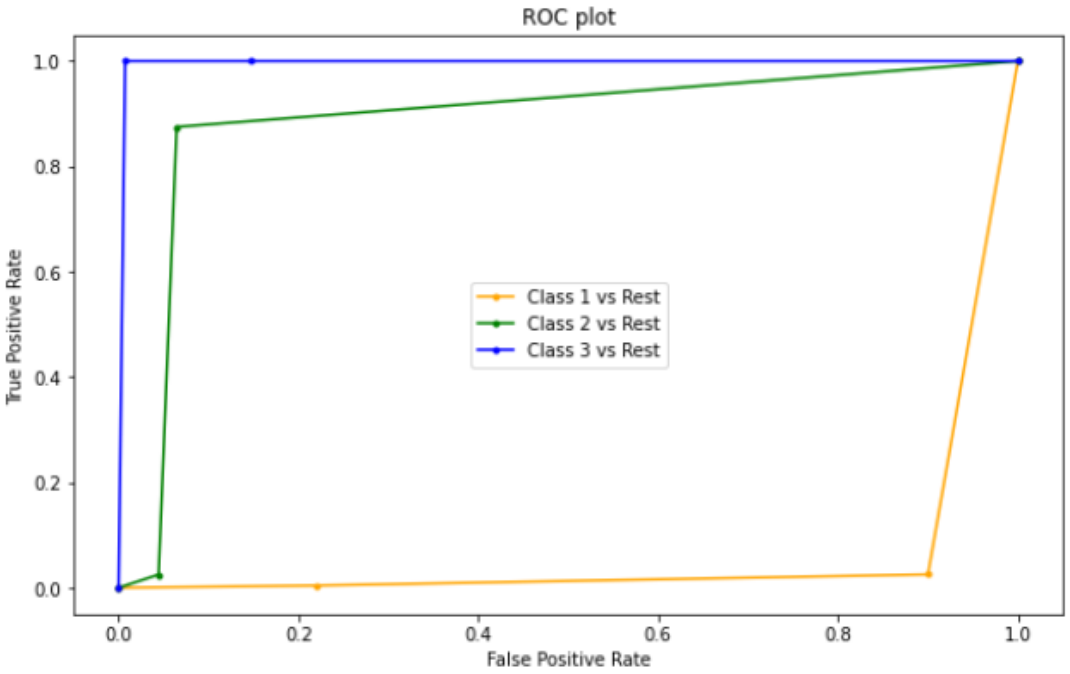
Pre-processing Steps:

1. Loaded the ‘data_1’ dataset
2. Checked missing values, and found that the data doesn’t have any null values
3. Removed duplicate samples to have a generalised dataset
4. Split the dataset into features and output class labels
5. Normalised the feature values as the scale of many features vary a lot
6. Split the feature and output sets into training and test sets (80:20 split)
7. **Assumptions:** The outliers in the dataset are a natural part of the population.

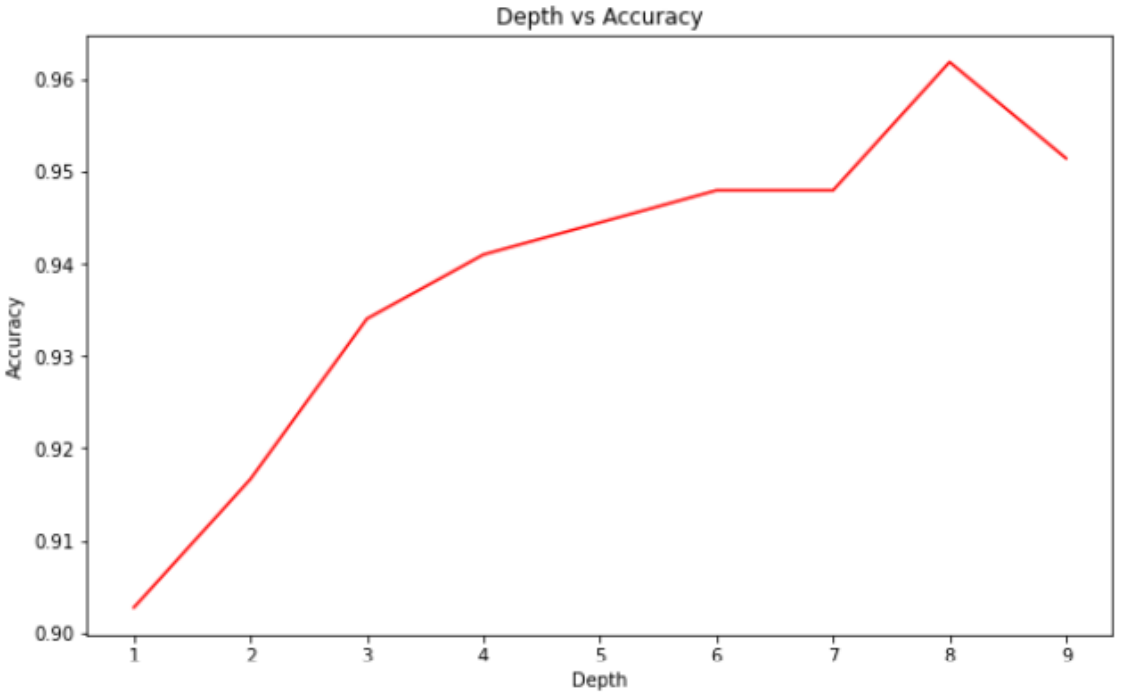
A (Training):

Question 1:

```
Accuracy:  0.9583333333333334
Recall Score:  0.9415966386554621
Precision Score:  0.8946770529049011
```



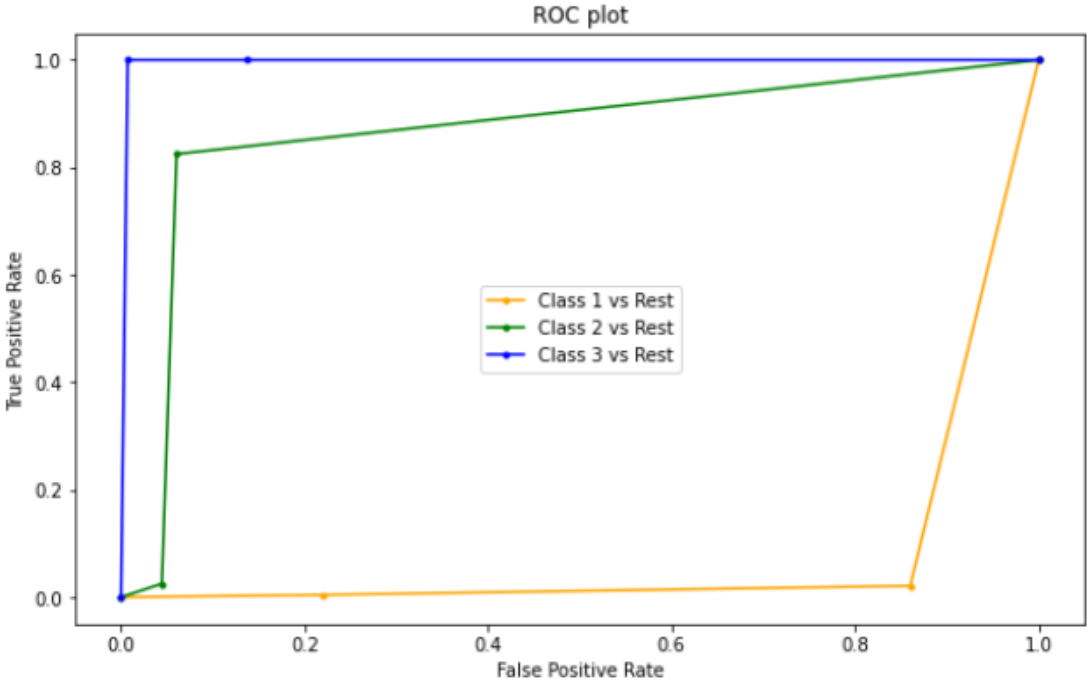
Question 2:


Depth vs Accuracy
Accuracy
0.96
0.95
0.94
0.93
0.92
0.91
0.90
1
2
3
4
5
6
7
8
9
Depth

Question 3:

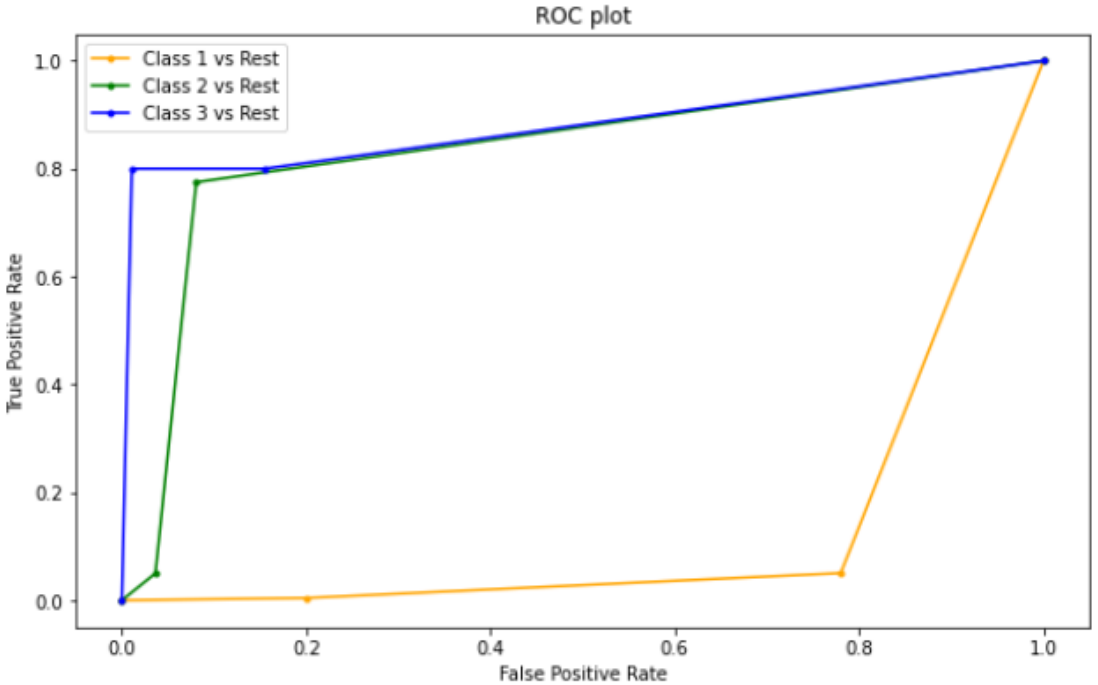
Criterion: Entropy

```
Accuracy:  0.9548611111111112
Recall Score:  0.9263305322128851
Precision Score:  0.8976851851851851
```



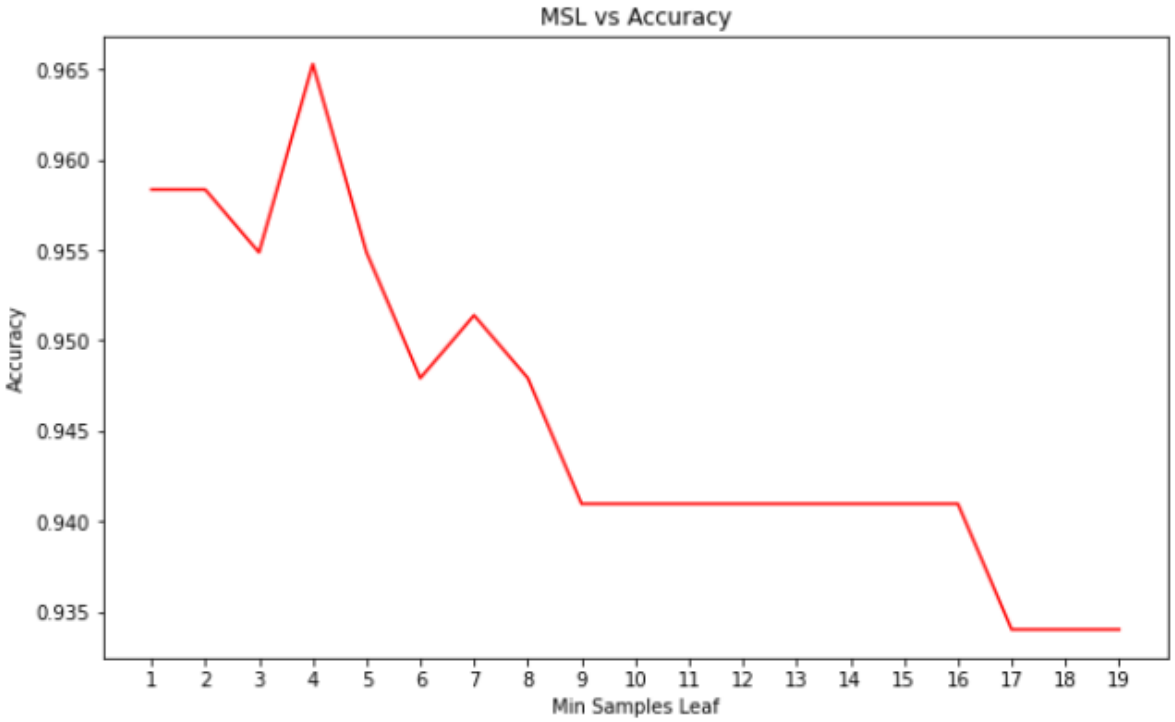
Splitter: Random

```
Accuracy:  0.9131944444444444
Recall Score:  0.8248599439775909
Precision Score:  0.801953075054341
```



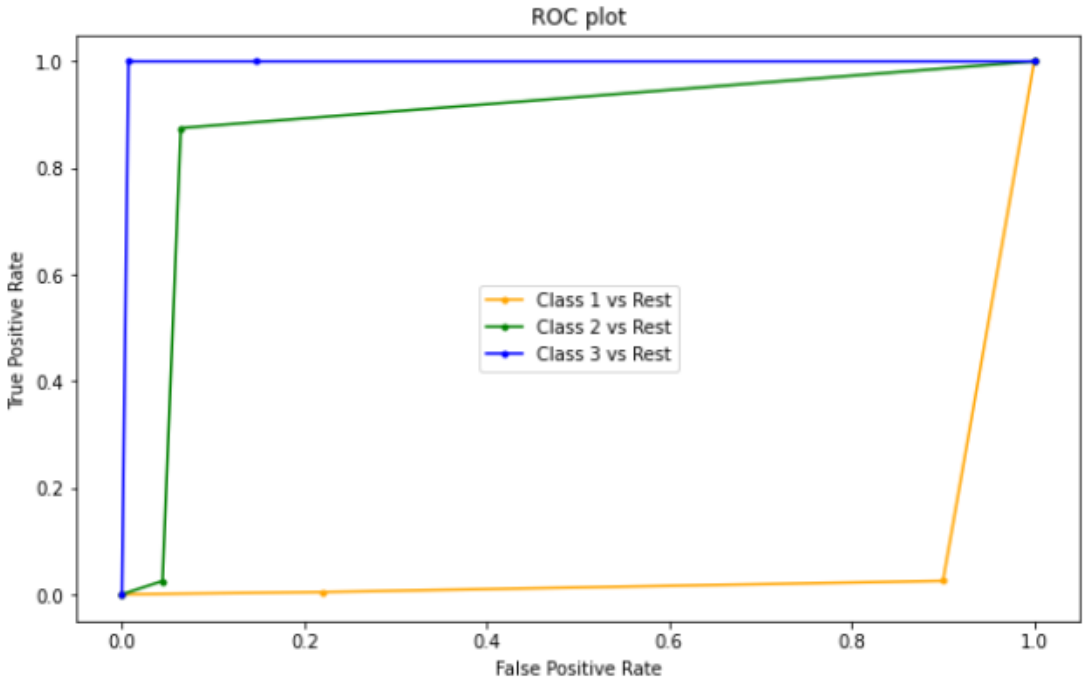
Min Samples Split(MSS):

MSS vs Accuracy



MSS = 4

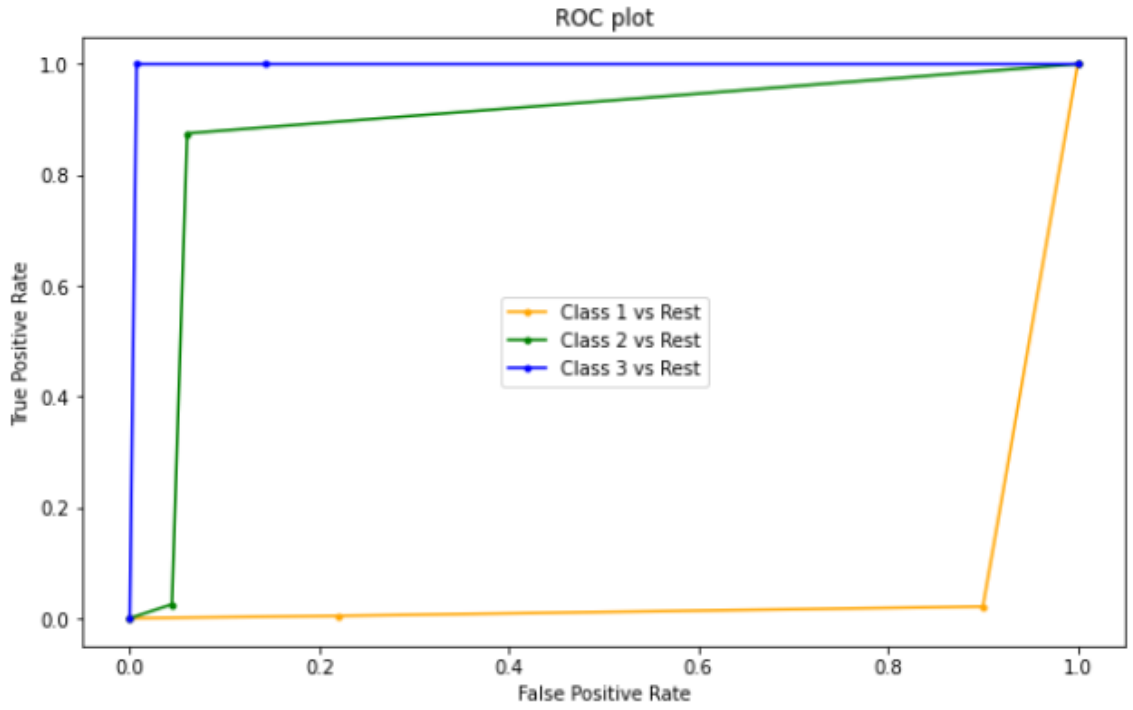
```
Accuracy:  0.9583333333333334
Recall Score:  0.9415966386554621
Precision Score:  0.8946770529049011
```



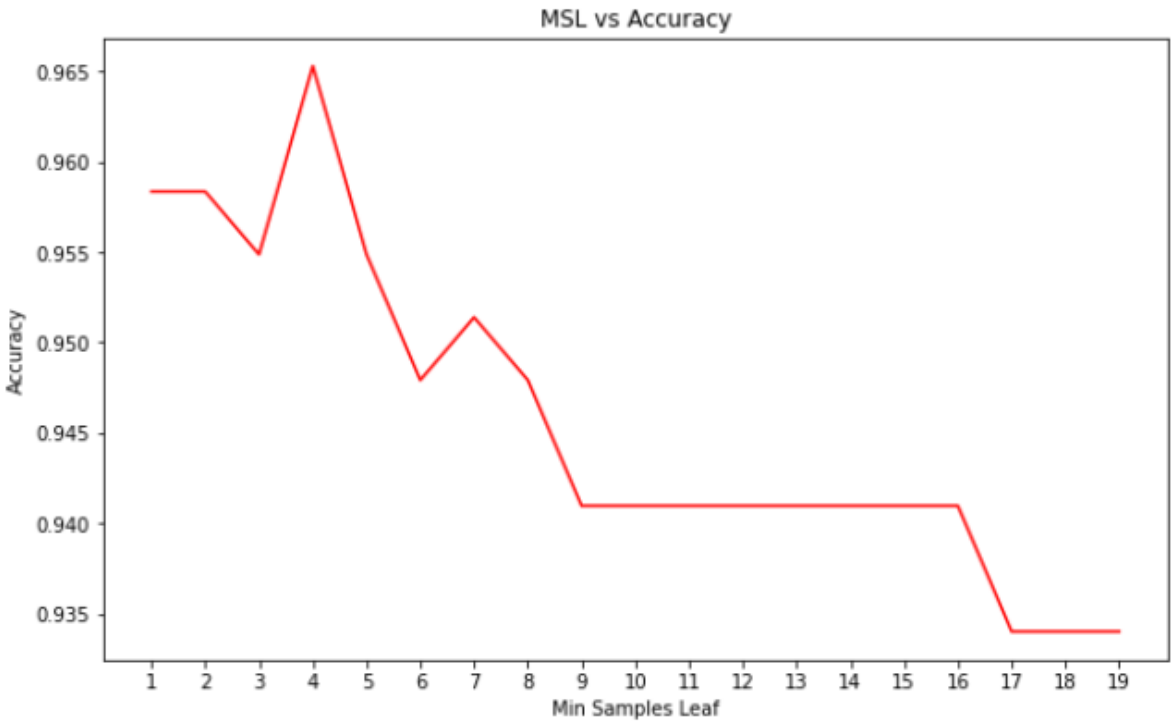
Max Depth:

Max Depth = 8

```
Accuracy:  0.9618055555555556
Recall Score:  0.9429971988795517
Precision Score:  0.9023539240257507
```

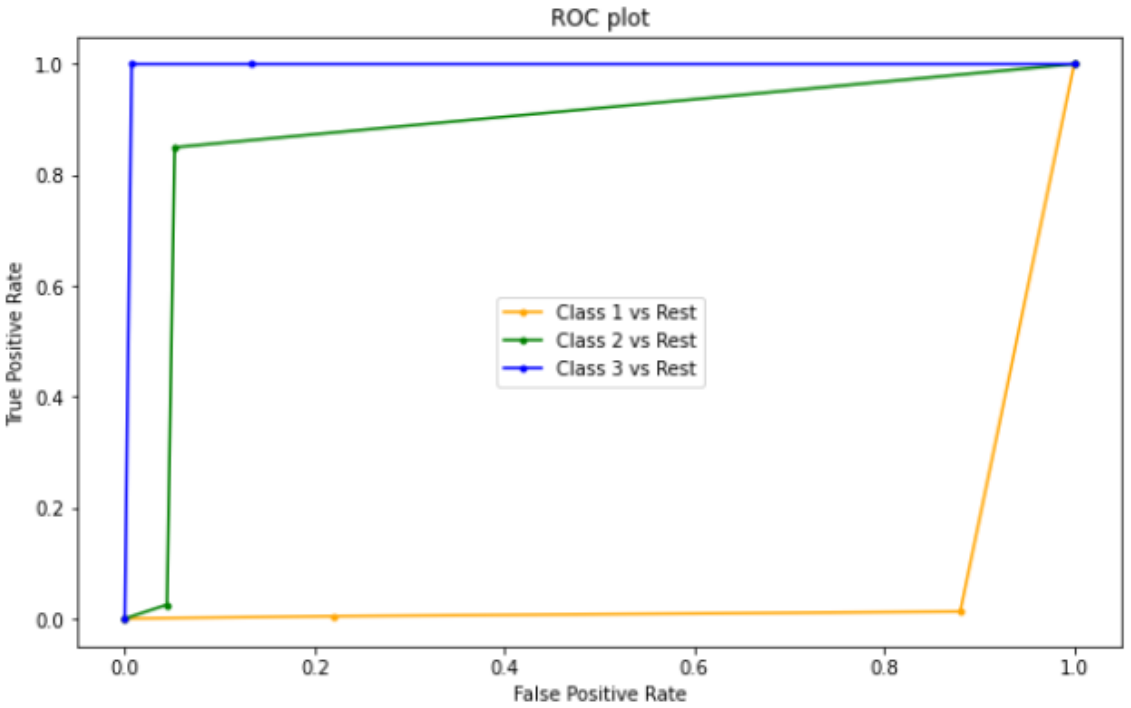


Min Samples Leaf (MSL) :



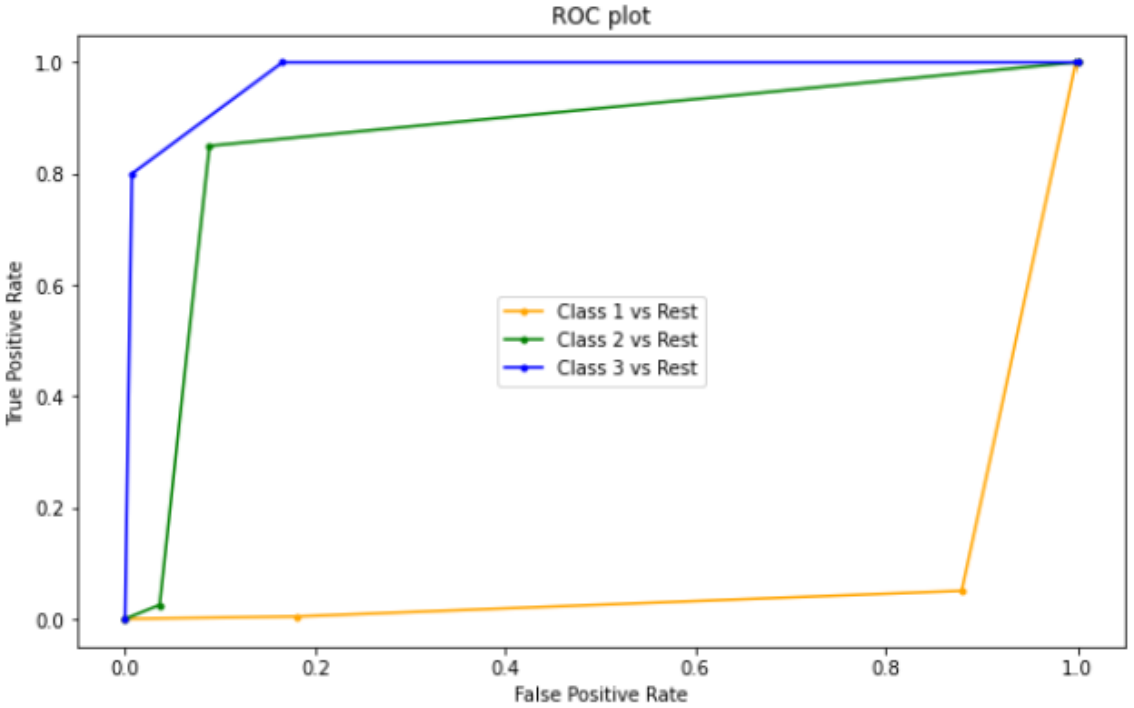
MSL = 4

```
Accuracy:  0.9652777777777778
Recall Score:  0.9374649859943979
Precision Score:  0.9170980702101034
```

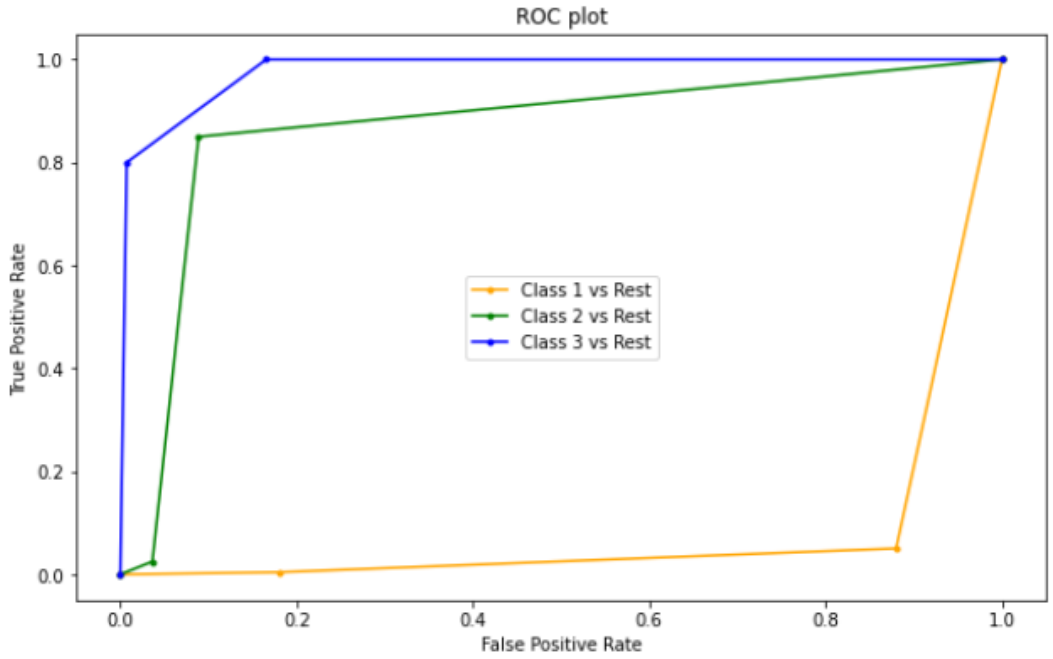


Max Feature:

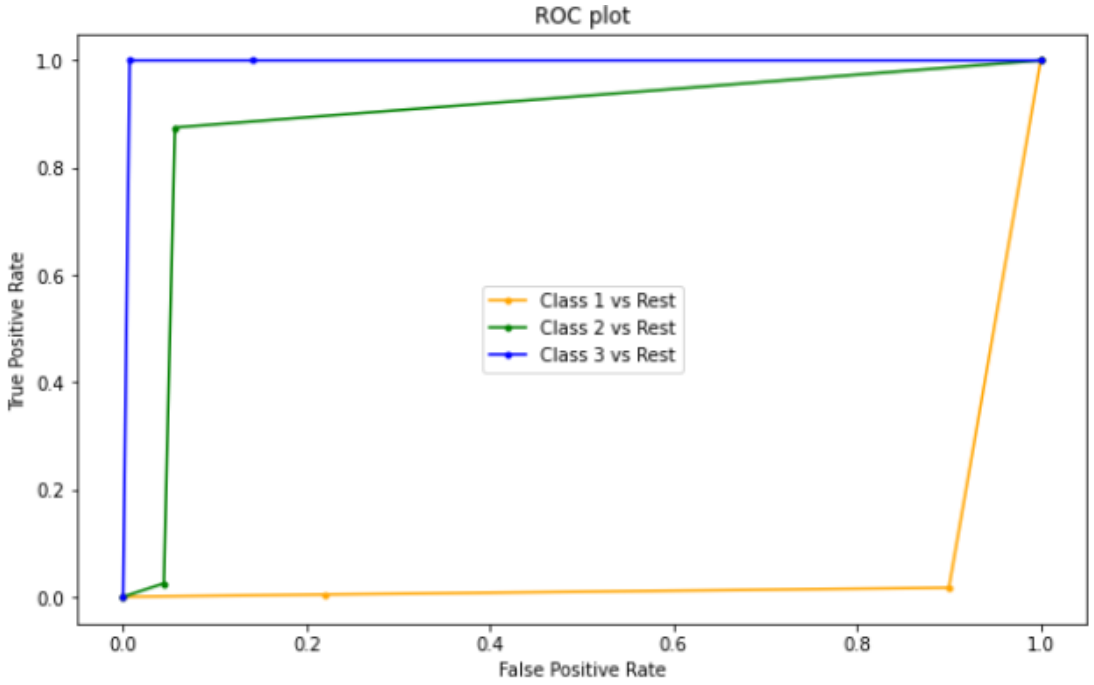
Max Feature = log



Max Feature = sqrt

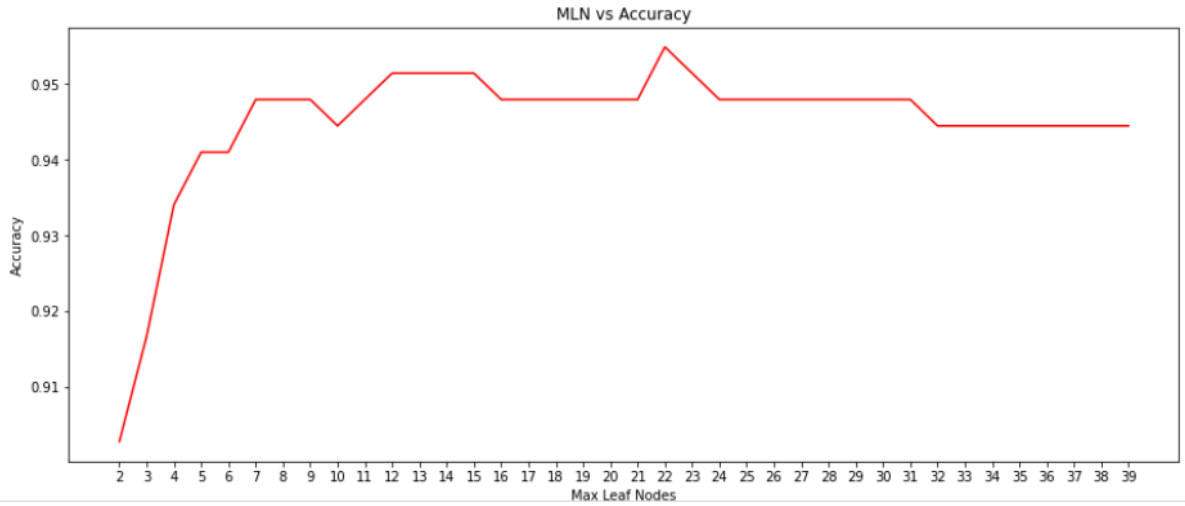


Class Weight: Balanced



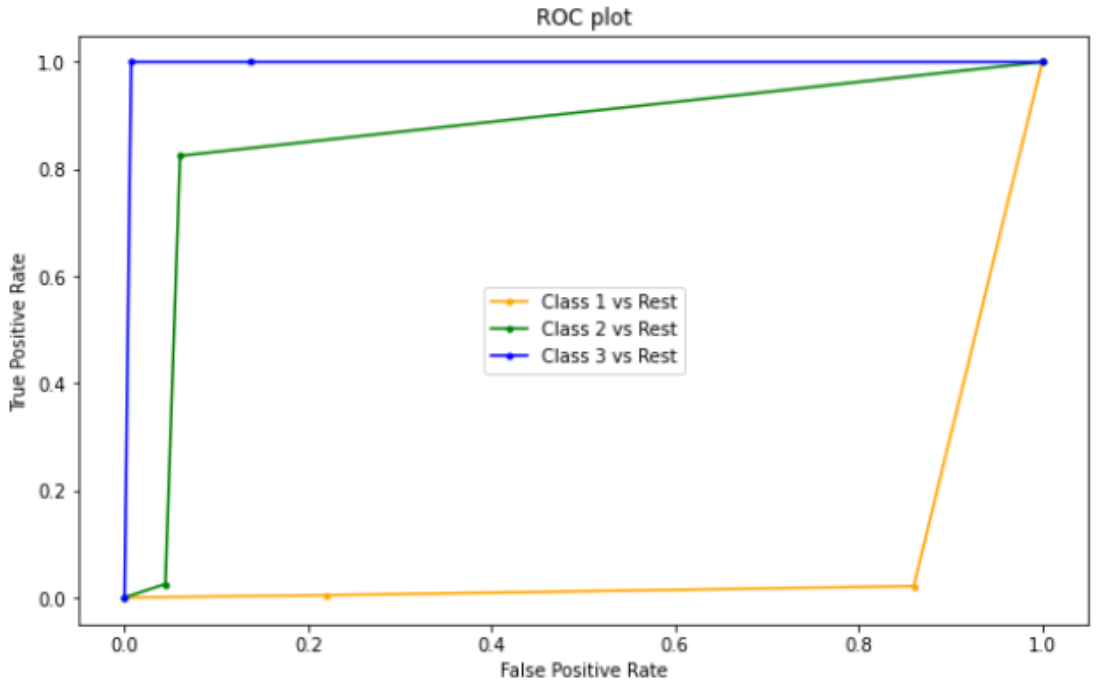
Max Leaf Node(MLN):

MLN vs Accuracy



MLN = 22

```
Accuracy:  0.9548611111111112
Recall Score:  0.9263305322128851
Precision Score:  0.8976851851851851
```



Observation Table:

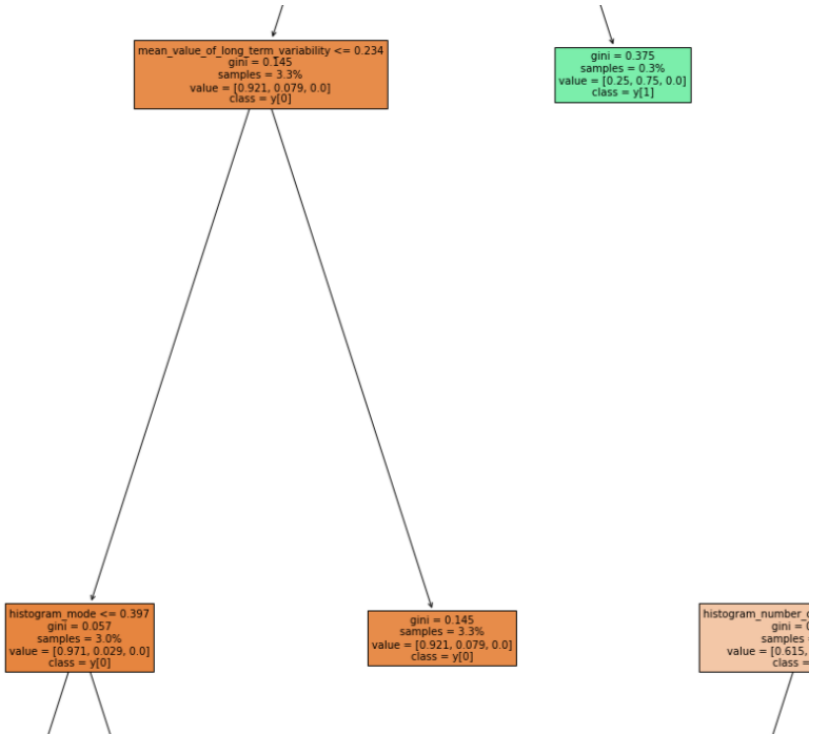
Changed Hyperparameter	Accuracy	Recall	Precision	Reasoning
Base Model	**0.958**	**0.942**	**0.895**	**-**
Criterion: Entropy	0.955	0.927	0.898	The scores remain almost the same, as both gini index and entropy give the same performance, though the decision tree might be different.
Splitter: Random	0.913	0.825	0.802	Since it chooses random best attributes for splitting, it might leave out important features for splitting which results in large complex trees, which will decrease the performance.
Min Samples Split = 4	0.958	0.942	0.895	Small values of minimum sample splits (say 2) tend to overfit the data, while large values (say 10) tend to underfit the data.
Max Depth = 8	0.962	0.943	0.902	Small values of max depths (say 3 or 4) tend to underfit the data, while large values tend to overfit the data. (as shown in Q2 as well). Since in the base model, we haven't specified the depth, it increases until we have pure leaf nodes, thus overfitting the data. Hence max depth = 8 gives a better performance.
Min Samples Leaf = 4	0.965	0.937	0.917	Small values of minimum sample splits (say 2) tend to overfit the data, while large values (say 10) tend to underfit the data. In the base model, we take the min samples required to be at a leaf node to be 1, thus overfitting the data. Hence min samples leaf = 4 gives a better performance.
Max features: log2	0.927	0.858	0.830	Since it doesn't consider all the features for splitting, it might leave

				out important features for splitting which results in large complex trees, which will decrease the performance.
Max features: sqrt	0.927	0.858	0.830	Since it doesn't consider all the features for splitting, it might leave out important features for splitting which results in large complex trees, which will decrease the performance.
Class Weight: Balanced	0.965	0.944	0.910	When we use balanced class weight then it automatically adjusts weights inversely proportional to class frequencies which give equal weightage to each class on the basis of frequency which reduces splitting error results in increase in all parameters compared to base model.
Max Leaf Nodes=22	0.955	0.926	0.898	When we fix max leaf nodes (say 22) then instead of splitting an unlimited number of nodes (default behaviour) it stops early and this results in a change of accuracy.

B (Pruning):

Question 1:

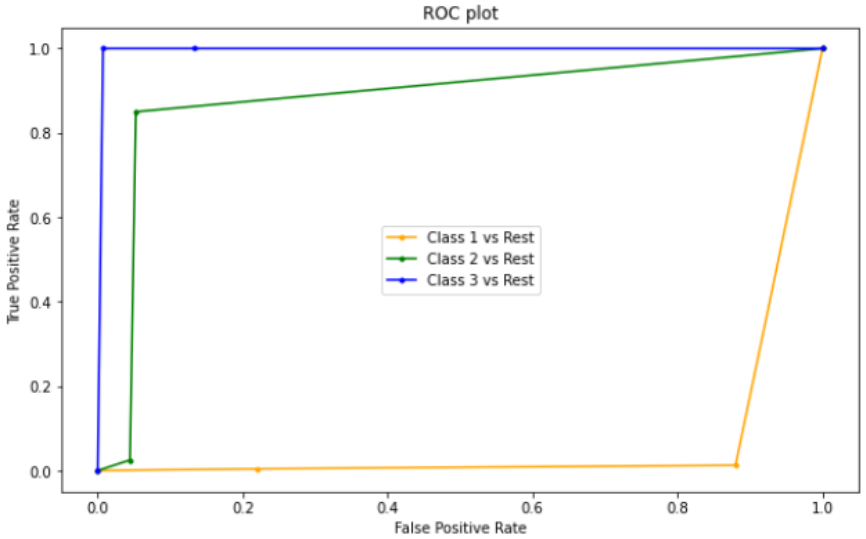
When we are removing a random node from the tree we will remove that node and duplicate the parent node in its place.
Assumption -> Removed node will not be root Node.



DT_B_1 performance:

We are removing node of index = 20

```
Accuracy:  0.9652777777777778
Recall Score:  0.9374649859943979
Precision Score:  0.9170980702101034
```

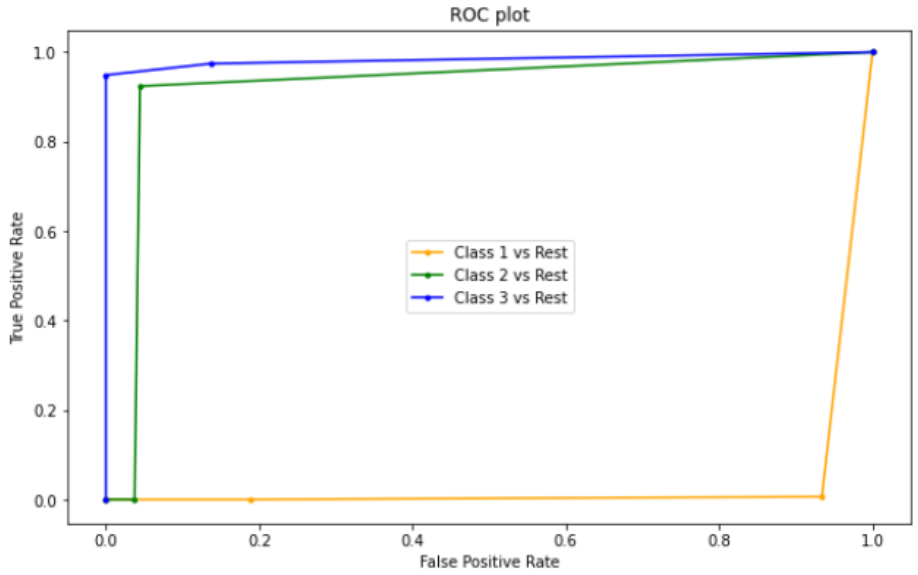


We are removing a node and assigning its parent value to node due to which no extra node will be removed which is resulting in the model generalization and hence our performance will almost remain the same.

Question 2:

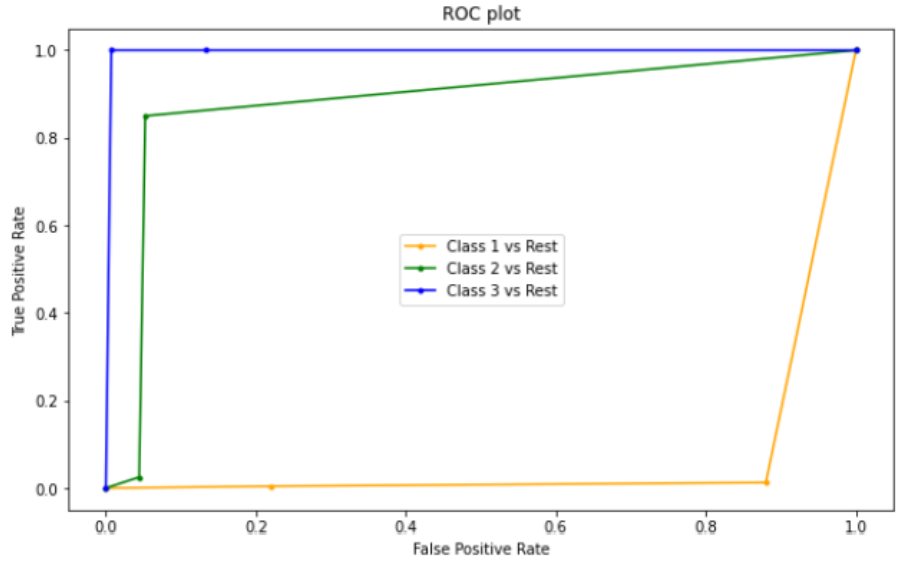
DT_A Training:

```
Training Set:
Accuracy:  0.9826086956521739
Recall Score:  0.9554908913231067
Precision Score:  0.9802355664488017
```



DT_A Testing:

```
Testing Set:
Accuracy:  0.9652777777777778
Recall Score:  0.9374649859943979
Precision Score:  0.9170980702101034
```

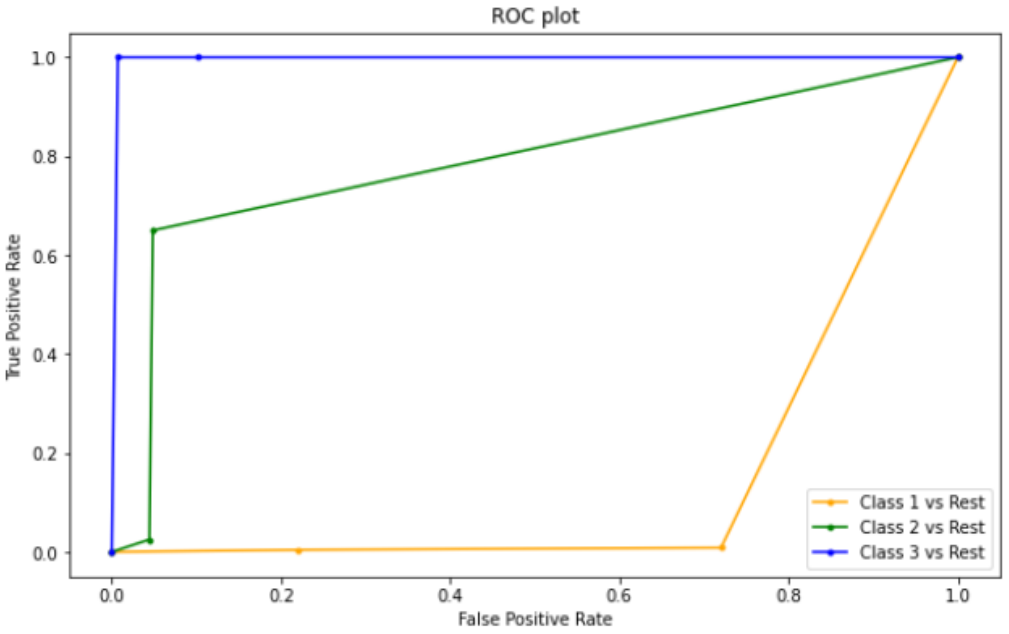


Cost Complexity Pruning:

ccp_alpha=0.004384

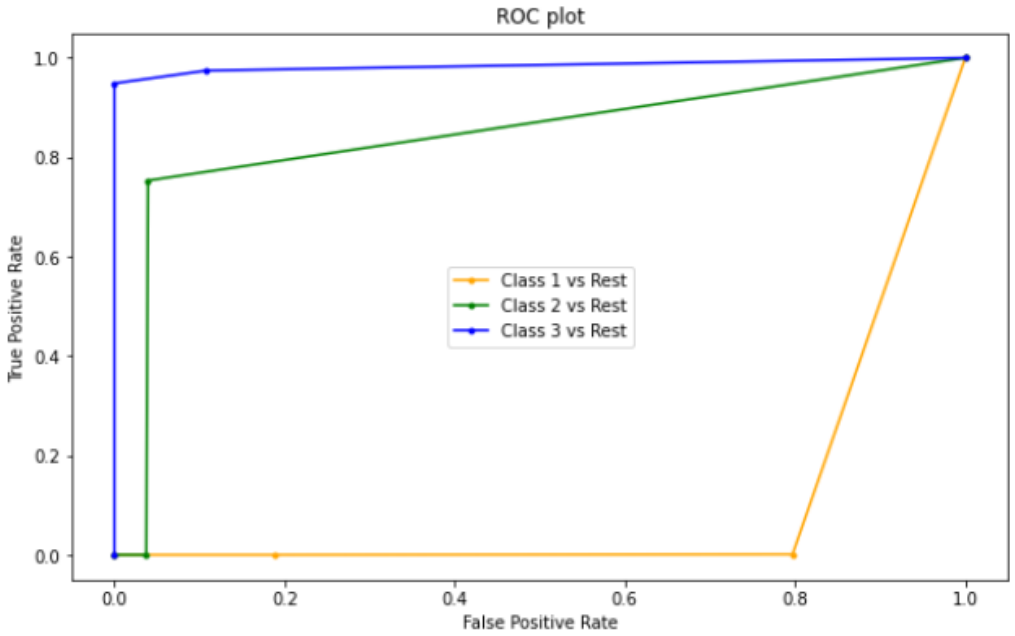
DT-B-2-CC Testing:

```
Testing Set:
Accuracy:  0.9409722222222222
Recall Score:  0.8721988795518207
Precision Score:  0.912957264957265
```



DT-B-2-CC Training:

```
Training Set:
Accuracy:  0.9634782608695652
Recall Score:  0.9002777292455457
Precision Score:  0.981049497911668
```



In base model , DT_A we have trained our classifier using alpha value of 0 which signifies that it can create as much node as possible to reduce the impurity but this results in overfitting

but during cost complexity pruning we passing alpha value more than 0 hence it will not create unnecessary splitting which is not contributing much to the change in the overall impurity of the model.
And since higher splitting ensures that classes are categorized correctly(i.e. accuracy is more) so as we increase alpha value from 0 it results in decreased performance(mostly accuracy).

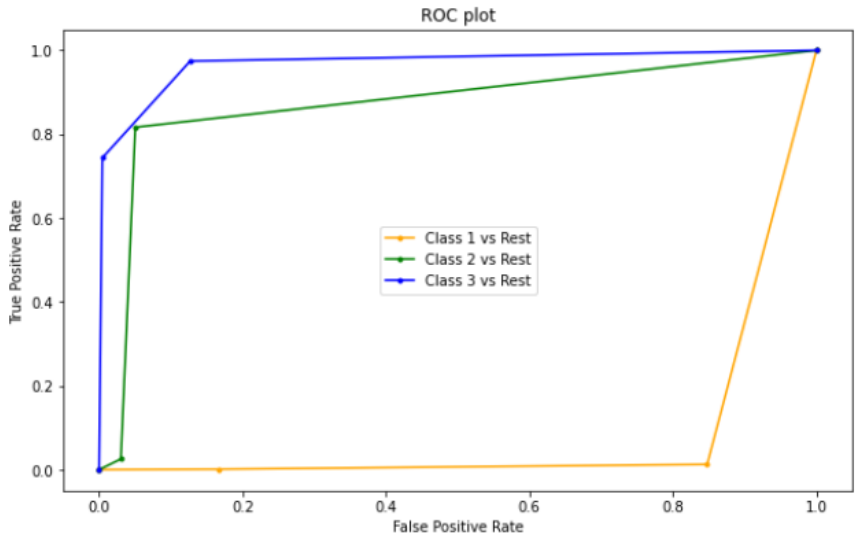
Preprunning:

Parameter Values :

max_depth=9,
max_features='sqrt',
max_leaf_nodes=38,
min_samples_leaf=4,
min_samples_split=3,
splitter='best'

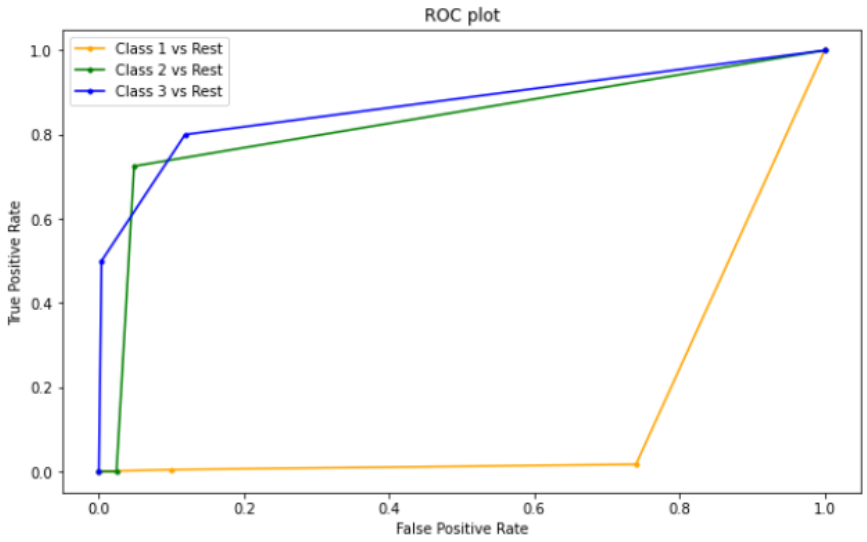
DT-B-2-XX Training:

```
Training Set:
Accuracy:  0.9521739130434783
Recall Score:  0.8407123895920102
Precision Score:  0.8947047194885624
```



DT-B-2-XX Testing :

```
Testing Set:
Accuracy:  0.9305555555555556
Recall Score:  0.7360644257703081
Precision Score:  0.8697577276524645
```

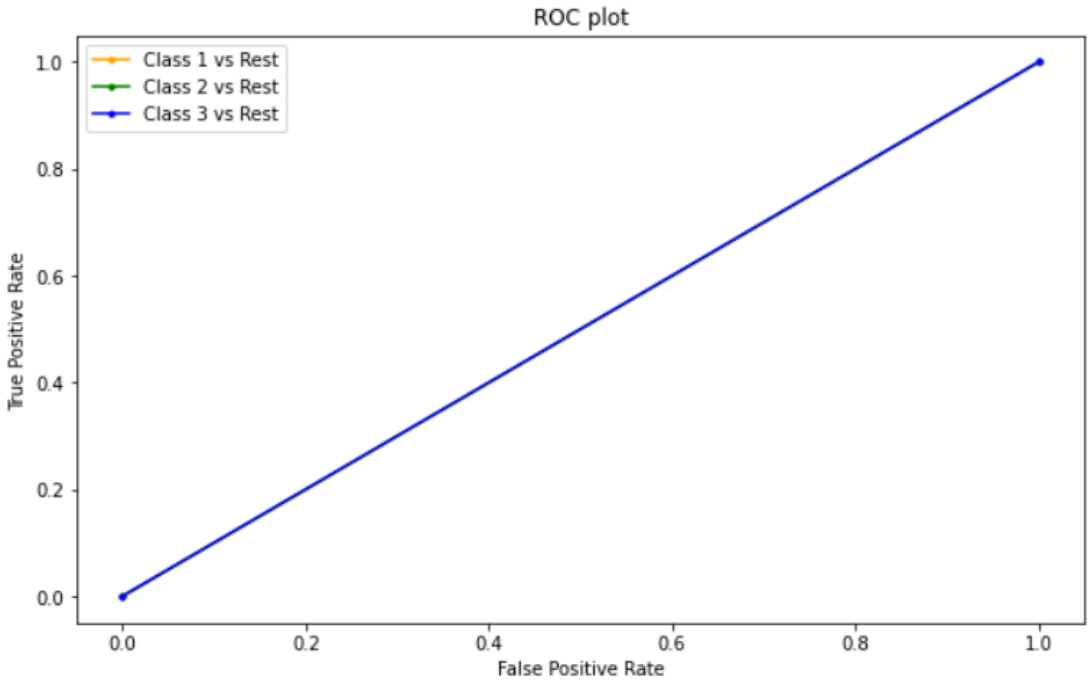


In pre-pruning we are choosing multiple parameters(above shown) and checking optimum value for model parameters using GridSearchCV. Due to uses of multiple parameters we have pruned our model a lot which results in decrease of performance compared to base model.

Question 3:

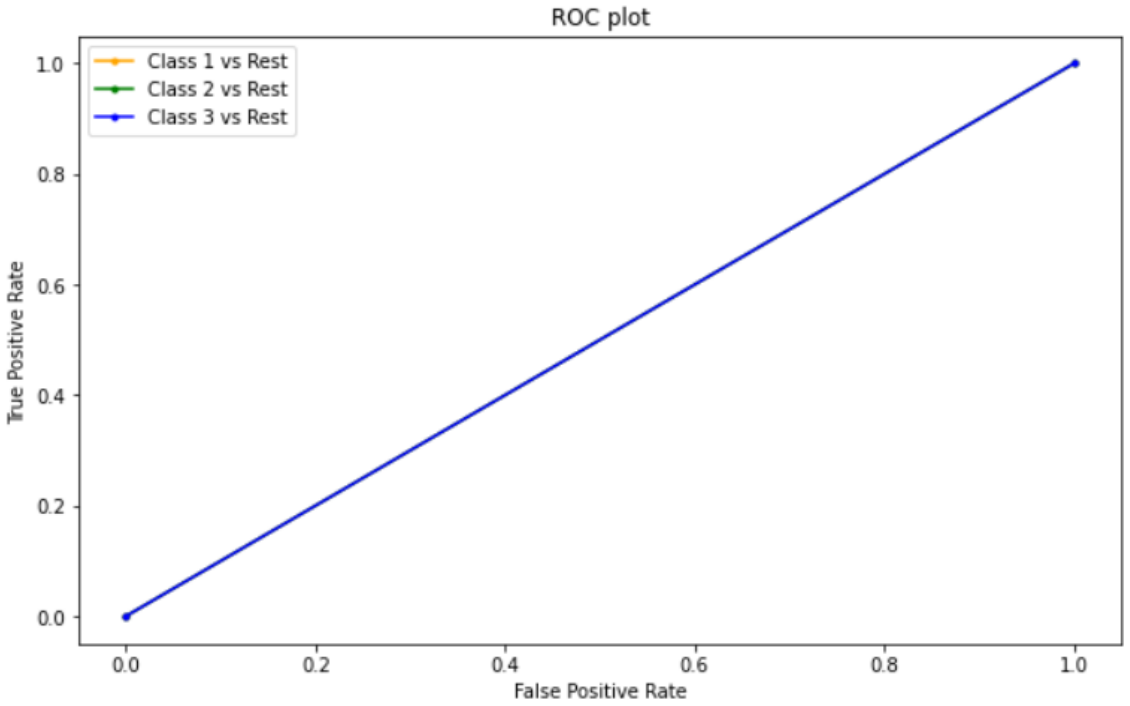
DT-B-3 Testing:

```
Testing Set:
Accuracy:  0.8263888888888888
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undef
  _warn_prf(average, modifier, msg_start, len(result))
Recall Score:  0.3333333333333333
Precision Score:  0.27546296296296297
```



DT-B-3 Training:

```
Training Set:
Accuracy:  0.828695652173913
Recall Score:  0.3333333333333333
Precision Score:  0.276231884057971
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Un
  _warn_prf(average, modifier, msg_start, len(result))
```



After the full pruning phase only, we get a single node tree. Since further pruning is not possible, the next pruning phase is avoided.

Also since it's only predicting based on the majority of that single node, the performance is compromised and the model is working the same as random guessing, hence we are getting a straight line in ROC plot.

Based on the size and performance, DT_B_2_CC works the best.

C (Experiments):

Question 1:

DT-C-1:

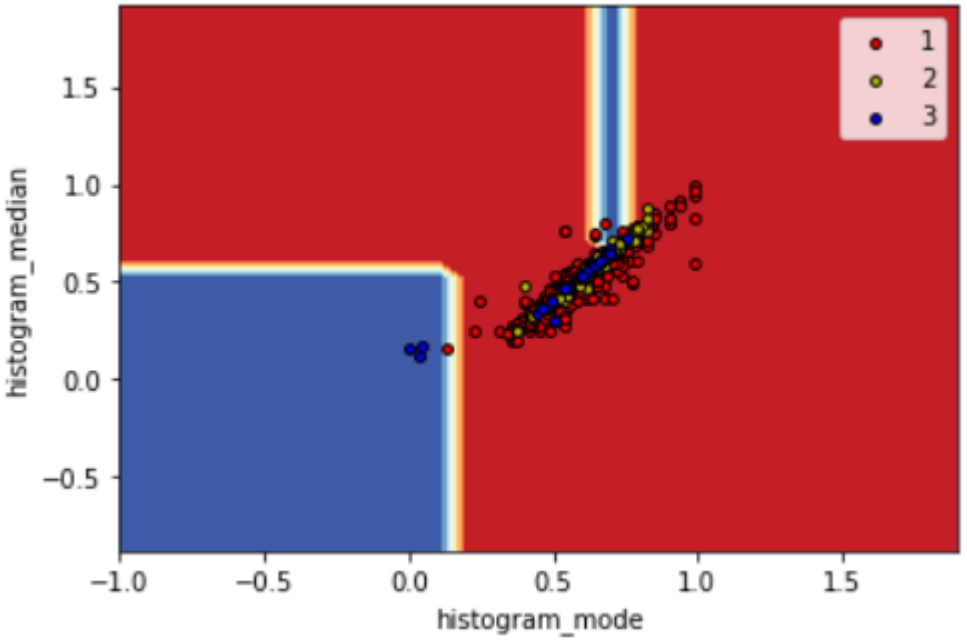
	precision	recall	f1-score	support
1	0.92	0.91	0.91	329
2	0.61	0.76	0.68	59
3	0.65	0.43	0.52	35

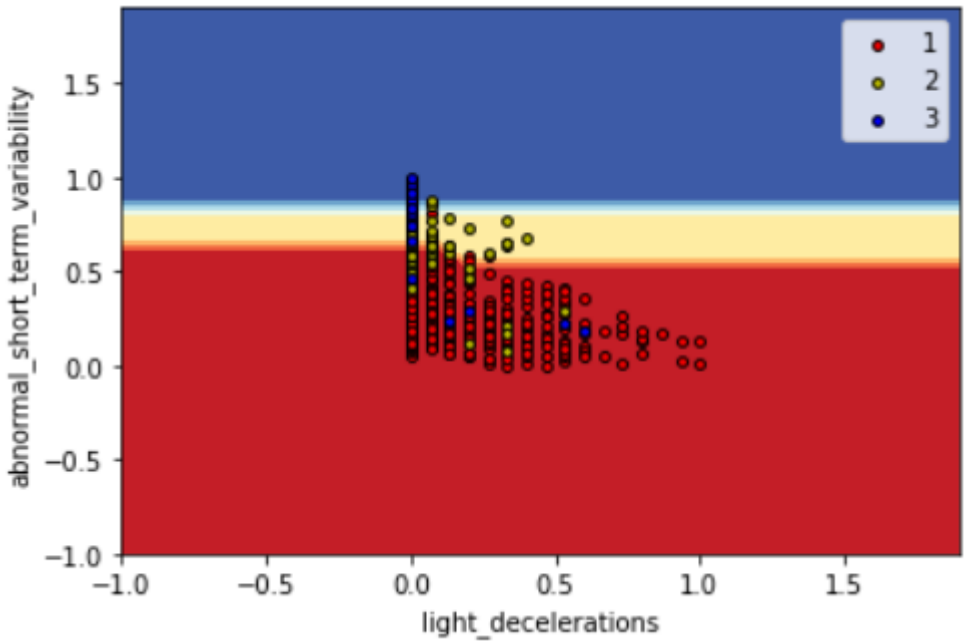
DT-C-1-X:

	precision	recall	f1-score	support
1	0.97	0.92	0.95	329
2	0.61	0.81	0.70	59
3	0.66	0.60	0.63	35

Question 2:

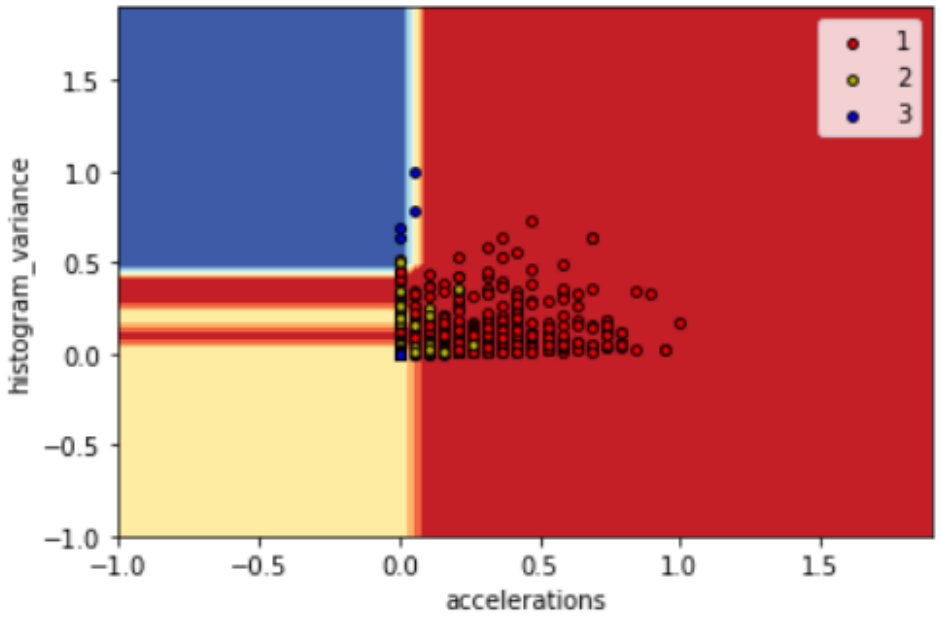
Decision Surface Boundaries





Using the Decision Tree corresponding to the above Decision Boundary, we took a random sample, and calculated the distance (euclidean) to the nearest decision boundary.

Random sample: (light_decelerations: 0.0, abnormal_short_term_variability: 0.866667)
Euclidean Distance: 0.02 (Approx.)



References:

1. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_roc_curve.html
4. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
5. https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html
6. https://sci2s.ugr.es/keel/pdf/algorithm/congreso/SLIQ.pdf
7. https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html