1) PCA,SVD,t-SNE, Dataset A, Logistic regression

a) **Principal Component Analysis** is a technique that helps in reducing the dimensionality of a given dataset, while at the same time, retaining the maximum possible information. It offers a tradeoff between the amount of information and the dimensionality of the dataset. A reduced number of features means that machine learning algorithms can run faster since they mostly run in polynomial time wrt the number of features. PCA calculates a covariance matrix. A positive covariance means directly related, while a negative covariance means inversely related features. PCA uses this matrix to identify the most important features that need to be retained, whose information is the least related to any other features and hence unique. This way, PCA is able to select the most important features while eliminating the least important and still retaining the maximum possible information.

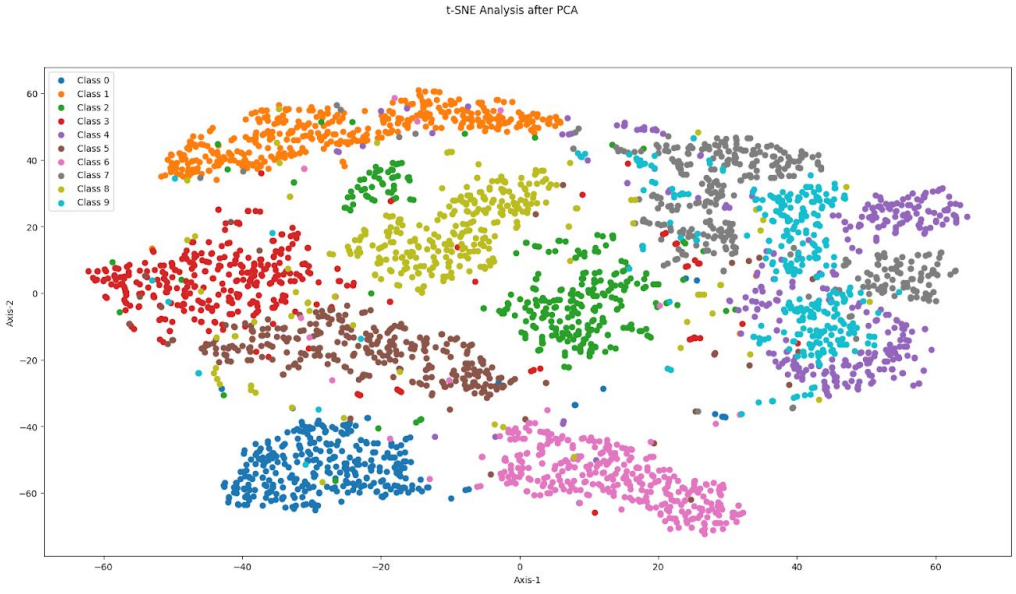
b) **Singular Value Decomposition** is actually a linear algebraic decomposition technique where an (mxn) matrix M, is broken down into three component matrices U(mxp), S(pxp), and V'(pxn). Here p is the rank of the matrix M which is the maximum number of independent column vectors in M. The V matrix is representative of the unique features that must be selected in order to retain maximum information. The singular columns can be chosen by selecting a p1. We then select a subset of the V matrix as V1(nxp1). The original matrix M multiplied by this V1 gives us the new feature matrix M1 = M*V1(mxp1) having a reduced number of features representing almost equal information as M.

c) **t-distributed Stochastic Neighbor Embedding** is another dimensionality reduction algorithm, but with an exceptional feature that it can work very well identifying and filtering related features even having non-linear relationships. t-SNE is often the preference when it comes to linearly inseparable data. t-SNE's working, as a high-level explanation, involves fitting a high dimension probability distribution to the given dataset and later followed by recreating a dataset with a lower dimension, following the same distribution as best as possible. The distribution used in this process is called the t-distribution. The terms stochastic and neighbor appear in the name since the t-distribution is used to fit across neighboring points by stochastic guessing.

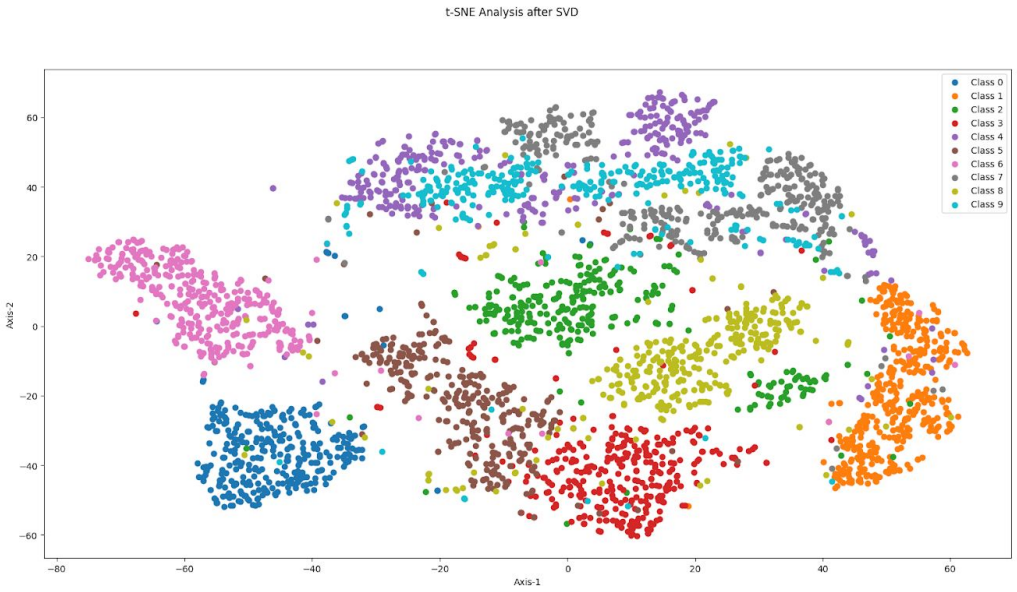
d) Consider the following screenshot which shows the class distribution for class labels 0-9 after stratified train-test split both the frequency and the fraction. We can clearly see that the stratified split resulted in almost equal distribution of samples from each class label for both the test and train set.

```
Train Data Class Distribution
Count           Fraction
320       0.09523809523809523
395       0.11755952380952381
314       0.09345238095238095
339       0.10089285714285715
333       0.09910714285714285
318       0.09464285714285714
353       0.10505952380952381
345       0.10267857142857142
328       0.09761904761904762
315       0.09375
Total: 3360
Test Data Class Distribution
Count           Fraction
80        0.09523809523809523
99        0.11785714285714285
79        0.09404761904761905
85        0.10119047619047619
83        0.0988095238095238
80        0.09523809523809523
88        0.10476190476190476
86        0.10238095238095238
82        0.09761904761904762
78        0.09285714285714286
Total: 840
```

e) The accuracy obtained after PCA: 0.861904761904762



f) The accuracy obtained after SVD: 0.8738095238095238



g) We can see that both PCA and SVD resulted in similar accuracy which is in fact also very close to the accuracy obtained without PCA/SVD. Thus we successfully reduced the dimensionality of the datasets without losing much information.

```
# features before: 784 and after PCA: 100
Test Accuracy after PCA: 0.861904761904762
# features before: 784 and after SVD: 100
Test Accuracy after SVD: 0.8738095238095238
Accuracy without PCA/SVD: 0.8690476190476191
```

2) Bootstrapping, Dataset C, Linear regression

a) 10 Bootstrap samples were taken from 80% of the data (20% separated for testing), of sample size 200 each.

```
# hyperparameters
num_test = data.shape[0]//5
sample_size = 200
num_samples = 10
```

Consider the following screenshot for the results.

```
Bias: 9.625994051833201
MSE: 146.06885875021848
Variance: 1.0837059735998804
Uncertainty: 52.325391290690426
```

b) Irreducible Uncertainty = MSE - Bias**2 - Variance

This is the uncertainty or the error demonstrated by the model. This includes the uncertainty involved in the data we are trying to model as well as whether the model is correctly modeling this data. This is coming out to be around 52.3254 as shown in the above screenshot.

3) Grid Search, K-Fold, Dataset A&B, Decision Tree & Gaussian NB

a) Selected the depth giving maximum validation accuracy after running a grid search for depths ranging from 2 to 20. K-Fold code was written with the following function header.

```
def kfoldCrossValidation(model, X,y, k):
    """
    Performing K-Fold Cross Validation

    Parameters
    ----------
    model : model to use

    k : k in k-fold

    X : input features

    y : output variable

    Returns
    -------
    2-tuple : (avg train accuracy, avg validation accuracy)
    """
```

Ran the above function for model=GaussianNB(), X,y from datasets A and B respectively, and k=5 to obtain the following results.

```
GNB Dataset A
Avg Training Accuracy: 0.6176388888888888
Avg Validation Accuracy: 0.5680952380952381
GNB Dataset B
Avg Training Accuracy: 0.5776388888888888
Avg Validation Accuracy: 0.575952380952381
```

Selected the depth giving maximum validation accuracy after running a grid search for depths ranging from 2 to 20. The following are the results of the maximum training/validation accuracy depth grid search for the decision tree using 4-fold cross-validation for model evaluation.

DATASET-A

```
        Depth   train acc       validate acc
Training 17 1.0 0.7035714285714286
Validation 17 1.0 0.7035714285714286
```

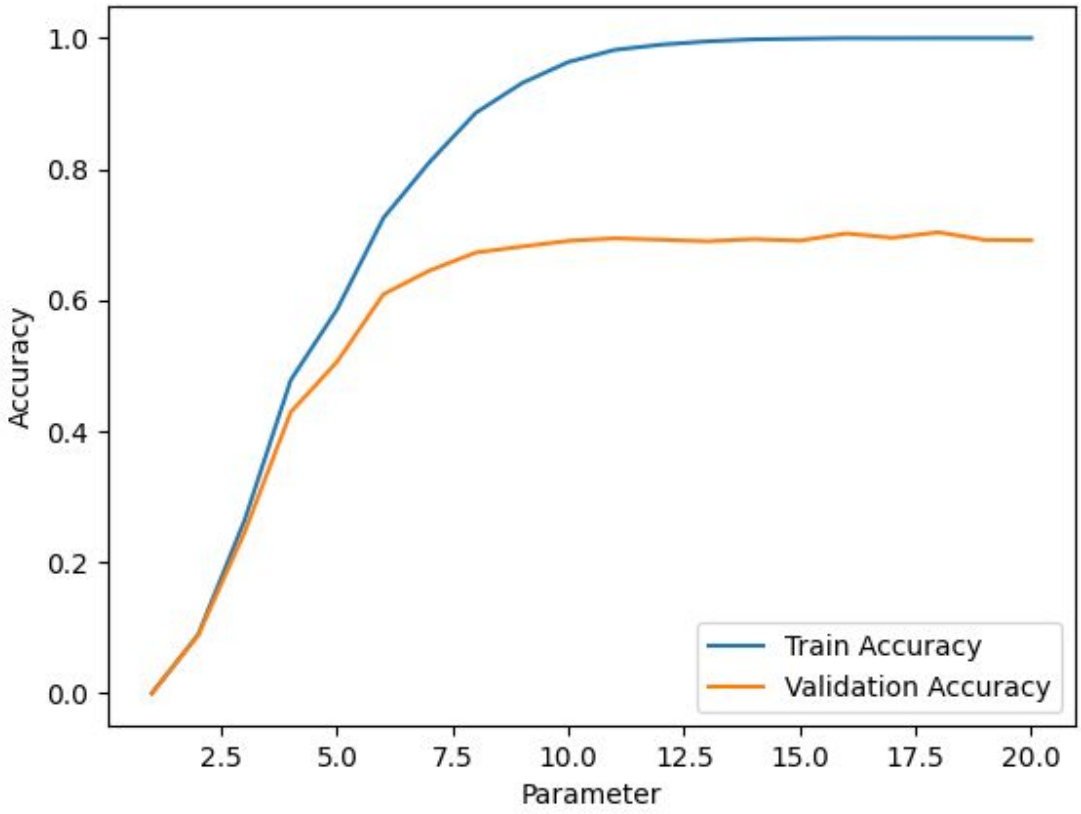
DATASET-B

```
        Depth   train acc       validate acc
Training 19 0.9999007936507937 0.5726190476190477
Validation 13 0.9895337301587303 0.5791666666666667
```

(The parameter axis here shows the max_depth of the decision tree)

b) Plots and analysis

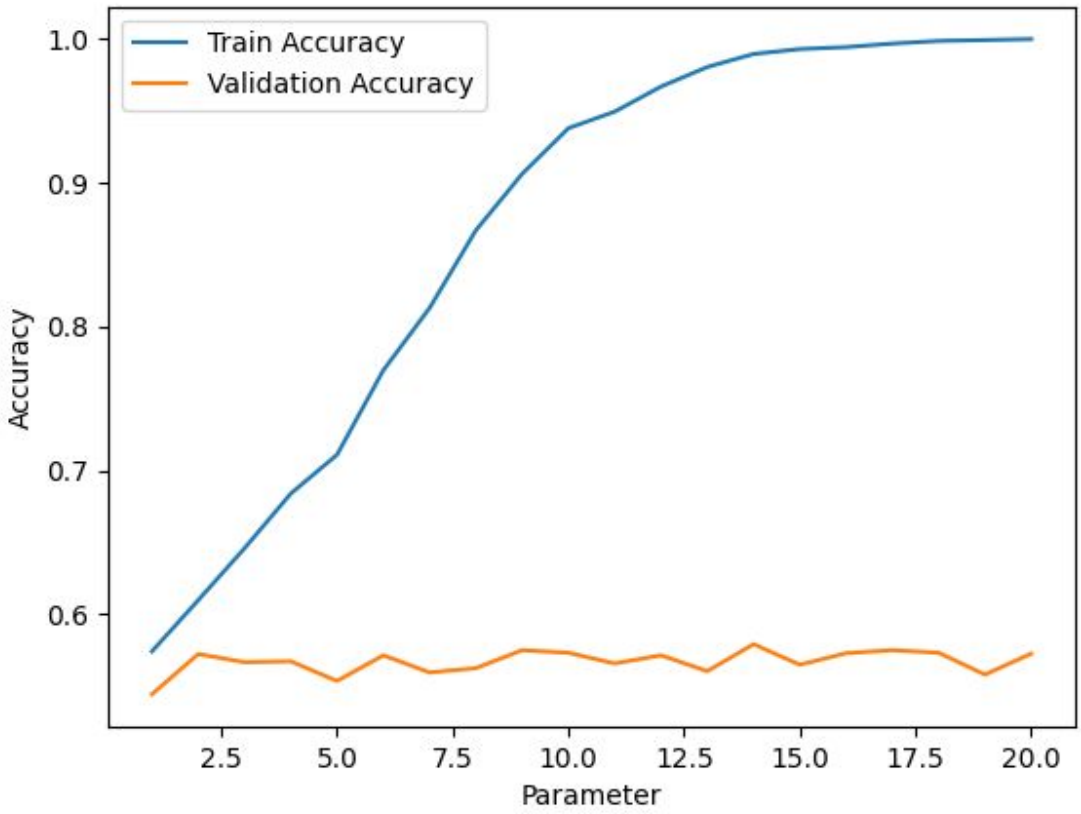
DATASET-A:



- As the tree depth is increasing, the training accuracy is moving towards 100%, while the validation accuracy is still around 75%.
- 100% train accuracy suggests that the leaf nodes are pure.

- But at the same time, the trees having depth>7 are starting to overfit, since the training accuracy is improving but the validation accuracy is not changing much.
- Ideally, there would be an early stopping point somewhere between depth=5 and depth=10. This point can be found out by training and testing over a large number of dataset samples that can be generated using various sampling methods.

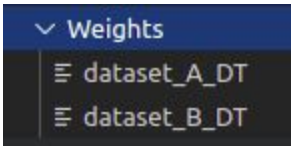
DATASET-B:



- As the tree depth is increasing, the training accuracy is moving very close to 100%, while the validation accuracy is still around 55%.
- Tending towards 100% train accuracy suggests that the leaf nodes are getting purer.
- While the training accuracy is improving with the depth increasing, there is no significant improvement in the validation accuracy, suggesting that the model is overfitting. But still, we can say that the models having depth<5 are better than those above it.

- The inability to improve validation accuracy suggests that either the data is not enough for the problem, or the model chosen is wrong.

c) Saved the best models in the 'project_root/Weights' directory
   i) depth=17 for dataset A
   ii) depth=13 for dataset B



Wrote the following functions using pickle library to save and load models:-

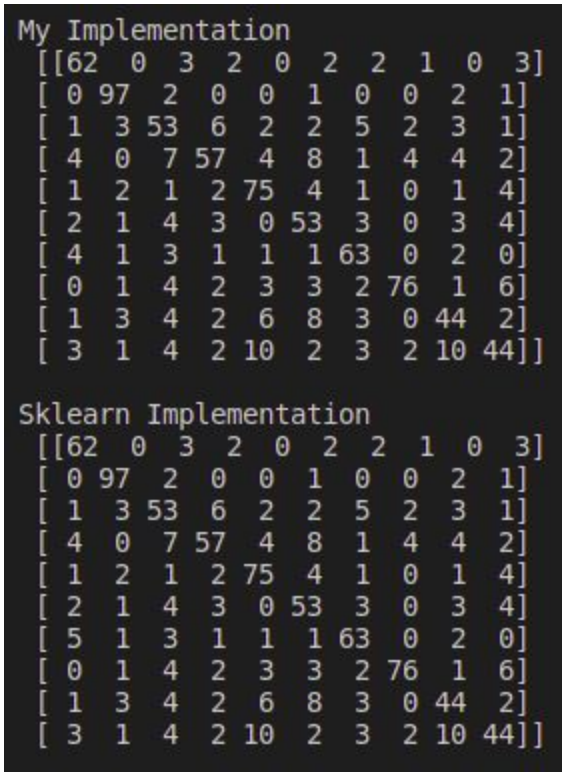
```
def save_model(Xtrain,ytrain,model1,filename,param=None):
    model = copy.deepcopy(model1)
    if param is not None:
        model.set_params(**param)
    model.fit(Xtrain,ytrain)
    pickle.dump(model, open(filename, 'wb'))
```

```
def load_model(filename):
    return pickle.load(open(filename,'rb'))
```

d) Consider the following output screenshots.

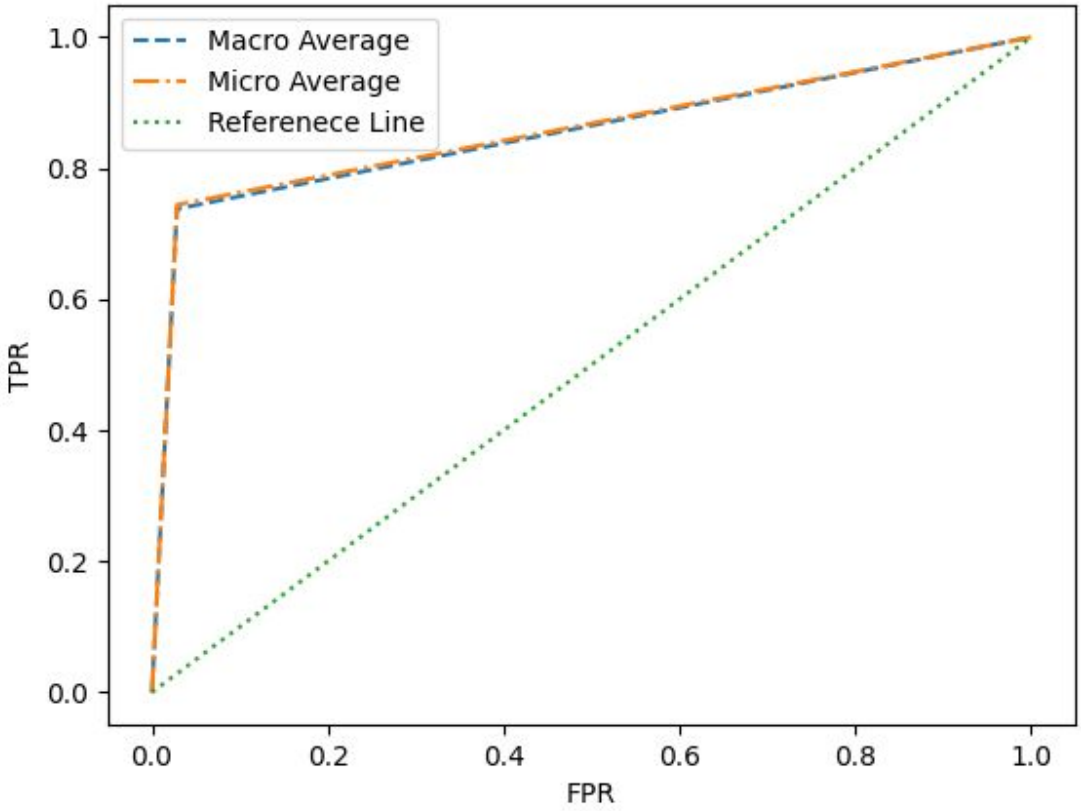
DATASET-A (Decision Tree)

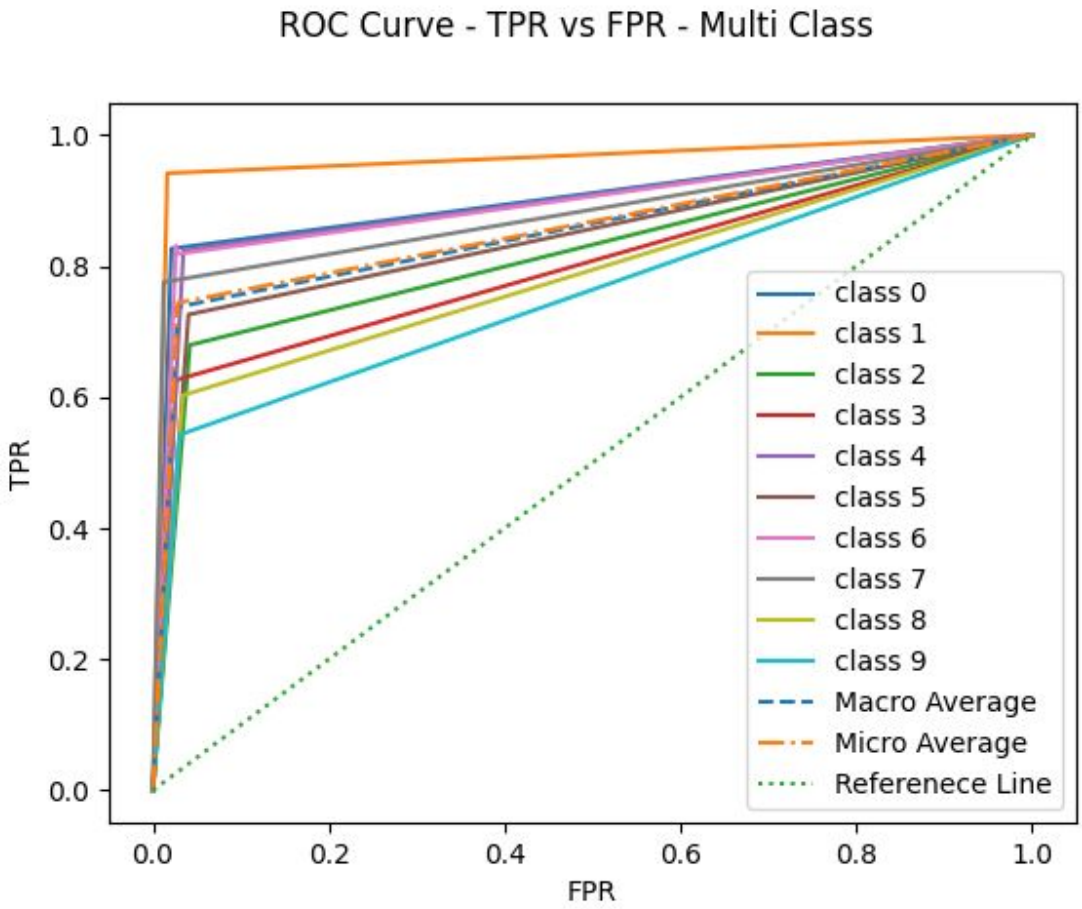
```
Accuracy Score: 0.7428571428571429

Macro Values
Precision: 0.7350475811984596   Recall: 0.7364119891613162      F1: 0.7357291526076017

Micro Values
Precision: 0.7428571428571429   Recall: 0.7428571428571429      F1: 0.7428571428571429
```



Note that the confusion matrix follows the same convention as the sklearn that **row represents the actual class while the column represents the predicted class**.

ROC Plot Dataset-A DecisionTree(max_depth=17)


ROC Curve - TPR vs FPR - Multi Class
Macro Average
Micro Average
Referenece Line
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0



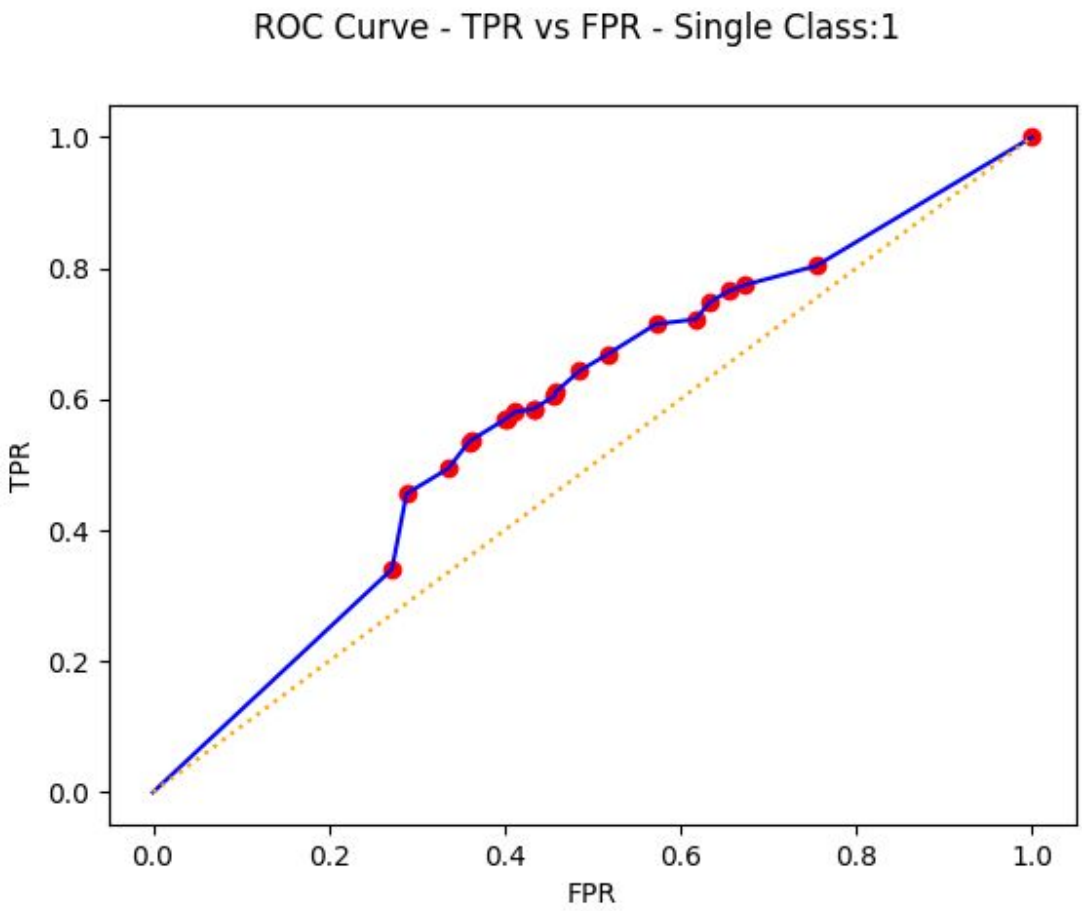
DATASET-B (Decision Tree)

```
Accuracy Score: 0.5761904761904761

Precision: 0.576271186440678    Recall: 0.5762538905046176      F1: 0.5762625383428678
```

```
My Implementation
 [[240 183]
 [173 244]]

Sklearn Implementation
 [[240 183]
 [173 244]]
```

Note that the confusion matrix follows the same convention as the sklearn that **row represents the actual class while the column represents the predicted class**.

ROC Plot Dataset-B DecisionTree(max_depth=13)



4) Gaussian Naive Bayes from scratch

- Performed random 80-20 train-test split.
- Ran both my own implementation and the sklearn library's implementation.
- The following results show the testing accuracy to compare the two models.
    - Dataset-A

```
My Implementation running...
0.5535714285714286
SkLearn Implementation running...
0.5619047619047619
```

    - Dataset-B

```
My Implementation running...
0.5035714285714286
SkLearn Implementation running...
0.5988095238095238
```

- Clearly, sklearn performs better than our model. The reason could be that sklearn is using some additional manipulation in order to improve the model performance, that my

implementation does not do. Our implementation is purely Gaussian Naive Bayes formula-based.

- Additionally, I had to use the decimal library in order to handle very small fractional numbers which might be handled differently by the sklearn module, leading to slightly different results.

Note: Find the theory questions in the document named 2018033_ML_THEORY_A2.pdf