

# DATA MINING

## ASSIGNMENT-1

---

AASTHA 2019224  
SATWIK TIWARI 2019100

### Preprocessing

1. Missing Values: There were no missing values in the dataset.
2. Feature Scaling: Not performed as results were the same before and after standardization, as decision trees are not sensitive to variance in data.
3. Binarization: Binarized the y values(fetal health column) as multiclass classification is not supported by roc\_curve.

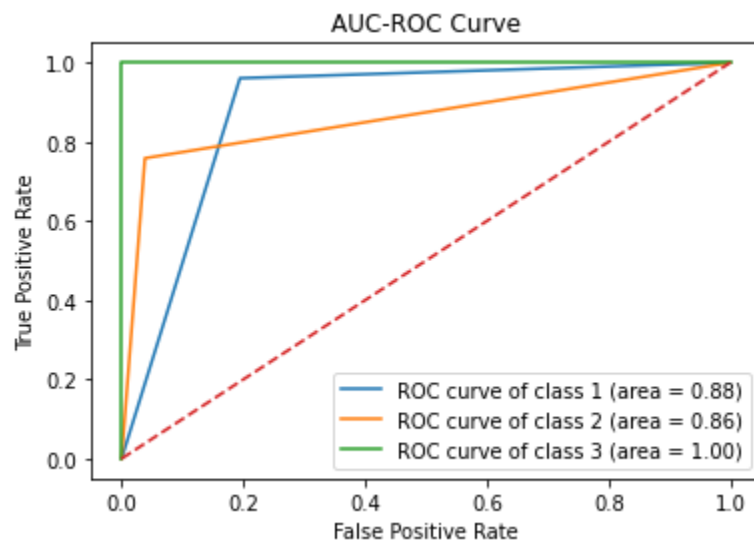
### A1

#### Results

Accuracy 0.9413793103448276

Precision 0.9413793103448276

Recall 0.9413793103448276



---

## Learnings

**Accuracy** = Number of correct predictions/ Total number of predictions

Accuracy indicates that 94% of our predictions are correct.

**Precision** =  $TP / (TP + FP)$

A higher precision(94%) indicates that False positives are less.

**Recall** =  $TP / (TP + FN)$

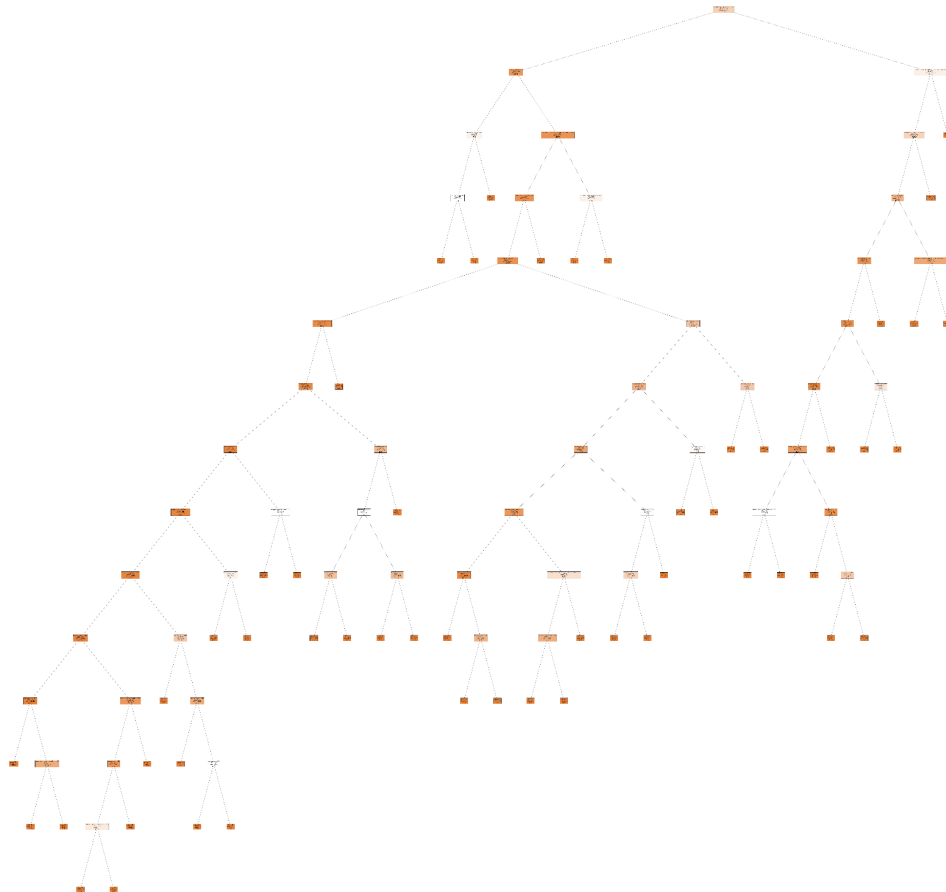
Recall is used to measure sensitivity. A recall of 94% indicates that False Negatives are much less than True Positives.

## AUC-ROC

ROC curve plots the true positive rate against the false-positive rate at different threshold values and AUC is used to measure the separability. The higher the AUC the better is the model in classifying the class correctly. From the curve, classes 1, 2, and 3 have AUC of 0.88, 0.86 and 1.0 which indicates that the model has high separability.

---

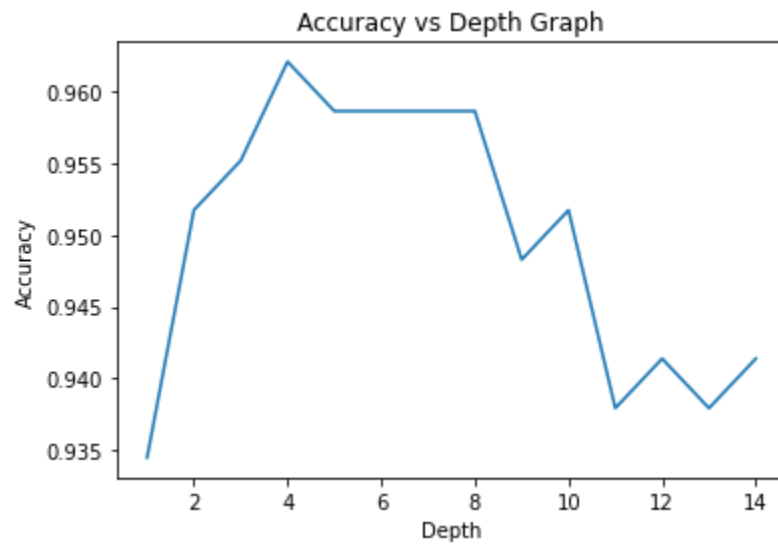
## Visualization of Decision Tree



---

## A2

The accuracy vs depth Graph is shown below. The best accuracy is obtained at depth 4.



---

## A3

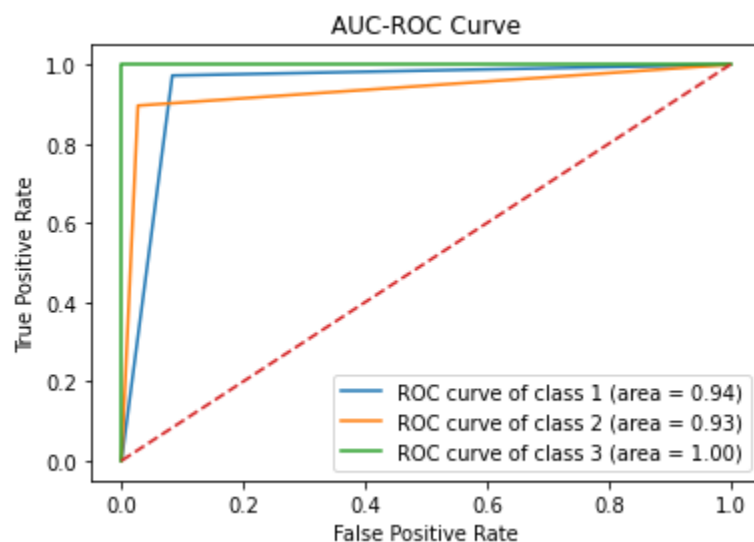
### Observations on varying hyperparameters

#### 1. `criterion='entropy'`

Accuracy 0.9655172413793104

Precision 0.9655172413793104

Recall 0.9655172413793104



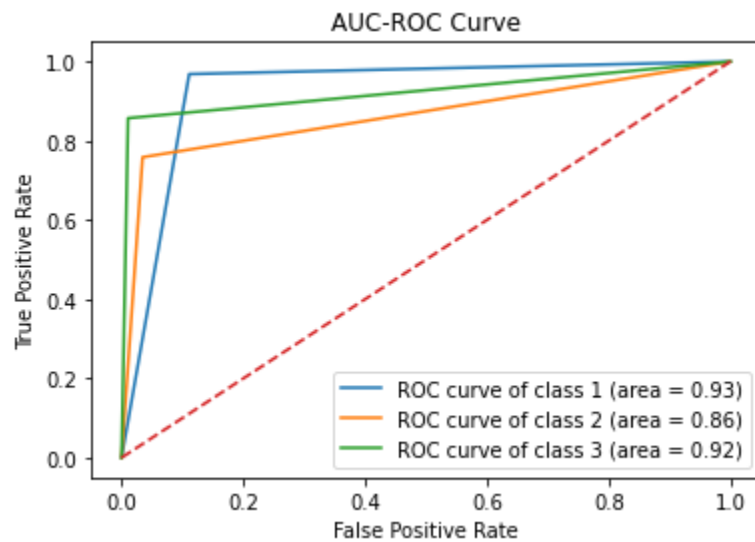
The model performs better than the base model with criterion set to entropy. In the base model, the criterion was set to gini. This indicates that calculating information gain performs better on the given dataset than gini impurity. Both criteria have their own advantages, entropy performs 2% better but requires more computations than Gini.

#### 2. `splitter="best"`

Accuracy 0.9448275862068966

Precision 0.9448275862068966

Recall 0.9448275862068966



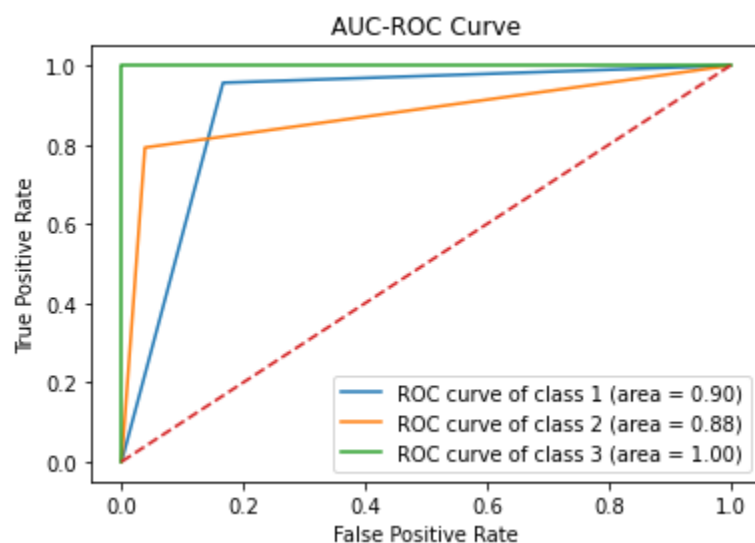
The model performs better than the base model. In the base model, the splitter is set to best while here it is set to random. The base model chooses the best split at each node whereas this model chooses a random split. The better performance is because the random splitter is less prone to overfitting hence it reduces the variance in the test set.

### 3. `min_samples_split=3`

Accuracy 0.9413793103448276

Precision 0.9446366782006921

Recall 0.9413793103448276



---

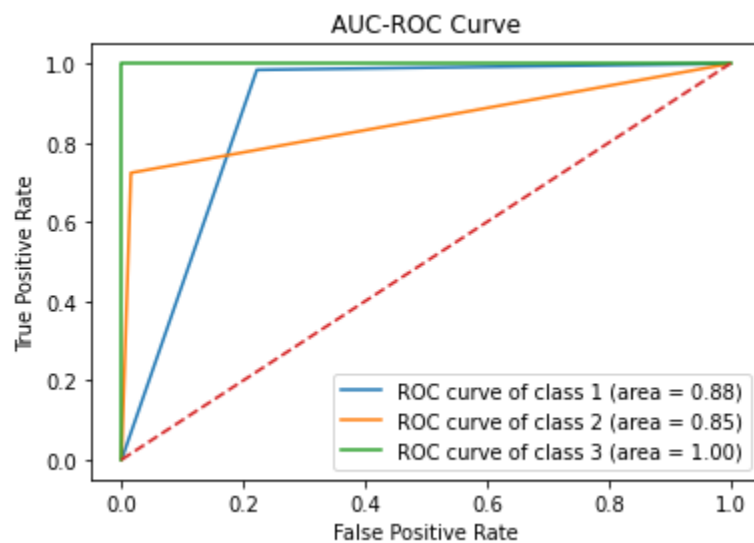
The model performs the same as the base model. The default value for the minimum number of samples required to split an internal node is 2 and the value for this model is 3. This indicates that most of the splits are occurring on greater than 2 nodes even in the base model.

#### 4. `max_depth=6`

Accuracy 0.9586206896551724

Precision 0.9586206896551724

Recall 0.9586206896551724



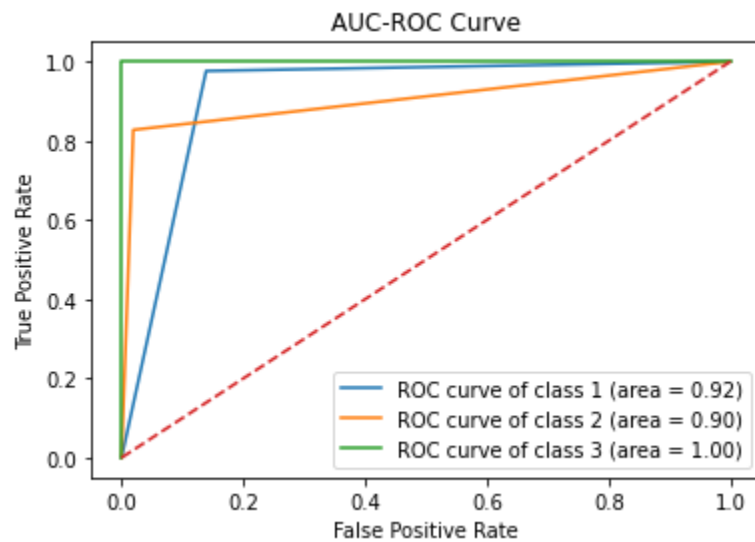
The model performs better than the base model. In the base model, the max depth is None so the nodes are expanded until all leaves are pure. Setting max\_depth to 6 helps prevent overfitting and reduces the variance as it pre prunes the tree.

#### 5. `min_samples_leaf=3`

Accuracy 0.9620689655172414

Precision 0.9653979238754326

Recall 0.9620689655172414



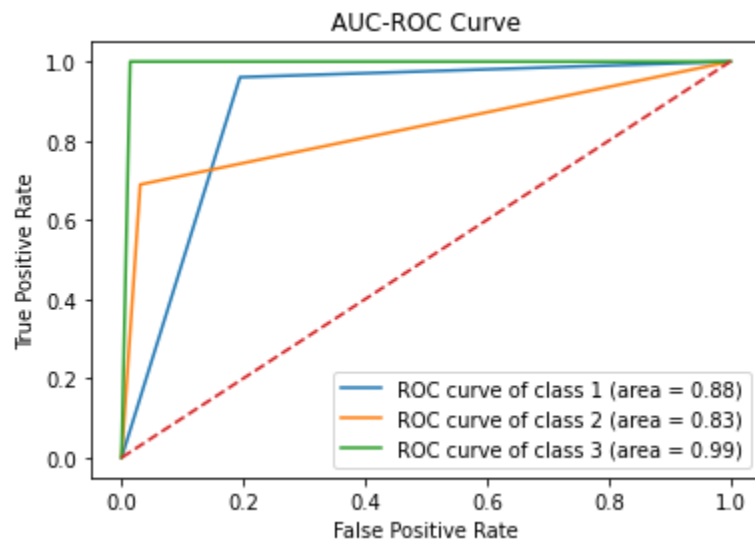
The model performs better than the base model. In the base model, the min samples leaf is 1 while here it is set to 3. This helps prevent overfitting as now a node can be leaf node with 3 samples instead of 1.

6. `max_features="sqrt"`

Accuracy 0.9344827586206896

Precision 0.9344827586206896

Recall 0.9344827586206896



The model performs worse than the base model. `max_features` is the number of features to consider when looking for the best split. The value of `max_features` is `n_features` in the



---

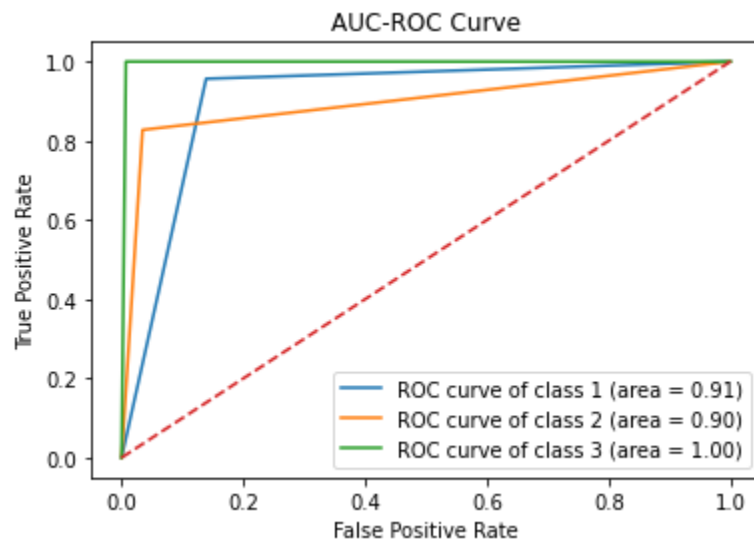
base model and is  $\sqrt{n\_features}$  in this model. Setting this hyperparameter to  $\sqrt{n}$  reduces the computational complexity and but reduces the performance as the number of features being considered for split decreases so we are not able to choose the most accurate feature.

#### 7. `class_weight="balanced"`

Accuracy 0.9448275862068966

Precision 0.9448275862068966

Recall 0.9448275862068966



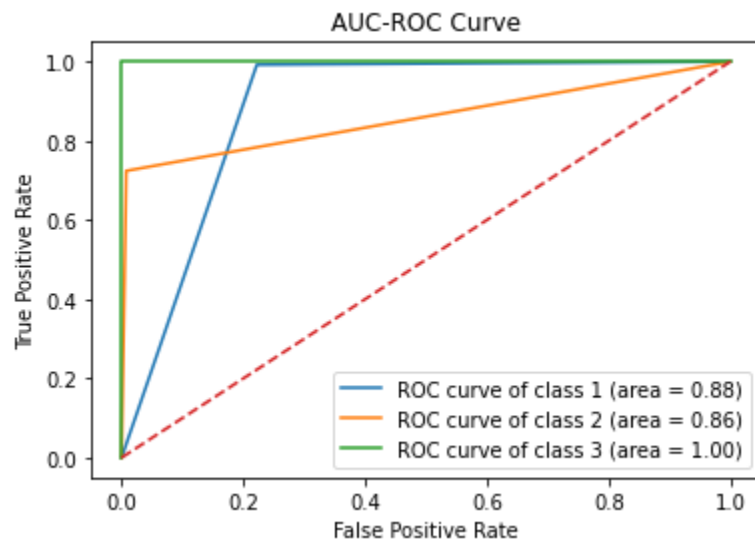
The model performs slightly better than the base model. In the base model, the value of weights assigned to each class is 1 whereas balanced mode assigns weights inversely proportional to class frequencies. This difference is because we have an imbalanced dataset with value counts as { 1 - 1200, 2-200, 3-50} and setting class weights to balanced penalizes samples of different classes relative to each other.

#### 8. `max_leaf_nodes=5`

Accuracy 0.9655172413793104

Precision 0.9655172413793104

Recall 0.9655172413793104



The model performs better than the base model. The default value of `max_leaf_nodes` is `None`. Setting `max_leaf_nodes` helps in pre pruning the tree, reduces the depth, overfitting and variance.

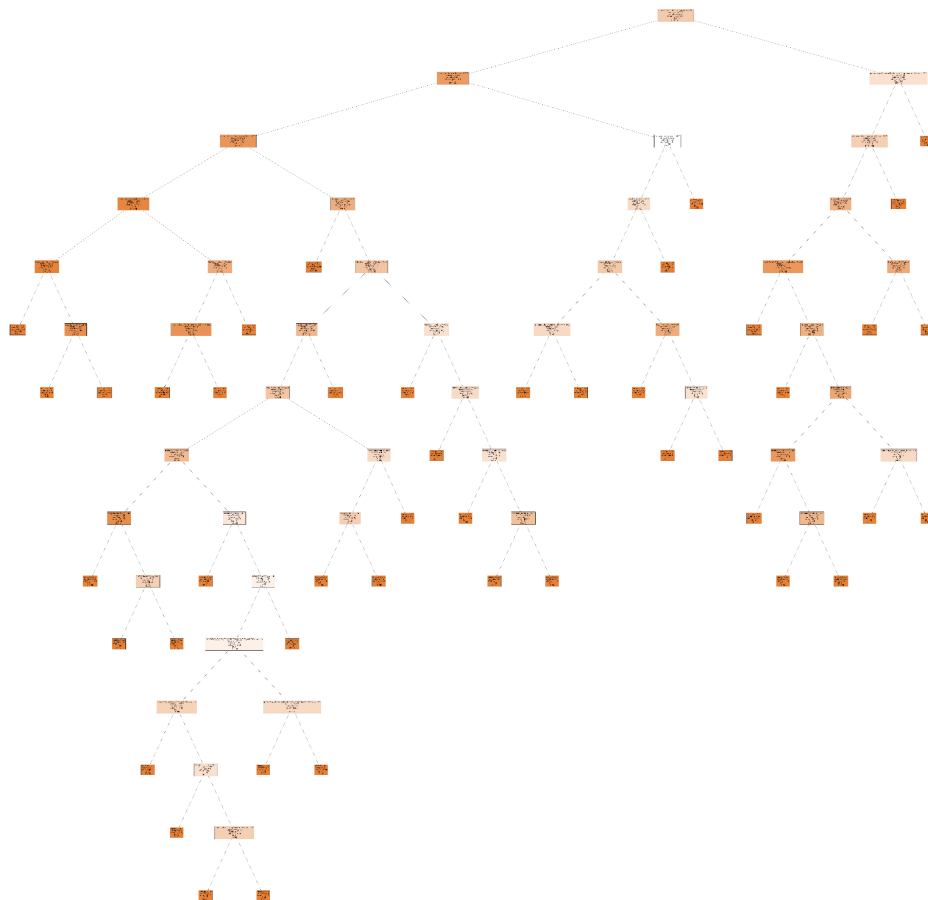
### Best Hyperparameter

The best hyperparameter obtained is `criterion='entropy'`.

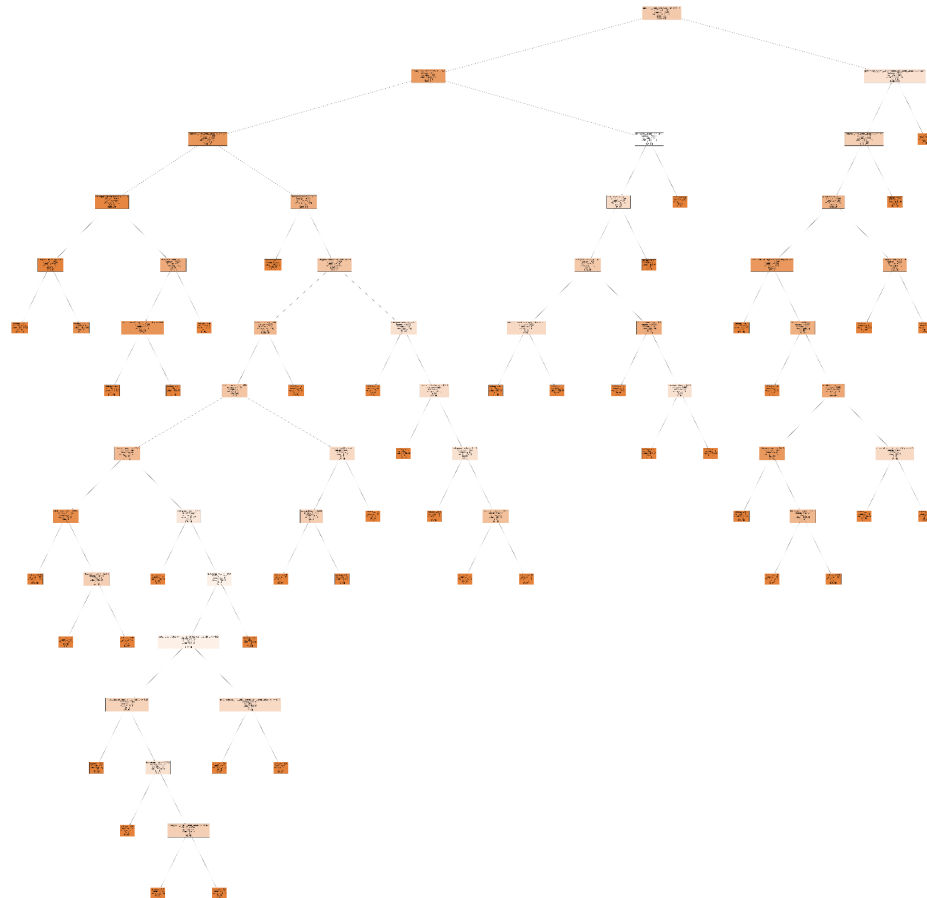
---

## B1

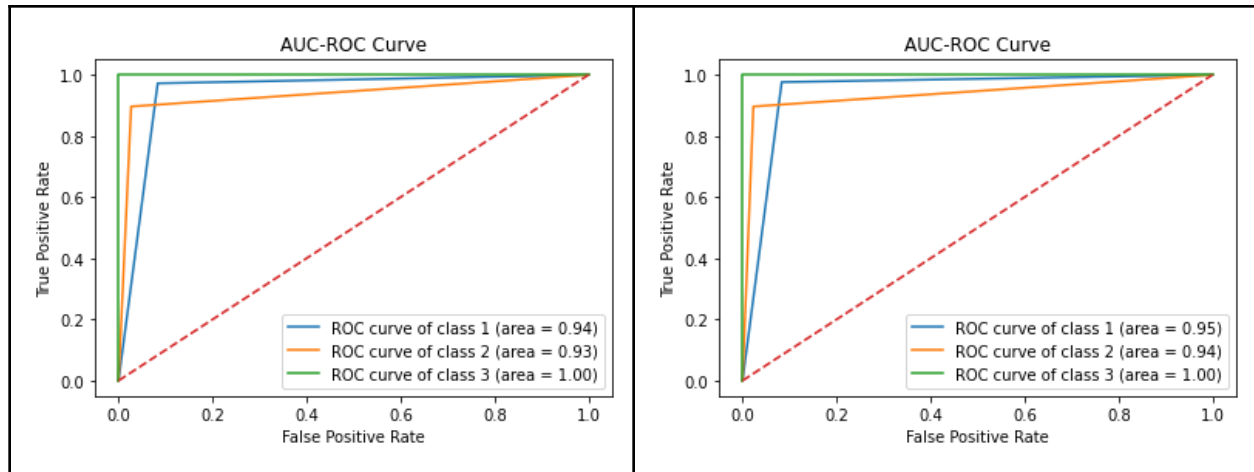
### Decision tree before removing a node



## Decision tree after removing a node



Before	After
Accuracy 0.9655172413793104 Precision 0.9655172413793104 Recall 0.9655172413793104	Accuracy 0.9689655172413794 Precision 0.9689655172413794 Recall 0.9689655172413794



## Learnings

There is an increase in the accuracy, precision and recall after removing a random node from the tree. This is because removing the node decreases overfitting and reduces the variance so the model performs better on the testing set.

---

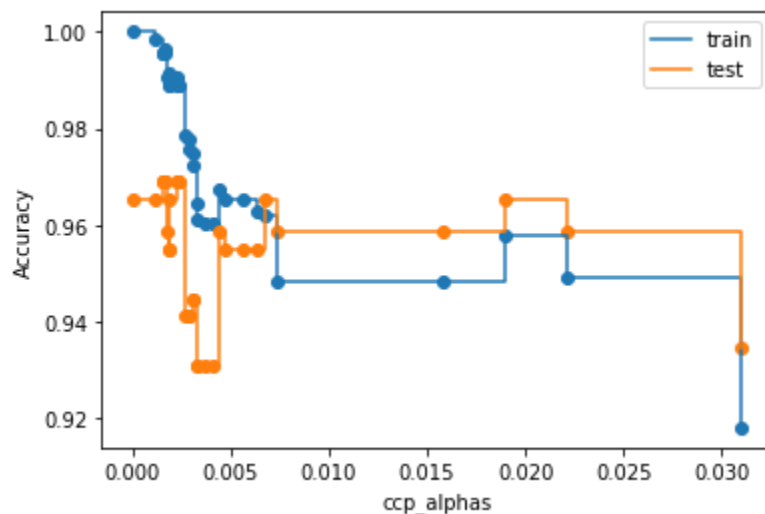
## B2

### Cost Complexity Pruning

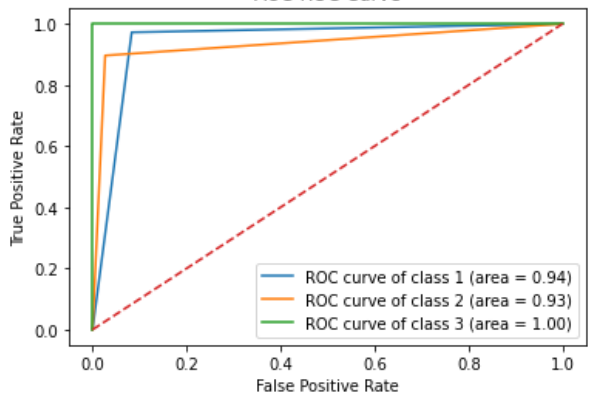
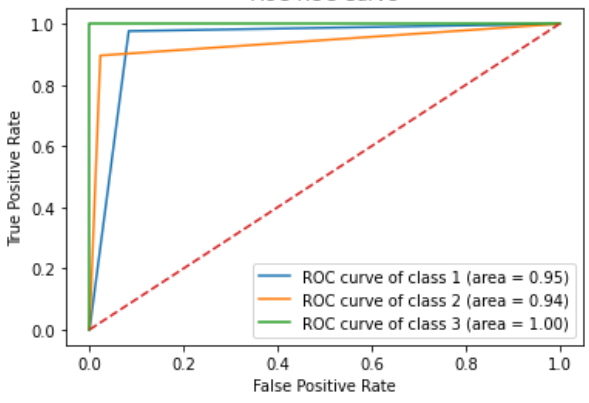
Using cost complexity pruning path the following values were obtained for ccp\_alpha and impurity(excluding the last value)

```
{'ccp_alphas': array([0.          , 0.00114943, 0.00158327, 0.00158327,
0.00158327,
      0.001629   , 0.00166953, 0.00172414, 0.00186501, 0.00186501,
      0.0018959   , 0.00224146, 0.00231234, 0.00233789, 0.00265997,
      0.00285391, 0.0028848   , 0.00308045, 0.00312012, 0.00326925,
      0.00330446, 0.00368961, 0.00407262, 0.0043762   , 0.00472348,
      0.00564125, 0.0062952   , 0.00673869, 0.00740288, 0.01583594,
      0.01894564, 0.02208928, 0.03098088]),
 'impurities': array([0.          , 0.00114943, 0.00273269, 0.00431596,
0.00589923,
      0.00752824, 0.00919776, 0.01264604, 0.01451105, 0.01637605,
      0.01827195, 0.02051341, 0.02282575, 0.02516364, 0.02782361,
      0.03067752, 0.03356232, 0.03972323, 0.04284335, 0.05265111,
      0.06586895, 0.06955856, 0.07363118, 0.07800738, 0.08273085,
      0.08837211, 0.10725772, 0.11399641, 0.12139929, 0.13723523,
      0.17512651, 0.19721579, 0.25917754])}
```

The following curve was obtained by plotting the accuracy obtained from model with the ccp\_alpha values taken from the path above. From the curve ccp\_alpha=0.0015832686794042922 was chosen for cost complexity pruning as it has the maximum test accuracy hence our model is performing better for this value on unseen data.



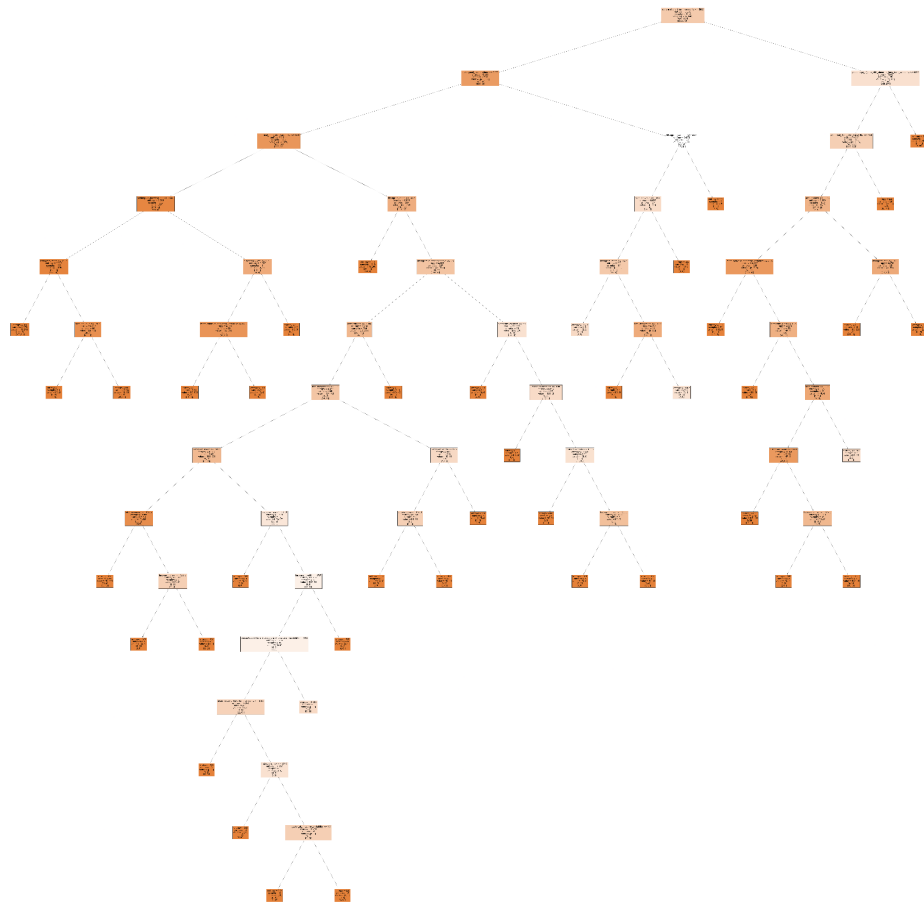
## Performance

Without Cost Complexity Pruning	With Cost Complexity Pruning
Accuracy 0.9655172413793104 Precision 0.9655172413793104 Recall 0.9655172413793104	Accuracy 0.9689655172413794 Precision 0.9689655172413794 Recall 0.9689655172413794
<p>AUC-ROC Curve</p>  <p>True Positive Rate</p> <p>False Positive Rate</p> <p>ROC curve of class 1 (area = 0.94) ROC curve of class 2 (area = 0.93) ROC curve of class 3 (area = 1.00)</p>	<p>AUC-ROC Curve</p>  <p>True Positive Rate</p> <p>False Positive Rate</p> <p>ROC curve of class 1 (area = 0.95) ROC curve of class 2 (area = 0.94) ROC curve of class 3 (area = 1.00)</p>

## Learnings

There is a slight increase in accuracy, precision and recall after cost complexity pruning as the model performs better on unseen data after pruning. Cost complexity pruning helps prevent overfitting and reduces the variance.

### Decision tree after cost complexity pruning





---

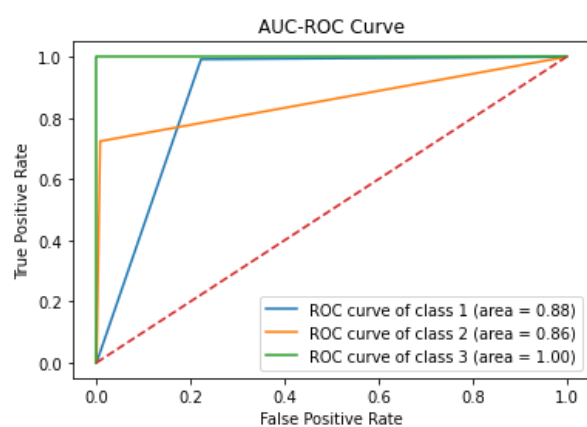
## Pre Pruning

Using Grid Search CV the following values were obtained for the best\_params\_ for pre pruning.

```
{'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

## Performance

```
Accuracy  0.9655172413793104  
Precision  0.9655172413793104  
Recall    0.9655172413793104
```

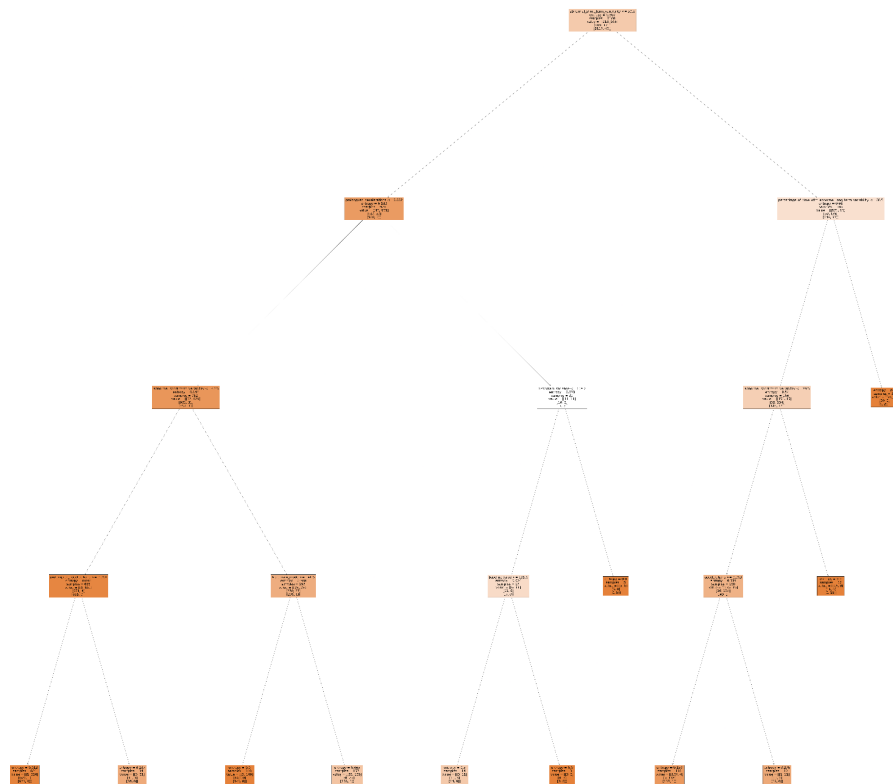


## Learnings

Pre pruning the decision tree helps in increasing the depth of the tree and prevents overfitting. We notice that we get the same accuracy, precision and recall even though the tree depth is much smaller than DT-A hence this technique is more efficient and reduces computational cost.

---

## Decision tree with pre pruning



---

## B3

### Approach -

So first I've calculated the number of nodes which are internal node and the number of nodes which are leaf nodes in the subtree for a particular node of the decision tree.

This can be done using a normal dfs tree traversal by using -

$\text{Innernodes}[\text{node}] = 1 + \text{innernodes}[\text{left\_child}[\text{node}]] + \text{innernodes}[\text{right\_child}[\text{node}]]$

Similarly for leadnodes[node] as well.

```
def dfs(tree, index, data):
    if(is_leaf(tree, index)):
        leafnodes[index] += 1
        return

    leftdata = []
    rightdata = []

    innernodes[index] += 1
    currfeature = tree.feature[index]
    for i in range(len(data)):
        if(data[i][currfeature] < tree.threshold[index]):
            leftdata.append(data[i])
        else: rightdata.append(data[i])

    dfs(tree, tree.children_left[index], leftdata)
    dfs(tree, tree.children_right[index], rightdata)

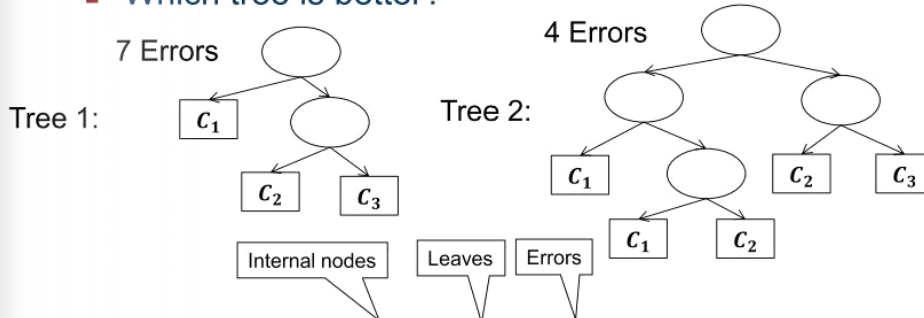
    innernodes[index] += innernodes[tree.children_left[index]] + innernodes[tree.children_right[index]]
    leafnodes[index] += leafnodes[tree.children_left[index]] + leafnodes[tree.children_right[index]]
```

Then to find the cost of the subtree we have used this formula -

$\text{Cost}[\text{node}] = \text{innernodes}[\text{node}] * \text{cost\_of\_encoding\_of\_innernode} + \text{leafnodes}[\text{node}] * \text{cost\_of\_encoding\_of\_leadnode} + \text{error\_t} * (\text{size\_of\_samples}) + \text{enoding\_of\_v}.$

## Pruning – MDL Example II

- Which tree is better?



- Cost for tree 1:  $2 * 4 + 3 * 2 + 7 * \log_2 n = 14 + 7 \log_2 n$
- Cost for tree 2:  $4 * 4 + 5 * 2 + 4 * \log_2 n = 26 + 4 \log_2 n$
- If  $n < 16$ , then tree 1 is better.
- If  $n > 16$ , then tree 2 is better.

57

So by again using the recursive function `calc_cost` we have calculated the cost for every node.

To decide whether to prune the node or not, we have used this condition that if the  $\text{cost}[\text{node}] < \text{cost}[\text{left\_child}[\text{node}]] + \text{cost}[\text{right\_child}[\text{node}]]$  then we'll prune this node and make it as a leaf node else we'll recurse on left and the right subtree for further pruning.

```
def pruning(tree, index):
    if(is_leaf(tree, index)): return

    # cost[node] < cost[left] + cost[right]
    if(cost[index] < cost[tree.children_left[index]] + cost[tree.children_right[index]]):
        # time to prune
        tree.children_left[index] = TREE_LEAF
        tree.children_right[index] = TREE_LEAF
        return
    else:
        pruning(tree, tree.children_left[index])
        pruning(tree, tree.children_right[index])
```

---

### Comparing Tree Size, Precision, Recall and Accuracy

DT-A	DT-B-2-CC	DT-B-2-XX	DT-B-3
Accuracy 0.96551724137931 04 Precision 0.96551724137931 04 Recall 0.96551724137931 04 Depth 14	Accuracy 0.96896551724137 94 Precision 0.96896551724137 94 Recall 0.96896551724137 94 Depth 14	Accuracy 0.96551724137931 04 Precision 0.96551724137931 04 Recall 0.96551724137931 04 Depth 4	

### Learnings -

Got to learn about SQIL pruning technique and explored a lot while going through the SQIL research paper.

# C1

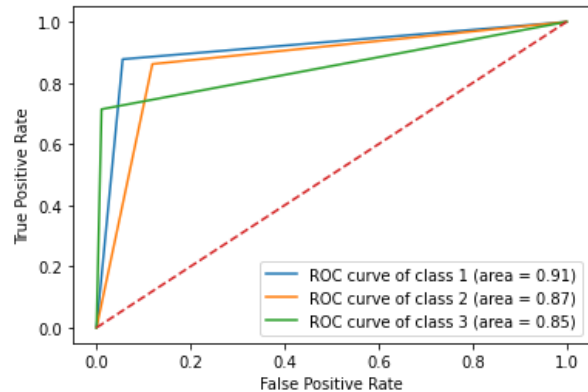
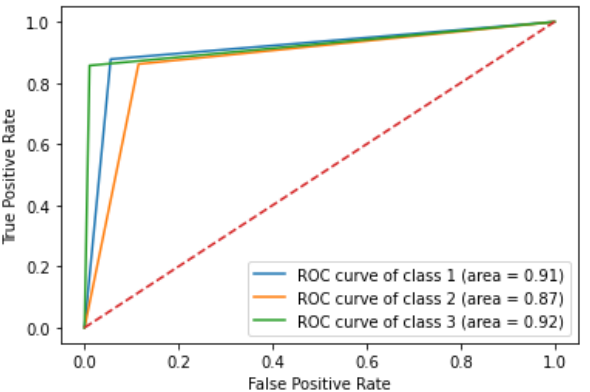
## Decision Tree Augmentation or Incremental Decision Tree

### Approach 1 -

We have used the Hoeffding Tree Classifier and trained it on the Data1 training set and then used the partial\_fit method to augment the decision tree with the training data set of Data2.

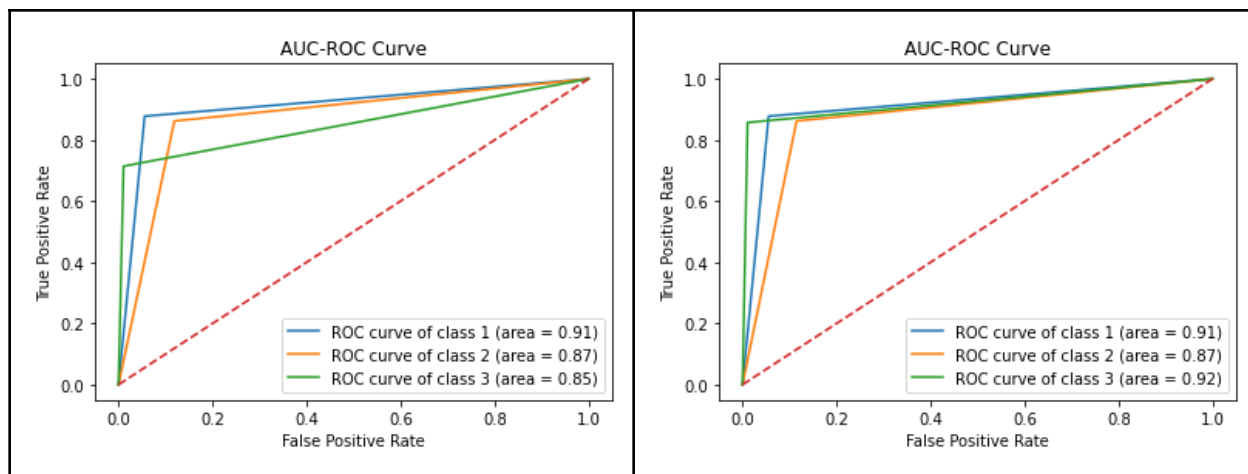
### Results -

#### For Data 1

Before	After
Accuracy 0.8724137931034482 Precision 0.8724137931034482 Recall 0.8724137931034482	Accuracy 0.8758620689655172 Precision 0.8758620689655172 Recall 0.8758620689655172
<p>AUC-ROC Curve</p> 	<p>AUC-ROC Curve</p> 

#### For Data 2

Before	After
Accuracy 0.8724137931034482 Precision 0.8724137931034482 Recall 0.8724137931034482	Accuracy 0.8758620689655172 Precision 0.8758620689655172 Recall 0.8758620689655172



## Learning -

We have explored different API's and then came across this scikit - multflow library which contains an incremental decision tree. Augmenting the decision tree with new available data sets does increase the accuracy of the model.

## Approach 2 -

Out of curiosity, We tried some different approaches using some heuristics to learn and explore more about the augmentation in the decision tree.

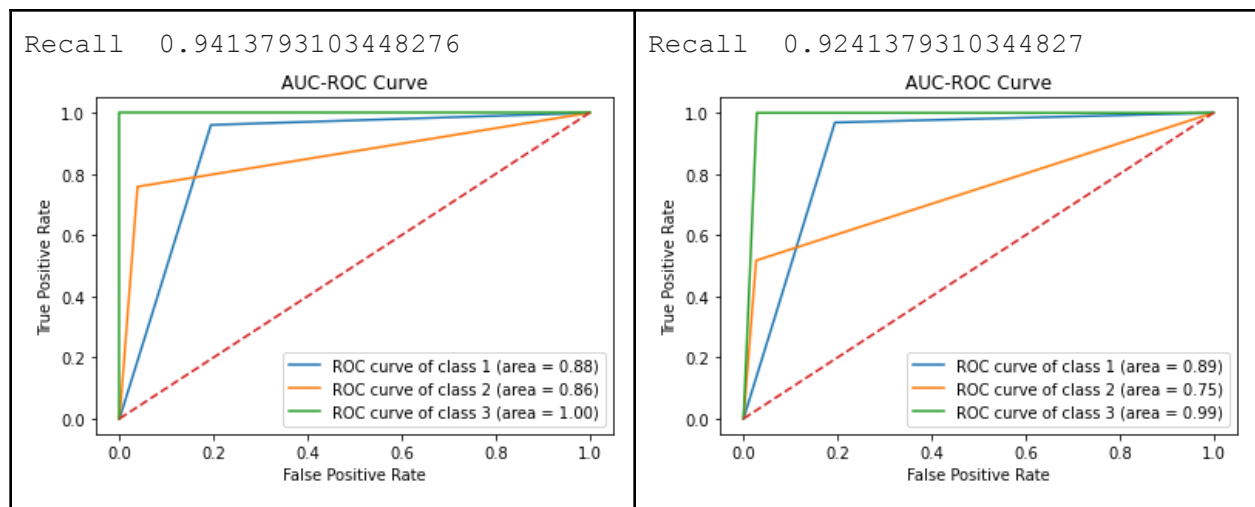
We are traversing the tree and making some heuristics to recalculate the impurity and the threshold of every node. This does reduce the accuracy from the previously trained decision tree.

During traversal, we are also splitting the dataset according to the threshold and then passing the left side and right side dataset in both the left and right subtree recursive call respectively.

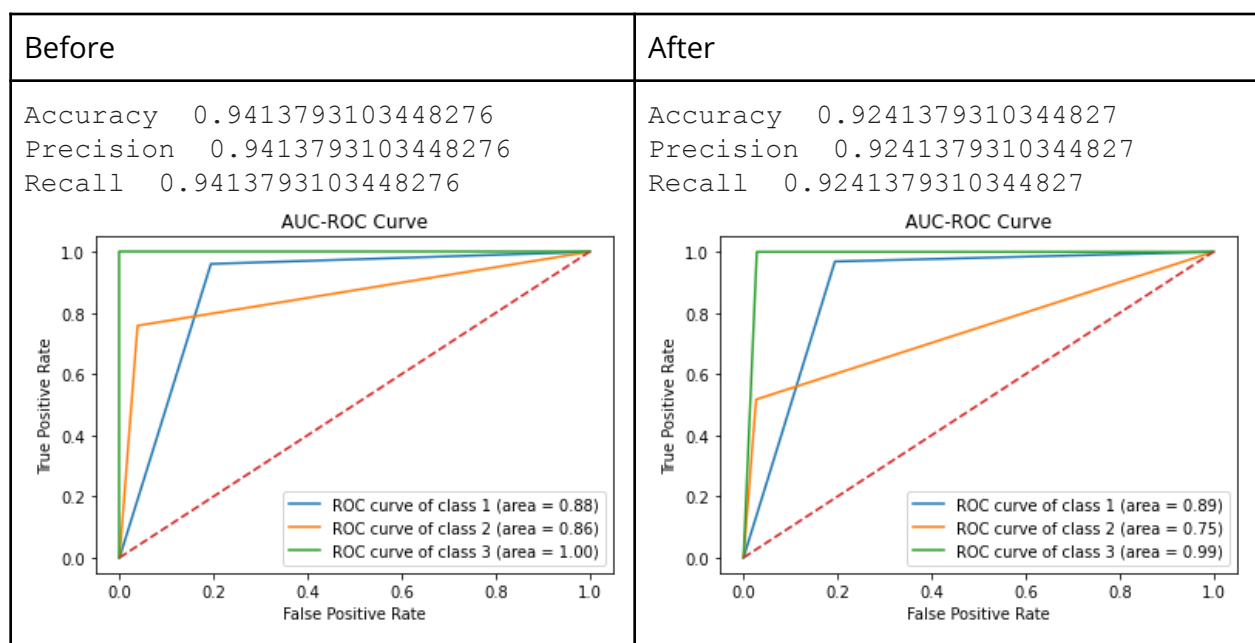
## Results-

### For Data 1

Before	After
Accuracy 0.9413793103448276	Accuracy 0.9241379310344827
Precision 0.9413793103448276	Precision 0.9241379310344827



## For Data 2



## Learnings -

The heuristics made was not up to the mark and the accuracy reduces from the previously trained decision tree. Though the Accuracy was still above 90%.



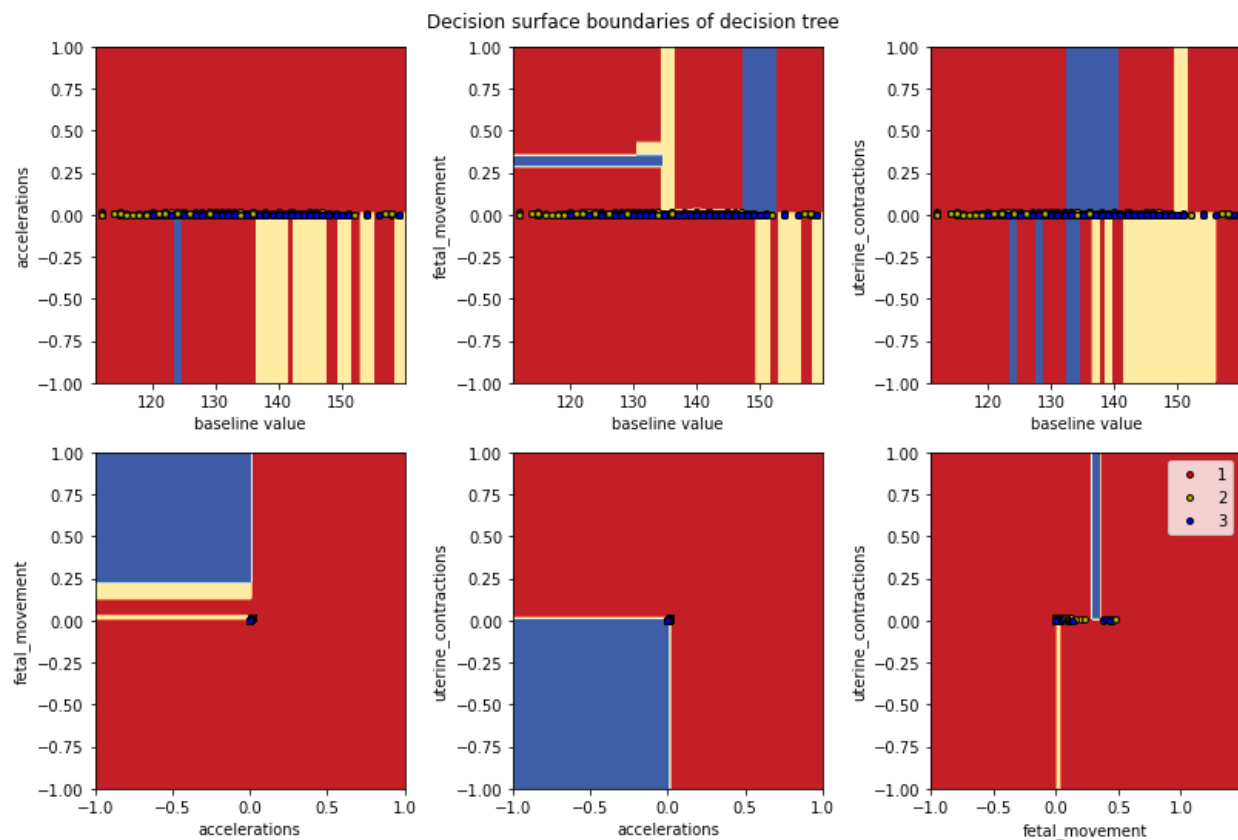
---

## C2

Assumption-The best performing tree from part B was with cost complexity pruning. Hence that has been taken for C2.

### Decision Surface Boundaries

Decision surface boundaries have been visualized by taking pairs from 4 features-'baseline value', 'accelerations', 'fetal\_movement', 'uterine\_contractions'. The code can be modified to plot for more features by modifying the n\_features variable and updating the figure size.



### Learnings

In the figure above, red, yellow and blue represent classes 1, 2 and 3 respectively. The decision surface separates the space into different regions and the points lying at one side of the decision surface are classified into that class. As the classes in our dataset are

imbalanced the red portion is much greater than yellow and blue. The decision boundaries are also different for different pairs of features which shows the dependence of classification on input features.

## Distance of a sample to the nearest decision boundary

### Algorithm -

So basically we can imagine our samples as k dimensions (k is the number of features it has) and every sample will lie in a class (here we had 3 classes).

So the main idea is to keep the sample point near the threshold everytime which will result in being the sample near the boundary of the classes.

Now at leaf node, checking whether this sample (updated one) is in a different class or not. If it is in a different class then this point can be the shortest distance between the original sample and the boundary to a different class.

So, if that is the case I return the euclidean distance from original sample to this sample else we can return a dummy very large number. Taking min of all the recursive calls will get us the minimum distance of the sample from the boundary.

#### ▾ Distance of Sample to Nearest Decision Boundary

```
from copy import deepcopy
def is_leaf(tree, index):
    return tree.children_right[index]==TREE_LEAF and tree.children_left[index]==TREE_LEAF

inf = 10**9

def min_distance_boundary(tree, index, sample, original_sample, predicted_ans):
    if(is_leaf(tree, index)):
        curr_ans = decision_tree.predict([sample])[0]
        if(curr_ans != predicted_ans):
            return np.linalg.norm(sample - original_sample, ord = 2) #the sample is updated one and not the original passed as parameter
        else:
            return inf

    dist = inf
    curr_feature = tree.feature[index]
    increase = 0.01
    new_sample = deepcopy(sample)

    #left subtree
    if(sample[curr_feature] <= tree.threshold[index]):
        dist = min(dist, min_distance_boundary(tree, tree.children_left[index], sample, original_sample, predicted_ans))
        new_sample[curr_feature] = tree.threshold[index] + increase
        dist = min(dist, min_distance_boundary(tree, tree.children_right[index], new_sample, original_sample, predicted_ans))

    #right subtree
    else:
        dist = min(dist, min_distance_boundary(tree, tree.children_right[index], sample, original_sample, predicted_ans))
        new_sample[curr_feature] = tree.threshold[index]
        dist = min(dist, min_distance_boundary(tree, tree.children_left[index], new_sample, original_sample, predicted_ans))

    return dist #minimum dist of boundary from update sample.
```

---

## Results -

We have calculated the Euclidean distance of the samples in X to their nearest decision boundary.

The following distances were obtained:

```
2.6600204375035106
0.010500000023748726
0.010500000023748726
0.010500000023748726
11.51111680761027
10.727339179261243
5.1599997138977045
.
.
.
0.5
4.5
4.5
2.5
```

## Learning -

Imagined each sample as a point in some k dimensional space and then changed each dimension to near the threshold value to keep it near the boundary of each class.

This way we can change the current sample to come near the boundary in some k to find the distance from the original sample. Distance used is the euclidean distance.

Helped us in understanding more about the feature and classes relationship in the k dimensional space.

---

## References

1. <https://scikit-learn.org/stable/>
2. [https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeClassifier.html#skmultiflow.trees.HoeffdingTreeClassifier.partial\\_fit](https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeClassifier.html#skmultiflow.trees.HoeffdingTreeClassifier.partial_fit)