COL774 Assignment 3 (Part-A)

Decision Tree & Random Forest

Het Shaileshkumar Patel(2019mcs2562)

March 27, 2020

1 Introduction

1.1 How to run code

I have submitted a file called Q1.py which will take multiple command line arguments.

- **-f|--data_folder <path>** : Path to parent folder where data is contained(i.e. the folder which contains given files train_x.txt, train_y.txt, valid_x.txt, valid_y.txt, test_x.txt and test_y.txt). If no value passed then it is assumed to be in `./data` directory.
- **-- part [any combination of a,b,c and d]** : which part of the assignment you want to run. e.x. you can run part a nd b by passing --part ab. If not passed then all four parts are executed.
- **--record_accuracies** : If passed then train, test and validation accuracies are recorded as tree grows during training as well as during post pruning.

1.2 Understanding Data

We were provided a sparse dataset for detecting files infected with virus, in specially stored form which we can read using xclib. The original dataset contained total **482** different features.

Now when I performed EDA on this dataset I found out that there were total **219** columns in train and validation dataset which contained only zero values. So in decision tree algorithm while we build decision tree using that training dataset, features having only zero value for every data-point will not be considered ever for splitting. So we can remove these columns from train, validation and test dataset as it without loss in accuracy as they won't be used for training purpose and doing so will provide improved run time.

So from now on we will be working on a subset of dataset having only 263 features! with Train Dataset containing 64713, Validation dataset containing 21572 and test Dataset with 21571 datapoints.

For both class labels datapoints are distributed as follows from which we can see that our dataset is not imbalanced:

	Y=0	Y=1
Train	30560	34153
Validation	10302	11270
Test	10151	11420

Table 1: Counts of datapoints in each class

2 Part A : Decision Tree Construction

I have implemented Decision Tree Classifier for the given problem in the `DecisionTreeClassifier` class in `DecisionTreeClassifier.py` file. Documentation of this class is as below (also given in the docstring)

```
Parameters:

        get_train_data(Default:False) : [boolean]
            If true then train, test and Validation accuracy recorded
            after adding record_data_frequency number of nodes

        record_data_frequency(Default:100) : [int]
            After how many nodes to record accuracy data

        get_prune_data(Default:False) : [boolean]
            If true then train, test and Validation accuracy recorded
            during pruning as we prune the nodeds from the tree

Attributes:

        root : [DTNode]
            Root of the decision Tree
        train_data : [Dictionary]
```

Dictionary containing accuracies over Train, Test and Validation Datasets
recorded as tree grows (You need to set get_train_data=True to get this)

prune_data : [Dictionary]
Dictionary containing accuracies over Train, Test and Validation Datasets
recorded as we prune tree (You need to set get_prune_data=True to get this)

Methods:

fit(self, X, Y[, X_test, Y_test, X_val, Y_val]) : [None]
This method builds decision tree iteratively using
X and Y as training Dataset [and records accuracies].
(If you have set get_train_data=True then pass X_test, Y_test and X_val, Y_val)

prune(self, X_val, Y_val, [X_train, Y_train, X_test, Y_test]) : [None]
This method post punes the tree that was trained during fit method using
methodology called Reduced error pruning which uses different pruning/validation
dataset to post prune the tree to avoid overfitting on fully grown tree

predict(self, X) : [list]
Returns list of predicted class labels for datapoints in X

score(self, X, Y_true) : [float]
Returns accuracy over dataset X and ground truth Y_true

get_number_of_nodes(self) : [tuple containing (total_nodes, total_leaves)]
Returns tuple containg two elements first one containing total number of nodes
and second one containing total number of leaves in the tree

plot_training_data(self) : [None]
Plots number of nodes v/s accuracies if get_train_data=True

plot_pruning_data(self) : [None]
Plots number of nodes v/s accuracies if get_prune_data=True

2.1 Outputs

After Creating the Decision Tree using this class using command **python Q1.py -f ./data --part a --record_accuracies**, we got the following output (I passed flags to record accuracies during both training and pruning, so the runtimes are including that also. If I don't record accuracies then I get runtime of less than 3 minutes for building tree and less than 1 second to prun the tree).

```
Generated tree in 5.44 minutes
Train Accuracy : 90.808647412421
Test Accuracy : 77.9936025219044
Validation Accuracy : 77.61450027813834

DecisionTreeClassifier:
Parameters :
 get_train_data=True
 record_data_frequency=100
Attributes :
 Total Number of nodes=21007
 Number of leafe nodes=10504
```

And this is the plot that I got for number of nodes v/s Train, Test and validation Accuracy

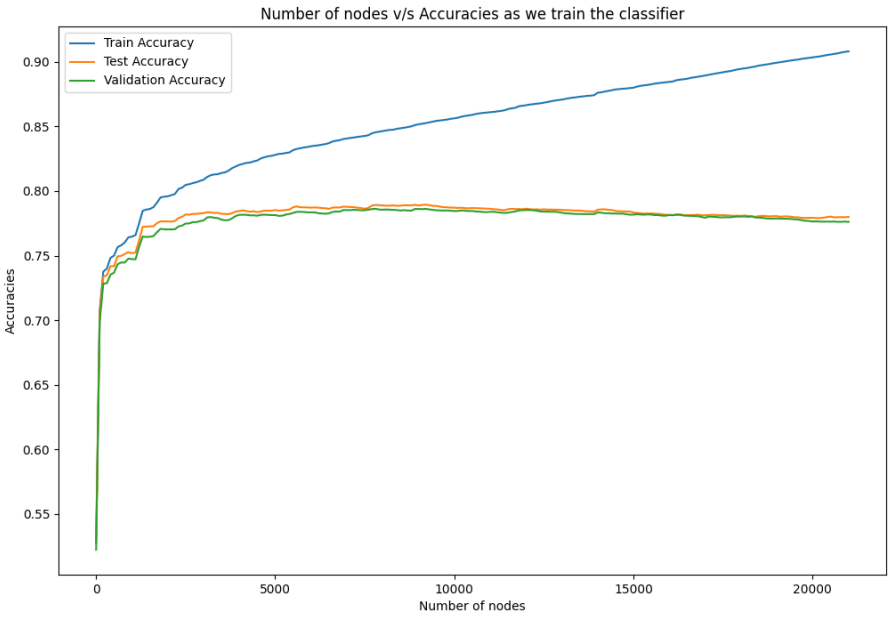


Figure 1: Number of nodes v/s Accuracy during Training

2.2 Observations

- Here we can see that train accuracy that we got is too high. Which is expected as we have fully grown the tree.
- As from the graph we can see that new nodes gets added in the tree(In BFS order) the train accuracy increases whereas Test and Validation accuracy increases until one point but after that it stays more or less, can say slightly decreasing moving forward. Which shows that as we add new nodes we are overfitting to training dataset

3 Part B : Decision Tree Post Pruning

In this part I have implemented reduced error post pruning, which checks validation accuracy for every node in a order and checks if making that node leaf increases the accuracy on validation/pruning dataset or not. If it increases accuracy then we make that node else we keep that node as it is and proceed further and check for next nodes in both cases. The order in which we proceed is post order as it will ensure that before checking for any interal node we will have checked interal node before that every time. So to prune tree we just call the method prune in the given class after training class and it will prune the decision tree learned by that class.

3.1 Outputs

This is how output will look like when we run **python Q1.py -f ./data --part b --record_accuracies**.

```
Pruned tree in 144.50 seconds


After Pruning :
Train Accuracy : 83.48245329377406
Test Accuracy : 79.81549302304019
Validation Accuracy : 82.52364175783423


DecisionTreeClassifier
Parameters :
     get_train_data=False
     get_prune_data=True
     record_data_frequency=100
Attributes :
```

```
Total Number of nodes=4617
Number of leafe nodes=2309
```

And this is the plot that I got for number of nodes v/s Train, Test and validation Accuracy as we prune the nodes. So here in this below table I have summarized accuracy and number of nodes and

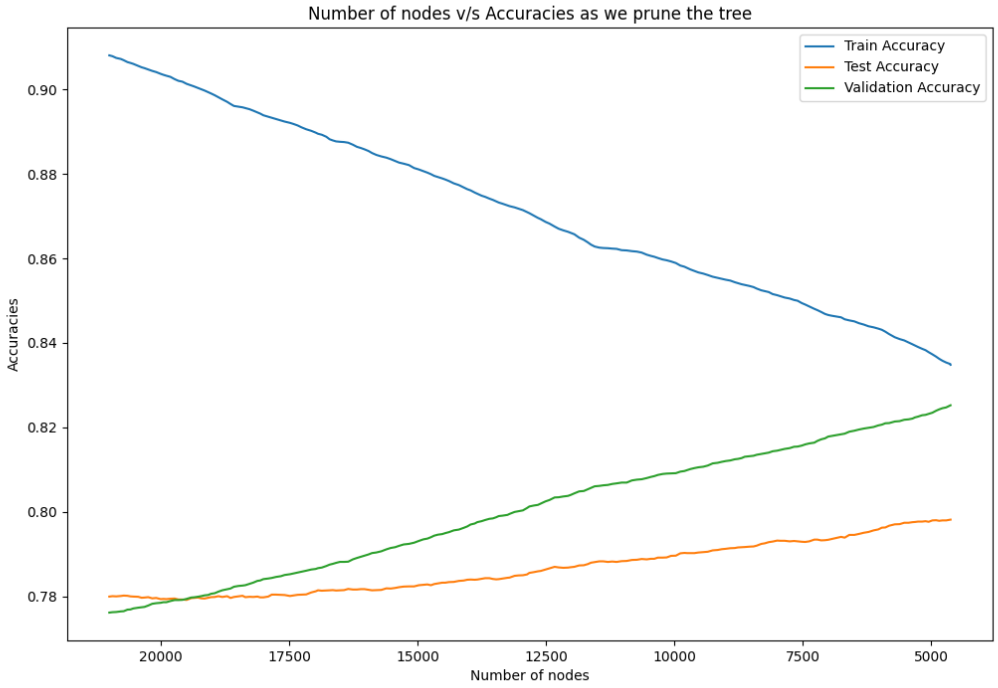


Figure 2: Number of nodes v/s Accuracy during Training

leaves in the tree in both part A and B.

	Before pruning	**After pruning**
Train Accuracy	90.8086	83.4824
Test Accuracy	77.9936	79.8154
Validation Accuracy	77.6145	82.5236
Total Number of nodes	21007	4617
Total number of leaves	10504	2309

Table 2: Various results before and after post pruning

3.2 Observations

- Here we can clearly see that post pruning reduces overfitting as we can see that train accuracy is reduced nut validation and test accuracy is increased. So this technique is helping us reduce

overfitting in fully grown Decision Tree in previous section.w

- As from the plot we can see that as we prune more and more nodes (i.e. number of node decreases), train accuracy is decreasing and test and validation accuracy are increasing.

4 Part C : Random Forests

In this section I have found optimal value of n_estimators, max_features and min_samples_split from range given in the question using oob_score_ as scoring metric to find optimal hyperparameters that optimizes it.

4.1 Testing Methodology

In this question we are deciding optimal hyperparameters which maximizes oob_score_ for the RandomForestClassifier. Now as we know that oob_score_ comes from the out of bag samples which are not included in the bootstrapped samples for given base estimator. So it would be logical to run this test multiple times and then take average of oob_score_ returned by multiple experiments as bootstrapped samples are randomly generated samples. So in my implemented method `GridSearchUsingOOB(param_grid, n_jobs=1, repeat=3)` I have kept a parameter repeat which specifies that how many times we want to run the experiment. So I have got the optimal hyperparameters by repreating this experiment 3 times.

And also I have implemented my own implementation of GridSearch instead of using GridSearchCV as in doing 3-fold or 5-fold CV using GridSearchCV made no sense in our case as we were supposed to take decision based on oob score and we get that while training only and we don't need Validation Dataset for that purpose and hence doing k-fold cross validation won't be required.

4.2 Output

By performing grid search on the given values for three hyperparameters I got following values as optimal hyperparameters.

- **n_estimators : 450**
- **max_features : 0.3**
- **min_samples_split : 10**

These are the accuracy that I got using Random Forest Classifier.

- **Train Accuracy : 88.1028**
- **Validation Accuracy : 80.6183**
- **Test Accuracy : 80.7751**
- **OOB score : 81.0594**

From these accuracy we can see that test accuracy is better in the random forest classifier which is expected as we know that Random Forest reduces variance.

5 Part D : Random Forests - Parameter Sensitivity Analysis

5.1 Methodology

In this part I have fixed two of the three hyperparameters that we have been tweaking and changed one parameter in the specified range and recorded Train, Test and Validation Accuracy. As we did in the previous part, in this part also I have trained model three times and then have taken the average of these accuracy that we got. And then I have plotted the train, test and validation separately as well as in the same plot to analyse how does these accuracy change as we change the hyper-parameter.

These are the ranges that I have took for all the three hyperparameters in part D (Here both end points are inclusive):

- n_estimators : 10 to 50 in range of 10 + 100 to 1000 in range of 50
- max_features : 0.02 to 0.98 in range of 0.04
- min_samples_split : 2 to 40 in range of 2

5.2 Output

5.2.1 changing n_estimators

Here are the plots that we got while changing n_estimators and keeping rest of two hyperparameters to their optimal values.

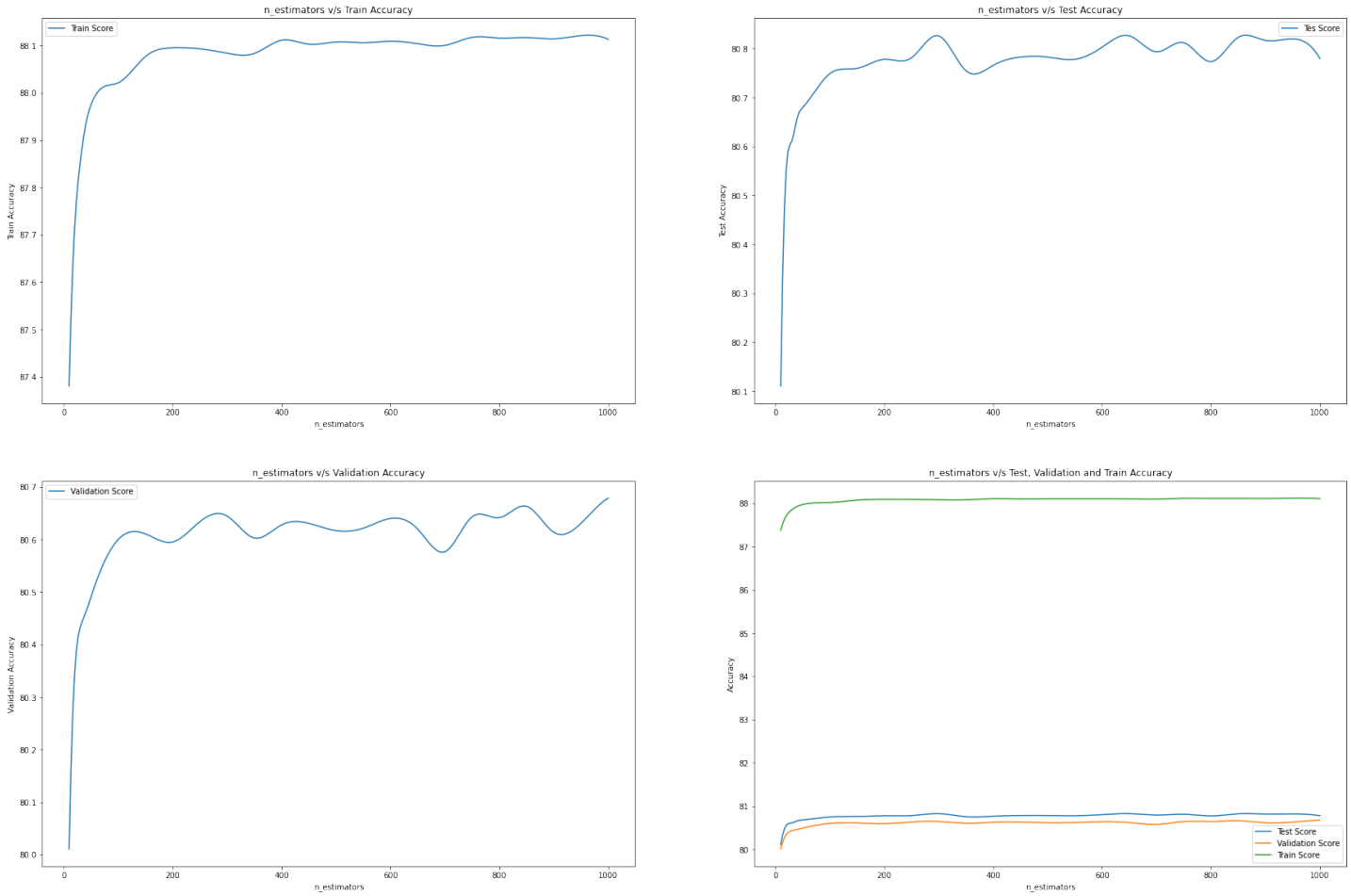


Figure 3: Sensitivity analysis for n_estimators

From the above plots we can observe the following

- We can see that as number of base learners increases train accuracy increases drastically from 10 to 50 and after that it almost saturates
- As Number of base learners increases validation and test accuracies also increases
- Hence we can conclude that when number of base learner is small we cannot reduce variance hence we do not get high validation and test accuracy

5.2.2 changing max_features

Here are the plots that we got while changing max_features and keeping rest of two hyperparameters to their optimal values.

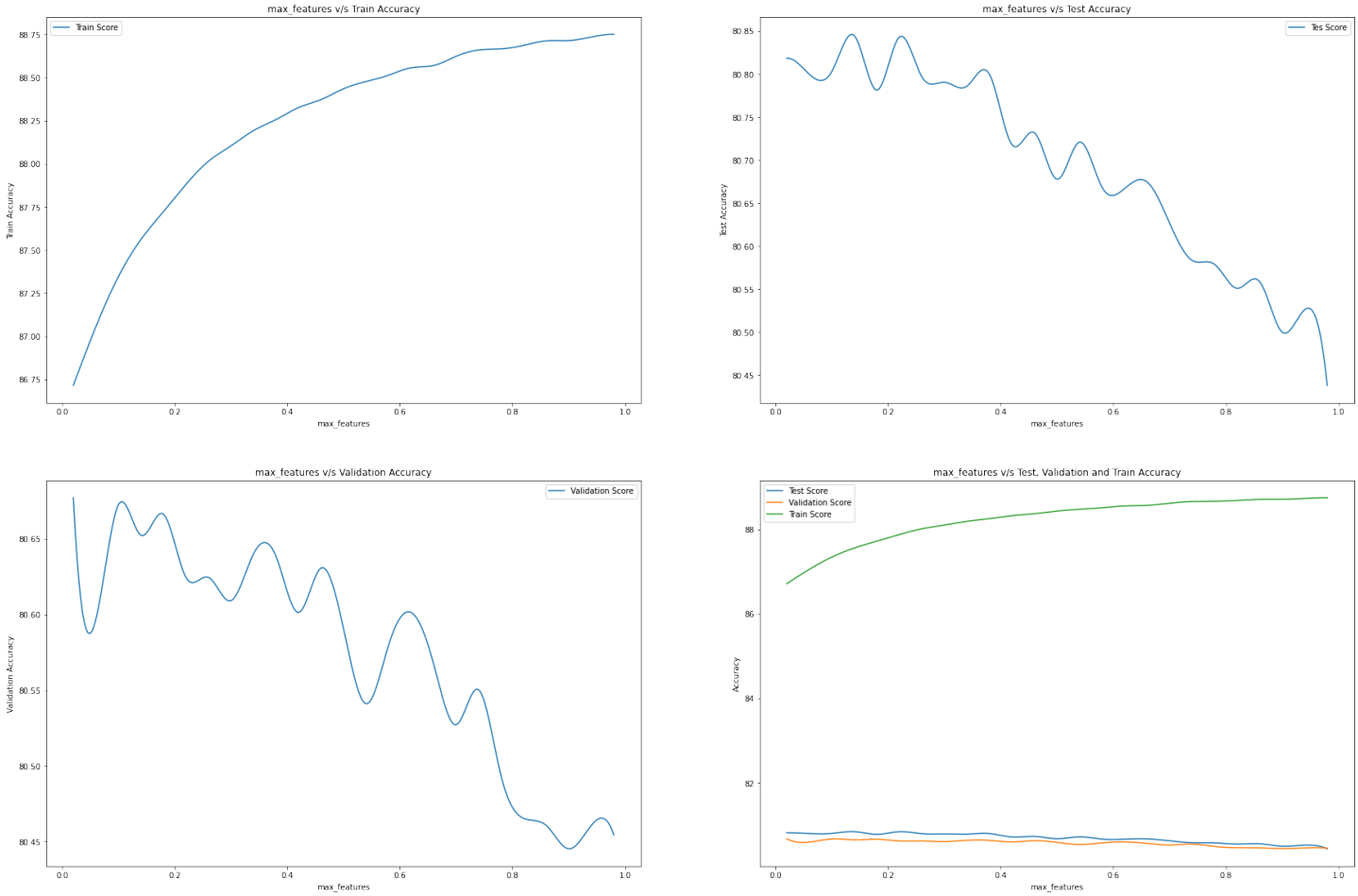


Figure 4: Sensitivity analysis for max_features

From the above plots we can observe the following

- We can see that as max_features increases train accuracy increases. This is expected as increasing max_features we will include more features in bootstrapped samples hence we will fit our training dataset in better way.
- As max_features increases validation and Test accuracy drops.
- Hence we can conclude that when we increase max_features we overfit as validation accuracy is dropping and train accuracy is increasing as we add more and more features.

5.2.3 changing min_samples_split

Here are the plots that we got while changing min_samples_split and keeping rest of two hyperparameters to their optimal values.

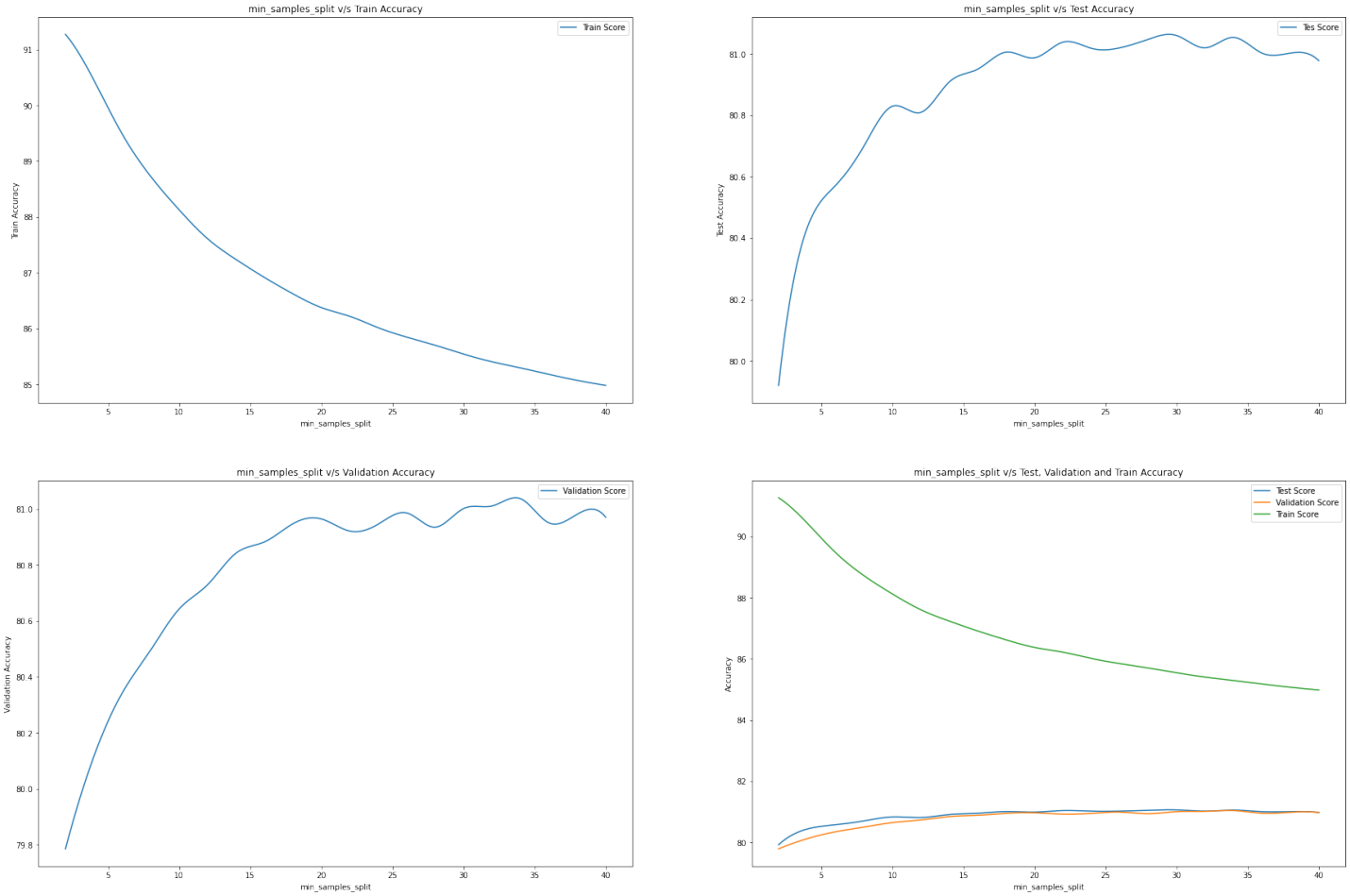


Figure 5: Sensitivity analysis for min_samples_split

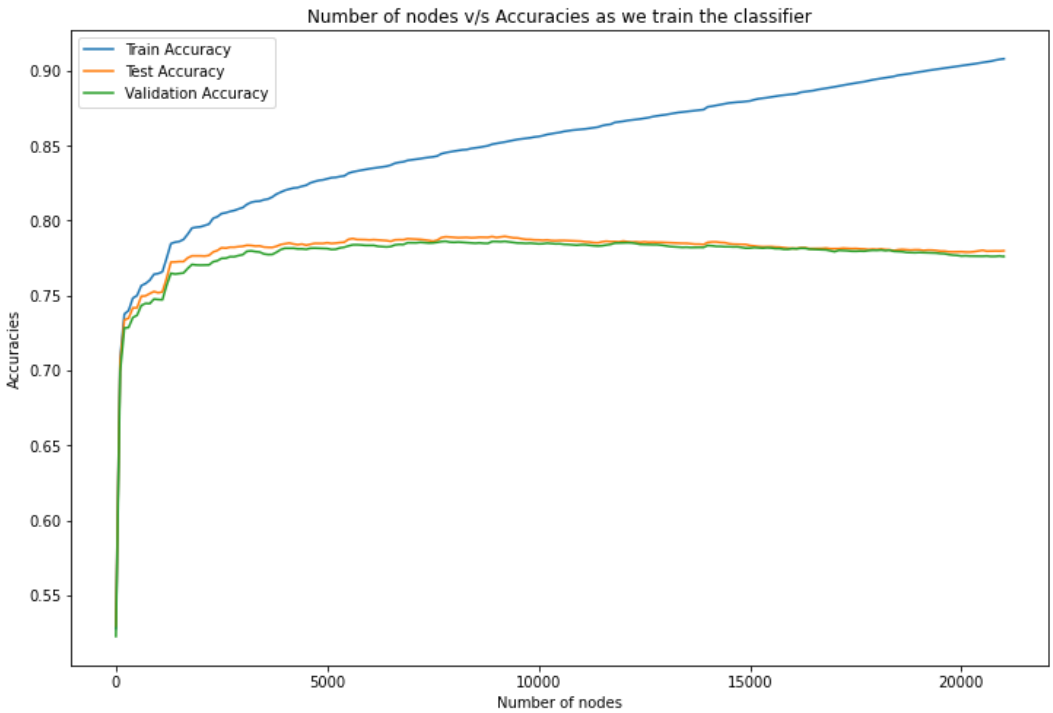
From the above plots we can observe the following

- We can see that as min_samples_split increases train accuracy decreases. This is expected as increasing min_samples_split Depth of the base trees will be reduced as we increase the hyperparameter.
- As min_samples_split increases we can see test and validation accuracy increasing.
- Hence we can say that our model is more generalized as we increase min_samples_split but if we take this value too high then we might underfit if trees become too shallow.

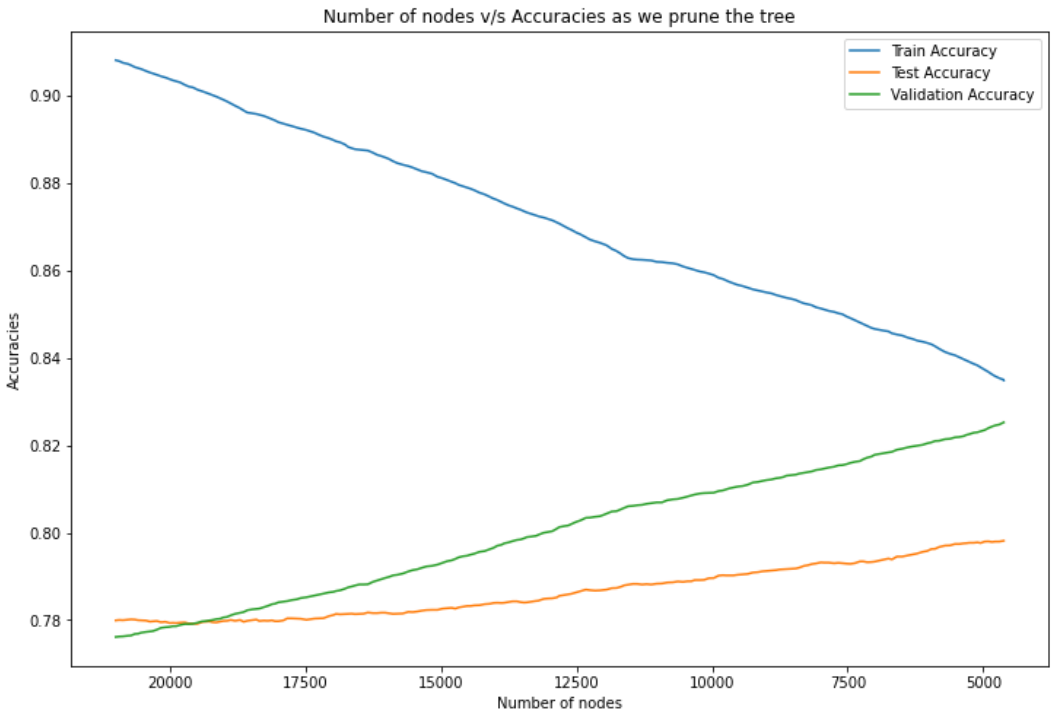
6 Output of the submitted file

Here I have also attached the how the output looked like when I ran `python Q1.py -f /content/drive/My Drive/ML/Ass3a --part abcd --record_accuracies` command in google colab.

```
Shape of X_train (64713, 263)
Shape of X_test (21571, 263)
Shape of X_val (21572, 263)


**********Running Part A**********


Generated tree in 6.62 minutes
Train Accuracy : 90.808647412421
Test Accuracy : 77.9936025219044
Validation Accuracy : 77.61450027813834


DecisionTreeClassifier

Parameters :
        get_train_data=True
        get_prune_data=True
        record_data_frequency=100

Attributes :
        Total Number of nodes=21007
        Number of leafe nodes=10504
```

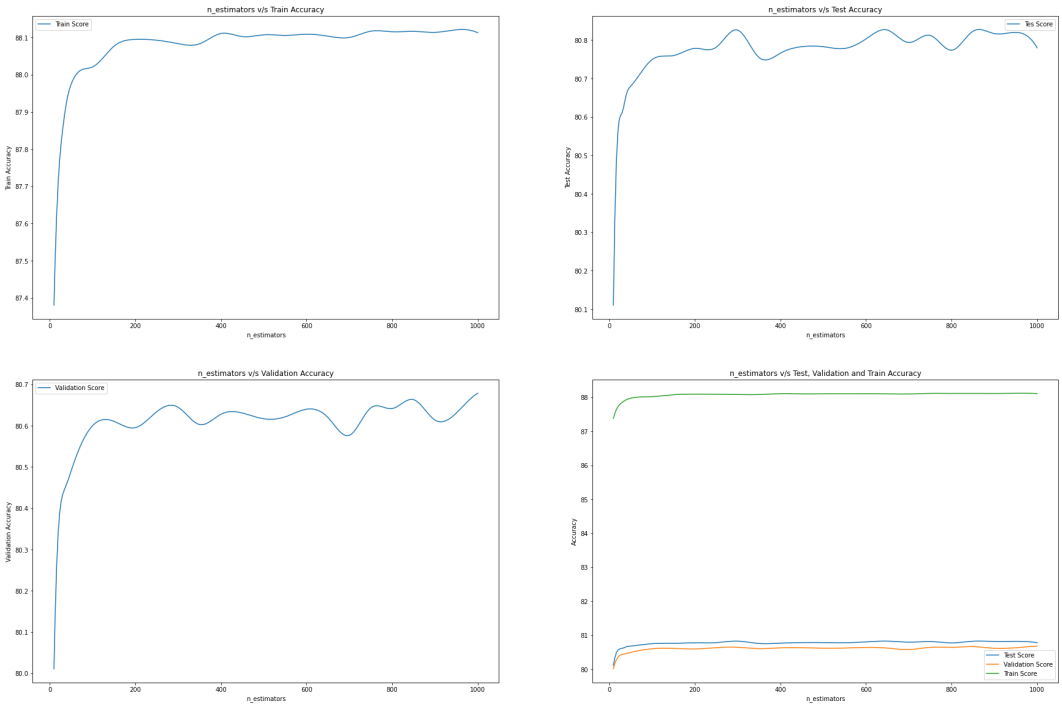


```
**********Running Part B**********


Pruned tree in 195.51 seconds

After Pruning :
Train Accuracy : 83.48245329377406
Test Accuracy : 79.81549302304019
Validation Accuracy : 82.52364175783423


DecisionTreeClassifier

Parameters :
        get_train_data=True
        get_prune_data=True
        record_data_frequency=100

Attributes :
        Total Number of nodes=4617
        Number of leafe nodes=2309
```

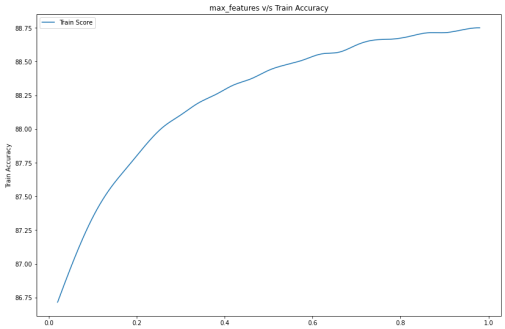


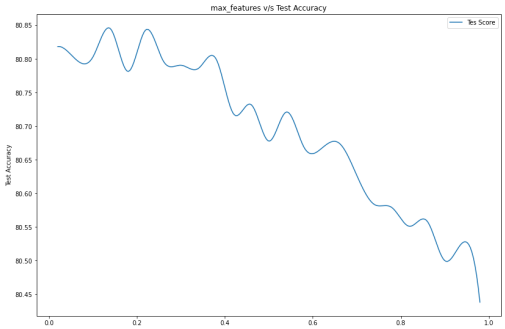

```
**********Running Part C**********


100%|██████████| 125/125 [2:03:49<00:00, 59.44s/it]
```

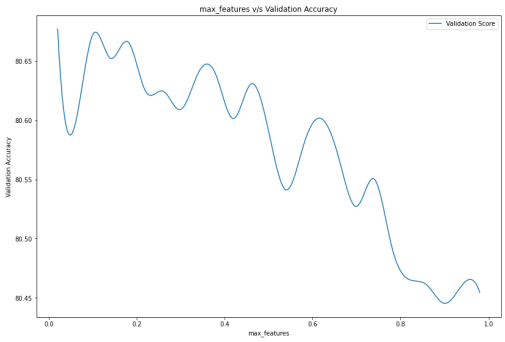
```
optimal Classifier :

  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                         criterion='gini', max_depth=None, max_features=0.3,
                         max_leaf_nodes=None, max_samples=None,
                         min_impurity_decrease=0.0, min_impurity_split=None,
                         min_samples_leaf=1, min_samples_split=10,
                         min_weight_fraction_leaf=0.0, n_estimators=450,
                         n_jobs=-2, oob_score=True, random_state=None, verbose=
0,
                         warm_start=False)
These are the hyperparametes for the optimal model
 n_estimators : 450
 max_features : 0.3
 min_samples_split : 10
Accuracies using Optimal classifier:
 Train accuracy : 0.8810285414058999
 Test accuracy : 0.8077511473737888

 Validation accuracy : 0.8061839421472279
 OOB accuracy : 0.8105944709718295


**********Running Part D**********


100%|██████████| 24/24 [17:19<00:00, 43.33s/it]
```

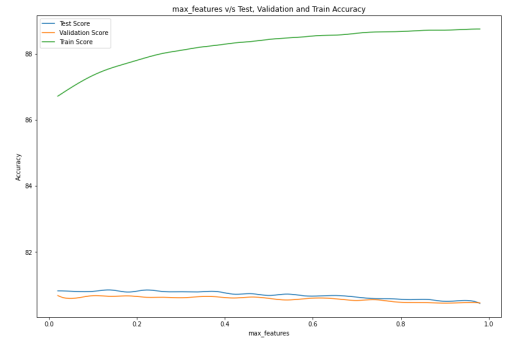



```
100%|██████████| 25/25 [28:21<00:00, 68.06s/it]
```









100%|[redacted]| 20/20 [16:28<00:00, 49.41s/it]