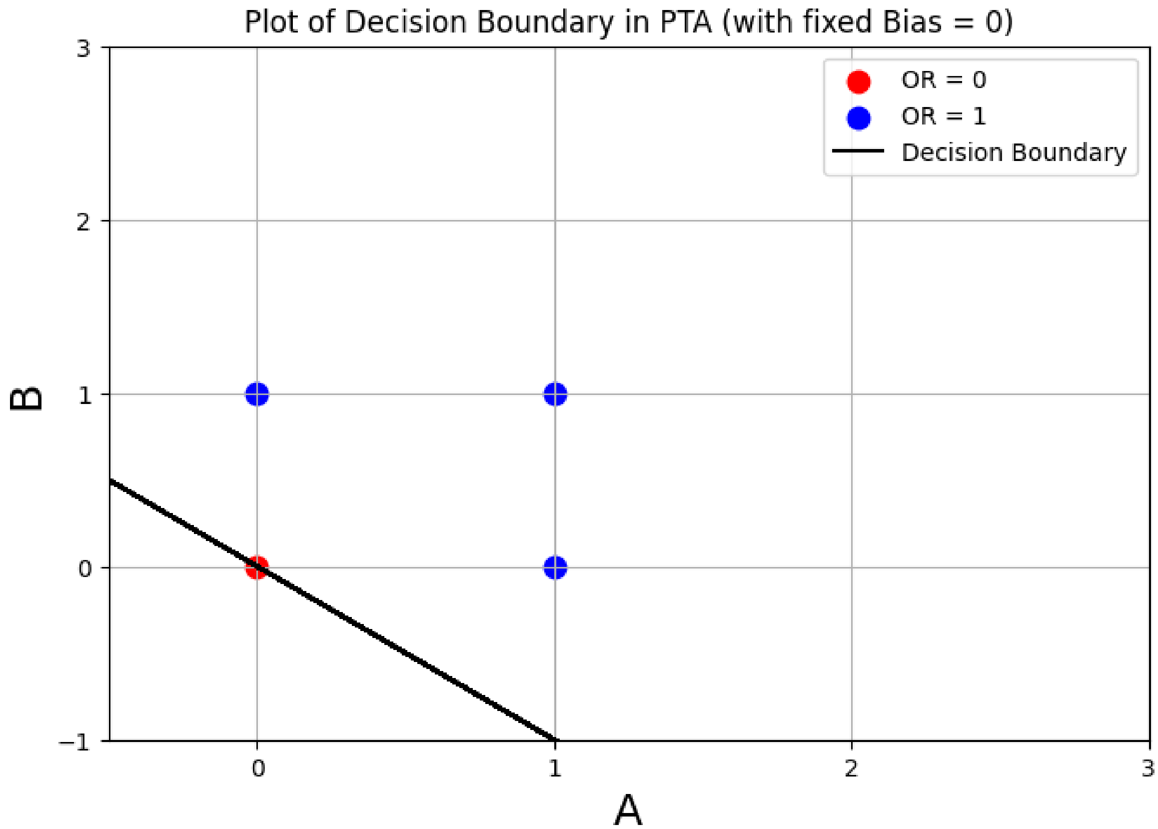
We can see that, linear Decision boundary is able to perfectly classify the two classes (OR=0 and OR=1) in case of PTA with fixed Bias of 0.

We can find a Decision boundary that passes through origin & classfies two classes.

In [ ]:

```
print("Applying PCA on OR Dataset (with Fixed Bias=0):")
apply_perceptron_on_bit_dataset(dataset=OR_df, with_bias=False)
```

```
Applying PCA on OR Dataset (with Fixed Bias=0):
Perceptron Model Weights [1. 1. 0.]
```



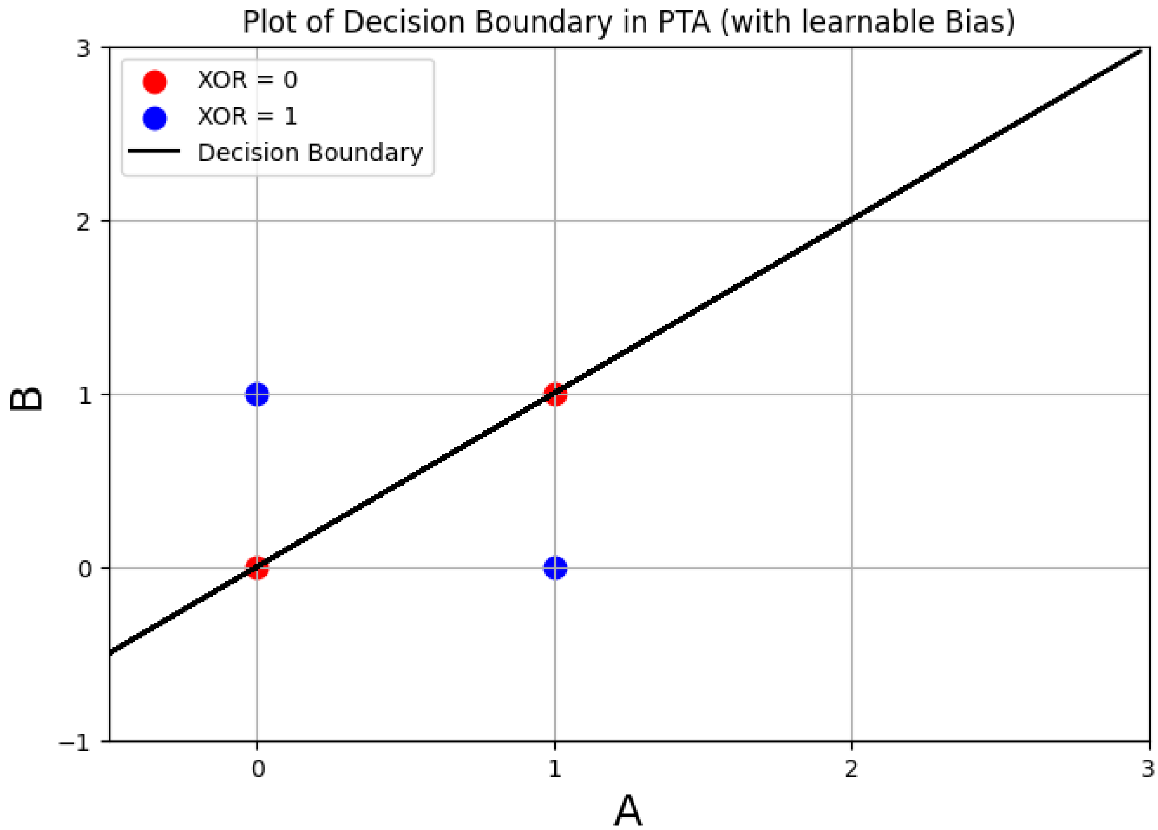
Applying PTA on XOR Dataset

We can see that, linear Decision boundary can not perfectly classify the two classes (XOR=0 and XOR=1).

In [ ]:

```
print("Applying PCA on XOR Dataset (with Learnable Bias):")
apply_perceptron_on_bit_dataset(dataset=XOR_df, with_bias=True)
```

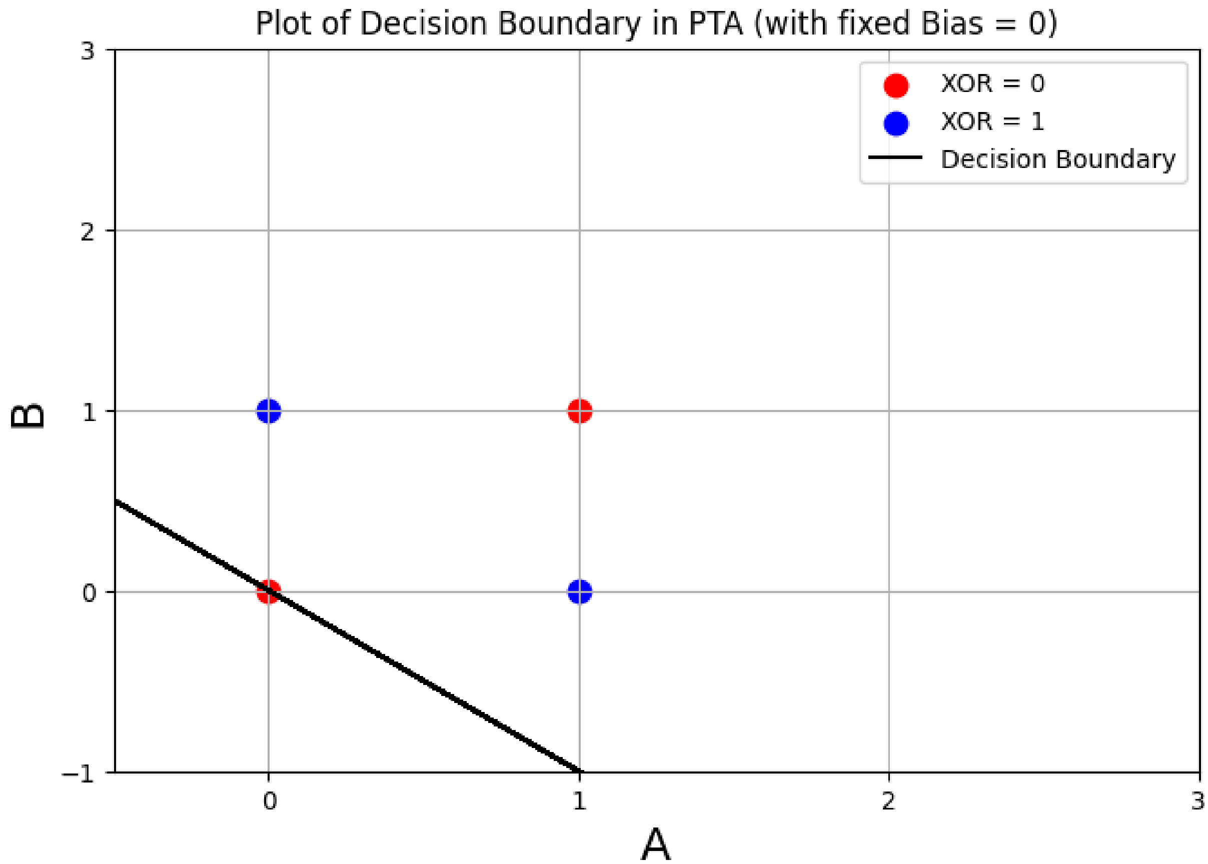
```
Applying PCA on XOR Dataset (with Learnable Bias):
Perceptron Model Weights [-1.  1.  0.]
```



In [ ]:

```
print("Applying PCA on XOR Dataset (with Fixed Bias=0):")
apply_perceptron_on_bit_dataset(dataset=XOR_df, with_bias=False)
```

```
Applying PCA on XOR Dataset (with Fixed Bias=0):
Perceptron Model Weights [-1. -1.  0.]
```



Explaination for why Decision Boundary doesn't exists in XOR dataset & PTA is not applicable in XOR data!

We know that, Perceptron Learning Algorithm makes a stronger assumption about the separablility of data by a hyperplane (linear deicision boundary).

By Perceptron Convergence Theorem, if the dataset is linearly separable then PTA is guaranteed to converge. But the XOR Dataset is not linearly separable by a hyperplane (Decision boundary), hence the PTA will never converge. Due to we can't use PTA here and find the values of Perceptron model's weights and bias. Hence, no Decision Boundary.

By a look at the AND and OR Dataset, we can see that a linear Hyperplane (Decision boundary) can separate the two output labels (0 and 1), hence the PTA will definitely converge. So, PTA can be applied in AND and OR data, and we can find the values of Perceptron model's weights and bias. Hence, Decision Boundary can be found easily.

As in XOR dataset, there is no separability between two output labels (0 and 1) by a hyperplane. So, PTA can't be used on such a XOR dataset. Hence, no hyperplane will exist that can separate two output labels (0 and 1) (as also clearly visible in above Plot). Hence, no Decision boundary will exits for above XOR Dataset.

From the above plot of XOR Dataset, we can see the linear Decision boundary fails to perfectly classifying the two labels (two output labels (0 and 1)), indicating Decision Boundary can't exist.

Q2 Part-6

If we have a Hyperplane equation and a Test sample data point. So, to compute the class (0 or 1) it belongs to:

We do the following:

1) Plug the values of Data points (x1, x2, ..., xn) in Hyperplane equation (w0 + w1.x1 + w2.x2 + ... + wn.xn) to get a real number h(x).

```
h(x)  =  w0 + w1.x1 + w2.x2 + ... + wn.xn
```

2) If this real number h(x) = (w0 + w1.x1 + w2.x2 + ... + wn.xn) is greater than or equal to 0 (>=0), then the class of this data point is "1" or "positive class". As, after signum activation function, the values of h(x) will be mapped to "1".

3) If this real number h(x) = (w0 + w1.x1 + w2.x2 + ... + wn.xn) is less than 0 (<0), then the class of this data point is "0" or "negative class". As, after signum activation function, the values of h(x) will be mapped to "-1" denoting "negative class or 0".

4) If h(x) >= 0:

```
        signum(h(x)) = 1       -----> "+ve class" or "1"

    If h(x) < 0:

        signum(h(x)) = -1      -----> "-ve class" or "0"
```

Assumptions:

If the data point lies on the Decision boundary (or Hyperline), i.e, it satisfies h(x) = 0.

Then the class of this data point is considered to be "+ve class" or "1".

Section - C

(Algorithm implementation using packages)

Q3

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

Load the Dataset

In [ ]:

```
df = pd.read_csv('BitcoinHeistData.csv')
```

In [ ]:

```
df.head()
```

Out[ ]:

	address	year	day	length	weight	count	looped	neighbors
0	111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.008333	1	0	2
1	1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.000244	1	0	1
2	112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.000000	1	0	2
3	1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.003906	1	0	2
4	1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.072848	456	0	1

In [ ]:

```
df['address'].value_counts()
```

Out[ ]:

```
1LXrSb67EaH1LGc6d6kWHq8rgv4ZBQAcpU    420
16cVG72goMe4sNqZhnpmnqfCMZ1uSFbUit    261
12wQZTDmA8onM3sEt4jwcvzDxnNXxD8Vza    207
12YursV58dRT2c9iuZg3jEWfwgTDamBcnd    183
1LEq4WmpCrqBd7V3PywE2nvFUFC3QTe52x    176
                                     ...
14m4NjEQjLKrcjtN3doN7TgNZi3nbvPnkL      1
1CJrNRSNJepexvLFt3wSKZkzrHRag2UMCA      1
1Fsi7R5115vXKcSmFEoDUqqmEW4oT2W5AV      1
1GTkpRYXAK71c5DP2V7irDmYtvmhS46h29      1
3LFFBxp15h9KSFtaw55np8eP5fv6kdK17e      1
Name: address, Length: 2631095, dtype: int64
```

In [ ]:

```
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
```

No null values present in the Datset

In [ ]:

```
df.isnull().sum()
```

Out[ ]:

```
address      0
year         0
day          0
length       0
weight       0
count        0
looped       0
neighbors    0
income       0
label        0
dtype: int64
```

Encoding 'address' column into numerical column using 'Label Encoder'

In [ ]:

```
encoder = LabelEncoder()
df['address'] = encoder.fit_transform(df['address'])
```

In [ ]:

```
df.head()
```

Out[ ]:

	address	year	day	length	weight	count	looped	neighbors	income	label
0	23	2017	11	18	0.008333	1	0	2	100050000.0	princetonCerber
1	128	2016	132	44	0.000244	1	0	1	100000000.0	princetonLocky
2	169	2016	246	0	1.000000	1	0	2	200000000.0	princetonCerber
3	217	2016	322	72	0.003906	1	0	2	71200000.0	princetonCerber
4	293	2016	238	144	0.072848	456	0	1	200000000.0	princetonLocky

In [ ]:

```
df['address'].value_counts()
```

Out[ ]:
```
1925732    420
481920     261
105390     207
65867      183
1895700    176
          ...
292374       1
1074094      1
1445117      1
1506008      1
2580865      1
Name: address, Length: 2631095, dtype: int64
```

Randomly shuffle the Dataset (since all the output labels are grouped)

In [ ]:
```
df = df.sample(frac=1)
```

In [ ]:
```
df
```

Out[ ]:

	address	year	day	length	weight	count	looped	neighbors	income	label
505972	506425	2012	110	24	8.786737e-01	16	0	2	1.941000e+09	white
2620114	2369474	2018	34	136	2.270832e-01	6873	0	2	4.556888e+07	white
1363813	17576	2014	238	0	1.000000e+00	1	0	2	9.998000e+07	white
1176214	2199457	2014	50	6	1.000000e+00	1	0	2	5.802917e+07	white
296992	2353600	2011	266	58	1.994046e-06	46	0	1	1.030000e+08	white
...	...	...	...	...	...	...	...	...	...	...
1906332	398193	2016	50	42	7.148791e-02	70	0	2	1.115550e+08	white
297907	156143	2011	267	14	5.156250e-01	2	0	2	6.368000e+09	white
805128	483661	2013	44	0	5.000000e-01	1	0	1	2.030000e+10	white
191875	2132493	2011	161	128	1.898076e-12	24	0	2	1.315936e+10	white
2637699	2528471	2018	52	0	1.000000e+00	1	0	1	3.256920e+08	white

2916697 rows × 10 columns

Q3 Part-1

Training a Decision Tree using both the Gini index and the Entropy by changing the max-depth

In [ ]:
```
def train_test_validation_split(dataset: pd.DataFrame, size):
    n = dataset.shape[0]
    train_size = round((size[0]/100) * n)
    valid_size = round((size[1]/100) * n)
    test_size = round((size[2]/100) * n)
```

```
    # Split into Training, Validating & Testing Data
    x_train = dataset.iloc[0:train_size].drop('label', axis=1)
    y_train = dataset.iloc[0:train_size]['label']
    x_valid = dataset.iloc[train_size: train_size + valid_size].drop('label', axis=1)
    y_valid = dataset.iloc[train_size: train_size + valid_size]['label']
    x_test = dataset.iloc[train_size + valid_size:].drop('label', axis=1)
    y_test = dataset.iloc[train_size + valid_size:]['label']
    return x_train, y_train, x_valid, y_valid, x_test, y_test
```

In [ ]:

```
x_train, y_train, x_valid, y_valid, x_test, y_test = train_test_validation_split(df, [70,15,15])
```

In [ ]:

```
def DecisionTreeAlgorithm(split_criteria, x_train, y_train, x_valid, y_valid, x_test, y_test):
    print(f"Training, Validating & Testing the Decision Tree with '{split_criteria}' criteria...")
    print("Accuracy of Decision Tree classifers with various depth is shown below.\n")

    for depth in  [4, 8, 10, 15, 20]:
        # Training the Decision Tree
        tree = DecisionTreeClassifier(max_depth=depth, criterion=split_criteria)
        tree.fit(x_train, y_train)

        # Accuracy on Validation Set
        y_pred_valid = tree.predict(x_valid)
        validation_accuracy = (y_pred_valid == y_valid).sum() / y_valid.shape[0]
        print(f"Accuracy with 'Depth = {depth}' on Validation set is: {validation_accuracy}")

        # Accuracy on Testing Set
        y_pred_test = tree.predict(x_test)
        testing_accuracy = (y_pred_test == y_test).sum() / y_test.shape[0]
        print(f"Accuracy with 'Depth = {depth}' on Testing set is: {testing_accuracy}\n")
```

Training Decision trees using the Gini index (for various depths)

In [ ]:

```
DecisionTreeAlgorithm(split_criteria='gini', x_train=x_train, y_train=y_train,
                      x_valid=x_valid, y_valid=y_valid, x_test=x_test, y_test=y_test)
```

```
Training, Validating & Testing the Decision Tree with 'gini' criteria...
Accuracy of Decision Tree classifers with various depth is shown below.

Accuracy with 'Depth = 4' on Validation set is: 0.9857075919132353
Accuracy with 'Depth = 4' on Testing set is: 0.9859566998244588

Accuracy with 'Depth = 8' on Validation set is: 0.9864207266202673
Accuracy with 'Depth = 8' on Testing set is: 0.9866538363077824

Accuracy with 'Depth = 10' on Validation set is: 0.9868207220488908
Accuracy with 'Depth = 10' on Testing set is: 0.987005833089526

Accuracy with 'Depth = 15' on Validation set is: 0.9881921349470292
Accuracy with 'Depth = 15' on Testing set is: 0.9884366771503803

Accuracy with 'Depth = 20' on Validation set is: 0.9875087141861236
Accuracy with 'Depth = 20' on Testing set is: 0.9878103971620831
```

Training Decision trees using the Entropy (for various depths)

In [ ]:

```
DecisionTreeAlgorithm(split_criteria='entropy', x_train=x_train, y_train=y_train,
                      x_valid=x_valid, y_valid=y_valid, x_test=x_test, y_test=y_test)
```

```
Training, Validating & Testing the Decision Tree with 'entropy' criteria...
Accuracy of Decision Tree classifers with various depth is shown below.

Accuracy with 'Depth = 4' on Validation set is: 0.9856298785156741
Accuracy with 'Depth = 4' on Testing set is: 0.9859109859566998

Accuracy with 'Depth = 8' on Validation set is: 0.9859978743100078
Accuracy with 'Depth = 8' on Testing set is: 0.9862721255119953

Accuracy with 'Depth = 10' on Validation set is: 0.9872092890366967
Accuracy with 'Depth = 10' on Testing set is: 0.9874652574605032

Accuracy with 'Depth = 15' on Validation set is: 0.9888206991920092
Accuracy with 'Depth = 15' on Testing set is: 0.9890081004973669

Accuracy with 'Depth = 20' on Validation set is: 0.9877144261208444
Accuracy with 'Depth = 20' on Testing set is: 0.9877692546811001
```

Greatest accuracy is observed for Decision Tree with "Depth" as 15 with criteria as "Entropy".

Q3 Part-2

Implementing a Random Forest Algorithm using 100 Decision Tree Classifier (with max-depth as 3)

In [ ]:

```
# number of trees
num_trees = 100
```

In [ ]:

```
def train_trees(x_train, y_train, num_trees):
    trees = [DecisionTreeClassifier(max_depth=3, criterion='entropy') for i in range(num_trees)]

    # random indices for bootstrap samples
    m = x_train.shape[0]
    indices = np.arange(m, dtype=np.int16)

    for i in range(num_trees):
        # Selecting n random samples with replacement from training set
        random_indices = np.random.choice(indices, m//2)

        # Bootstrap training data
        x_bootstrap = x_train.iloc[random_indices]
        y_bootstrap = y_train.iloc[random_indices]

        # Train/fit the Data on the Trees
        trees[i].fit(x_bootstrap, y_bootstrap)

    return trees
```

In [ ]:

```
def predict_outputs(trees, x_test, y_test, num_trees):
    # Test all the 'num_trees=100' trees on the Testing samples and save the Testing results
    trees_predictions = np.empty(shape=(num_trees, x_test.shape[0]), dtype='object')
    for i in range(num_trees):
        trees_predictions[i] = trees[i].predict(x_test)

    # Compute the "Majority vote" for each Testing sample outputs for all 100 of Decision Tree
    y_prediction = np.empty(shape=(x_test.shape[0]), dtype='object')
    for i in range(x_test.shape[0]):
        y_prediction[i] = pd.Series(trees_predictions[:,i]).value_counts().index[0]

    return y_prediction, trees_predictions
```

All 100 Decision Trees

In [ ]:

```
trees = train_trees(x_train=x_train, y_train=y_train, num_trees=num_trees)
```

Reporting Accuracy of Random Forest on Testing Set

In [ ]:

```
y_prediction, trees_prediction = predict_outputs(
            trees=trees, x_test=x_test, y_test=y_test, num_trees=num_trees)
```

In [ ]:

```
accuracy = (y_prediction == y_test).sum() / y_test.shape[0]

print(f"Training a Random Forest with '{num_trees} Decision-Trees' each of depth 3.\n")
print(f'Accuracy of Random Forest on Testing set is: {accuracy}')
```

```
Training a Random Forest with '100 Decision-Trees' each of depth 3.

Accuracy of Random Forest on Testing set is: 0.9859109859566998
```

Reporting Accuracy of Random Forest on Validation Set

In [ ]:

```
y_prediction, trees_prediction = predict_outputs(
            trees=trees, x_test=x_valid, y_test=y_valid, num_trees=num_trees)
```

In [ ]:

```
accuracy = (y_prediction == y_valid).sum() / y_valid.shape[0]

print(f"Random Forest with '{num_trees} Decision-Trees' each of depth 3.\n")
print(f'Accuracy of Random Forest on Validation Set is: {accuracy}')
```

```
Random Forest with '100 Decision-Trees' each of depth 3.

Accuracy of Random Forest on Validation Set is: 0.9856298785156741
```

Performance of Random Forest

The Testing accuracy of Random forests is nearly same as that we obtained in part-a.

In part-a we used strong classifiers, i.e, Decision Trees with greater depth.

In this Random forest, we used 100 weak Decision Trees classifiers (each of depth 3). Then, we ensembled or combined the results of all those 100 weak classfiers to produce a strong Decision Tree Classifier, which has nearly the same accuracy as those of strong Decision Tree Classifier. Hence, this shows the power of ensembling.

Q3 Part-3

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In [ ]:

```
# No. of estimators
estimators = [4, 8, 10, 15, 20]
```

Implementing a Random Forest Algorithm (of 100 Decision Tree Classifier) using ADABOOST

Training Random Forests that are ensembled with AdaBoost Algorithm

In [ ]:

```
def train_AdaBoost_classifiers(x_train, y_train, estimators):
    AdaBoosts = []

    for n_estimator in estimators:
        # Initializing a Decision Tree
        tree = DecisionTreeClassifier(max_depth=15, criterion='entropy')

        # Adaboost Classifier
        adaboost = AdaBoostClassifier(base_estimator=tree, n_estimators=n_estimator)

        # Training Adaboost algorithm using fit()
        adaboost.fit(x_train, y_train)

        # Adding Adaboost classifiers
        AdaBoosts.append(adaboost)
    return AdaBoosts
```

Testing Random Forests that are ensembled using AdaBoost Algorithm

In [ ]:

```
def test_AdaBoost_classifiers(adaboosts, x_test, y_test, estimators):

    for i in range(len(estimators)):
        # Computing Training and Testing accuracy
        # y_prediction = adaboosts[i].predict(x_test)
        # accuracy = accuracy_score(y_test=y_test, y_pred=y_prediction)
        accuracy = adaboosts[i].score(x_test, y_test)

        print(f"Accuracy with 'n_estimators = {estimators[i]}' is: {accuracy}")
```

In [ ]:

```
adaboosts = train_AdaBoost_classifiers(x_train=x_train, y_train=y_train, estimators=estimators)
```

Reporting Accuracy of Random Forest (ensembled with Adaboost) on Validation Set

In [ ]:

```
print("Accuracy of AdaBoost based Random Forests on Validation Set...\n")
test_AdaBoost_classifiers(adaboosts=adaboosts, x_test=x_valid, y_test=y_valid, estimators=estimators)
```

```
Accuracy of AdaBoost based Random Forests on Validation Set...

Accuracy with 'n_estimators = 4' is: 0.9882127061405013
Accuracy with 'n_estimators = 8' is: 0.9872595741762952
Accuracy with 'n_estimators = 10' is: 0.9858584473320305
Accuracy with 'n_estimators = 15' is: 0.9873692872081462
Accuracy with 'n_estimators = 20' is: 0.9874858573044879
```

Reporting Accuracy of Random Forest (ensembled with Adaboost) on Testing Set

In [ ]:

```
print("Accuracy of AdaBoost based Random Forests on Testing Set...\n")
test_AdaBoost_classifiers(adaboosts=adaboosts, x_test=x_test, y_test=y_test, estimators=estimators)
```

```
Accuracy of AdaBoost based Random Forests on Testing Set...

Accuracy with 'n_estimators = 4' is: 0.9882538216793446
Accuracy with 'n_estimators = 8' is: 0.9876823983323582
Accuracy with 'n_estimators = 10' is: 0.9860001279988297
Accuracy with 'n_estimators = 15' is: 0.9874675431538912
Accuracy with 'n_estimators = 20' is: 0.9877692546811001
```

Results

The accuracy of "AdaBoost based Decision Tree Clssifer" is nearly same as that of "Random forest Classifer" in part-b.

Though the Adaboost shows slightly greater accuracy after two decimal places.

Since AdaBoost and Bagging are iterative and paralle versions of Ensembling repectively.

Both of them shows similar accuracy on the Testing set. This shows benefits of Ensembling and how strong classifers are made from weak classifiers