Assignment-3

3.1 Binary Classification

3.1(a) Decision tree from scratch

I have trained the model using both gini index and Information_gain. I have created a new class named decision tree and made all the required functions for both of them.

Gini index :
The Gini index measures the impurity or heterogeneity of a set of data.
The gini index of each class is calculated as 1 – sigma(p_i^2) where p_i is the probability of samples in S belonging to class i.
When building a decision tree, the algorithm looks for the feature and split point that result in the lowest Gini index. To find the best split using the Gini index, the algorithm computes the Gini index for each feature and each possible split point for that feature. Then, the feature and split point with the lowest Gini index are chosen as the best split.

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_a.py
Train Accuracy: 0.9885
Validation Accuracy: 0.9225
Time taken to train the model: 1758.106659412384 seconds
Train Precision: 0.9877300613496932
Validation Precision: 0.8791208791208791
Train Recall: 0.966
Validation Recall: 0.8
```

The time taken for training is 1758.109 seconds.

Training accuracy is 0.9885. Training precision is 0.9877.

Validation accuarcy is 0.9225. Validation precision is 0.87912.

Train recall is 0.966. Validation recall is 0.8.

The important functions implemented here are best split, info gain and gini.

Information gain:

Entropy is a measure of the impurity or randomness of a set of data.
Entropy(S) = -∑(p_i * log2(p_i))
where S is the set of data samples and p_i is the proportion of samples in S that belong to class i.

Information gain is the measure of the reduction in entropy achieved by splitting the data based on a particular attribute.

Information Gain(S, A) = Entropy(S) - ∑(|S_v| / |S|) * Entropy(S_v)

where A is the attribute used for splitting the data , S_v is the subset of data samples in S that have a specific value for attribute A, |S_v| is the number of data samples in S_v, |S| is the total number of data samples in S.

Information gain measures how well a particular attribute separates the data into classes, by subtracting the entropy of the split subsets from the original entropy of the data. The attribute with

the highest information gain is chosen as the splitting attribute at each decision node in the decision tree.

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_a_info_gain.py
Train Accuracy: 0.9885
Validation Accuracy: 0.9225
Time taken to train the model: 2790.610960006714 seconds
Train precision: 0.9885
Validation Precision: 0.9225
Train Recall: 0.9885
Validation Recall: 0.9225
```

The time taken for training is 2790.61 seconds.

Training accuracy is 0.9885. Training precision is 0.9885.

Validation accuarcy is 0.9225. Validation precision is 0.9225.

Train recall is 0.966. Validation recall is 0.9225.

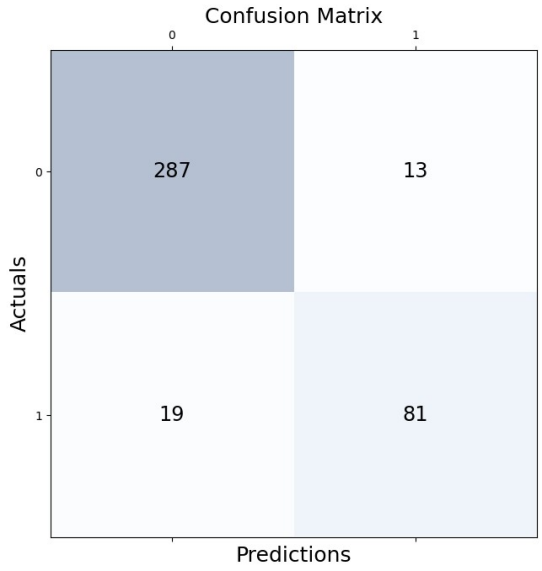
The main important functions implemented here are Entropy and the information gain and the best split function.

3.1(b) Decision tree sklearn

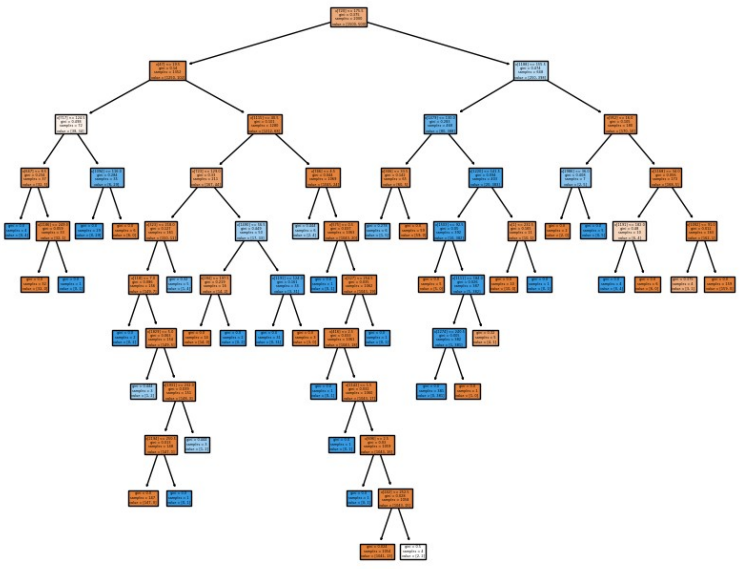
With hyperparameters max depth set to 10 and min samples split set to 7, the decision tree classifier is implemented.

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_b.py
2.363835096359253
Training Time: 2.363835096359253
Training Accuracy: 0.9885
Training Precision: 0.9877300613496932
Training Recall: 0.966
Validation Accuracy: 0.92
Validation Precision: 0.8617021276595744
Validation Recall: 0.81
```

The time taken for training is 2.36seconds.
The accuracies are as in the above diagram.
Comparison with part(a): *The validation and training accuracies of this are higher than the implementation of what I had done from scratch. The time taken to implement in part a is very high as compared to here. The precisions and recalls are also high as compared to the previous parts.*
Confusion matrix :

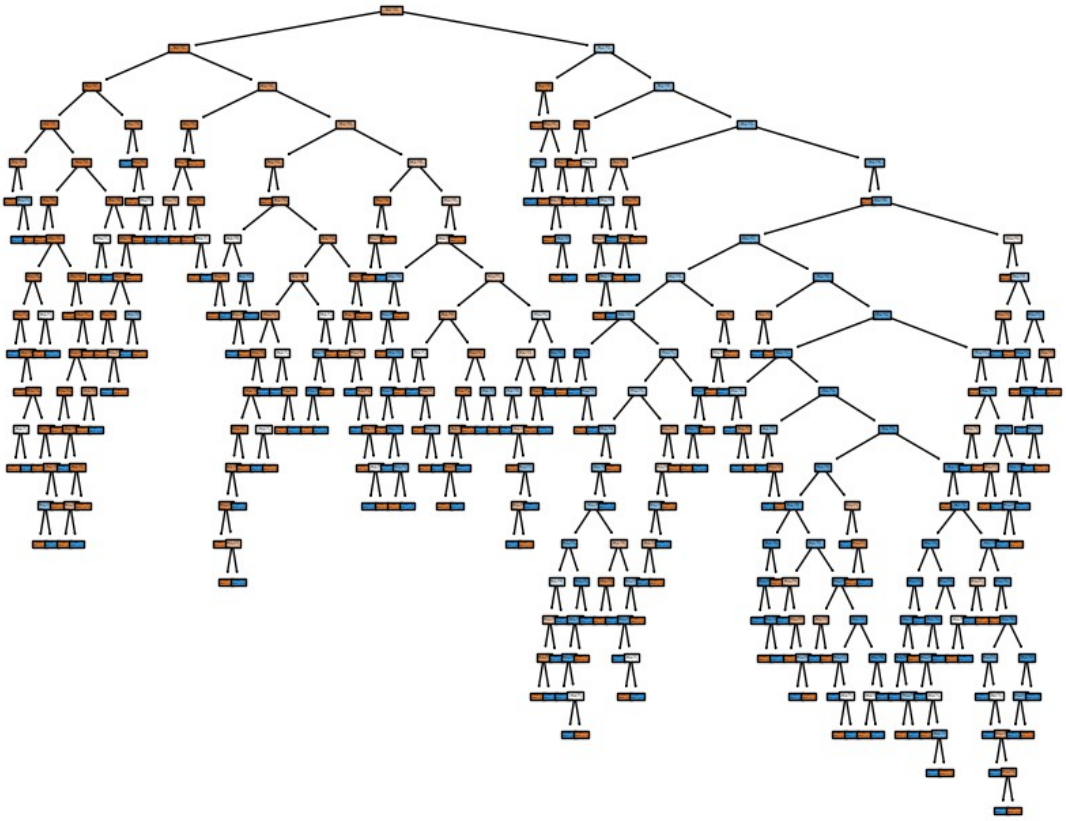


Decision tree:



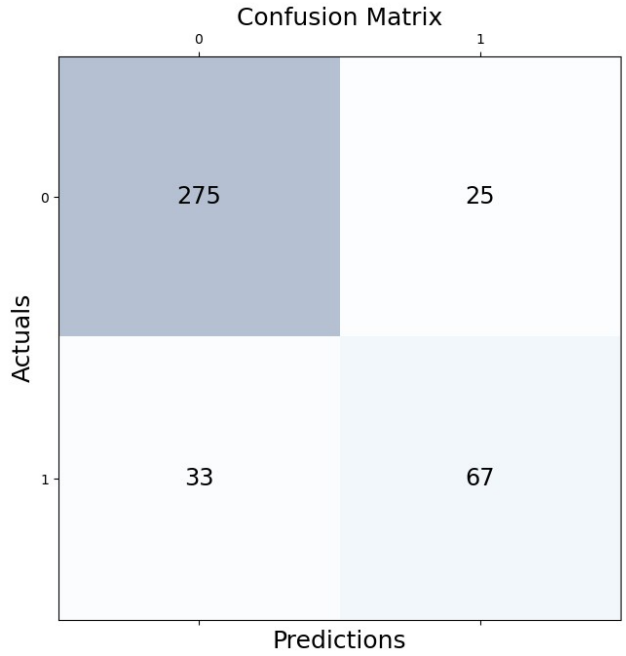
3.1(c) Decision tree Grid Search and Visualisation

We select top 10 features from the data set. Using these features, we build a decision tree. This can be done by useing the **SelectKBest** class from scikit-learn's **feature selection.**
After that, we use grid search on the given parameters and find the best parameters.
The top 10 selected feature indices are [627 714 717 720 723 726 810 813 816 819].



This is the decision tree using the top 10 features.

Confusion matrix after only using the top 10 features:



GridSearchCV using the given set of parameters:

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_c.py
[627 714 717 720 723 726 810 813 816 819]
Best hyperparameters for train data:  {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 7}
Training accuracy with best hyperparameters:  0.8584999999999999
Validation accuracy with best hyperparameters:  0.8474999999999999
```

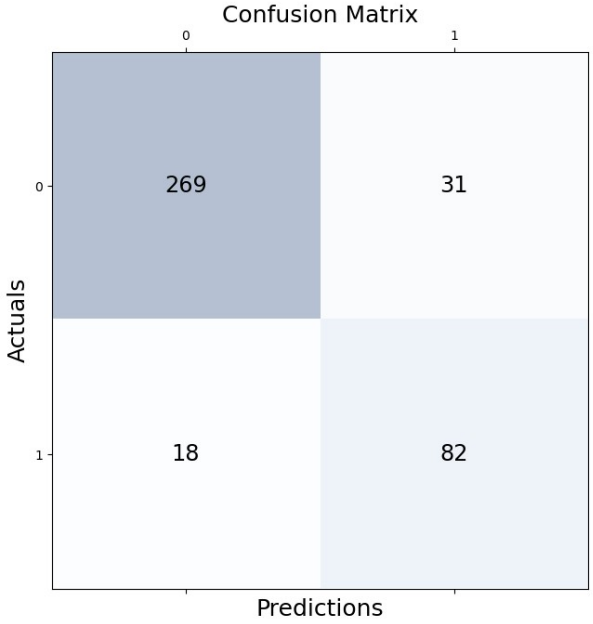
Performing a grid search on these top-10 features (using parameter cv=5) over the space mapped by following parameters: criterion: {'gini', 'entropy'}, max_depth: {None, 5, 7, 10, 15}, min_samples_split: {2, 4, 7, 9}, the values obtained are as follows.

The best parameters for the train data is with criteria entropy, max_depth = 5 and min_samples_split = 7.

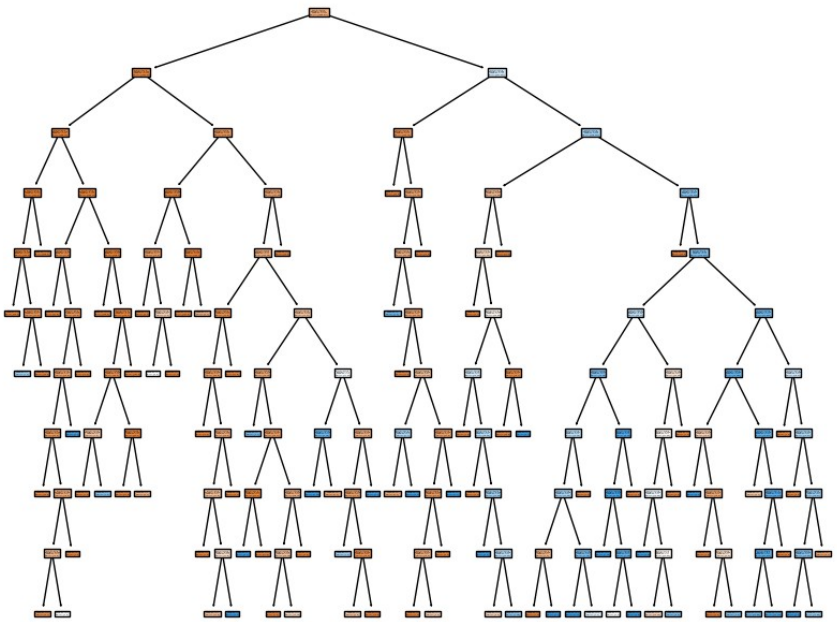
The training accuracy is 0.8584 and validation accuracy is 0.8474.

The time taken to run is 1.22 seconds.

The confusion matrix after training with the best set of parameters:



The decision tree with the best set of hyperparameters after using best 10 features:

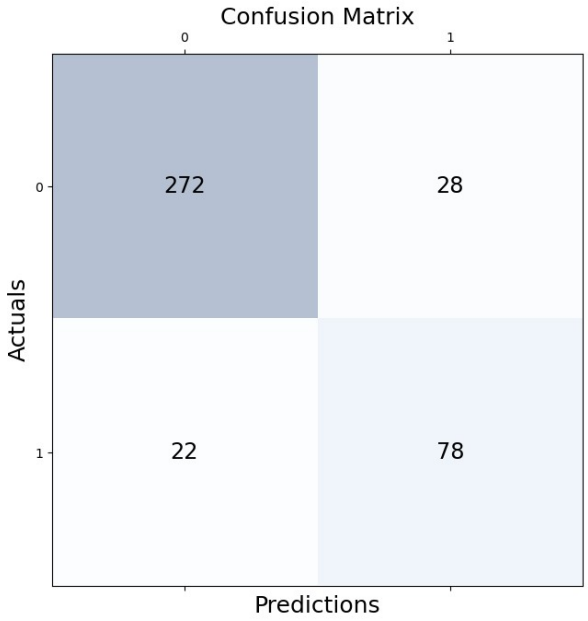


For the new set of hyperparameters testing,
Performing a grid search on these top-10 features (using parameter cv=5) over the space mapped by following parameters: criterion: {'gini', 'entropy'}, max_depth: {None, 3, 5, 7, 8, 10, 11, 15}, min_samples_split: {2, 3, 4, 7, 9, 10}, the values obtained are as follows.

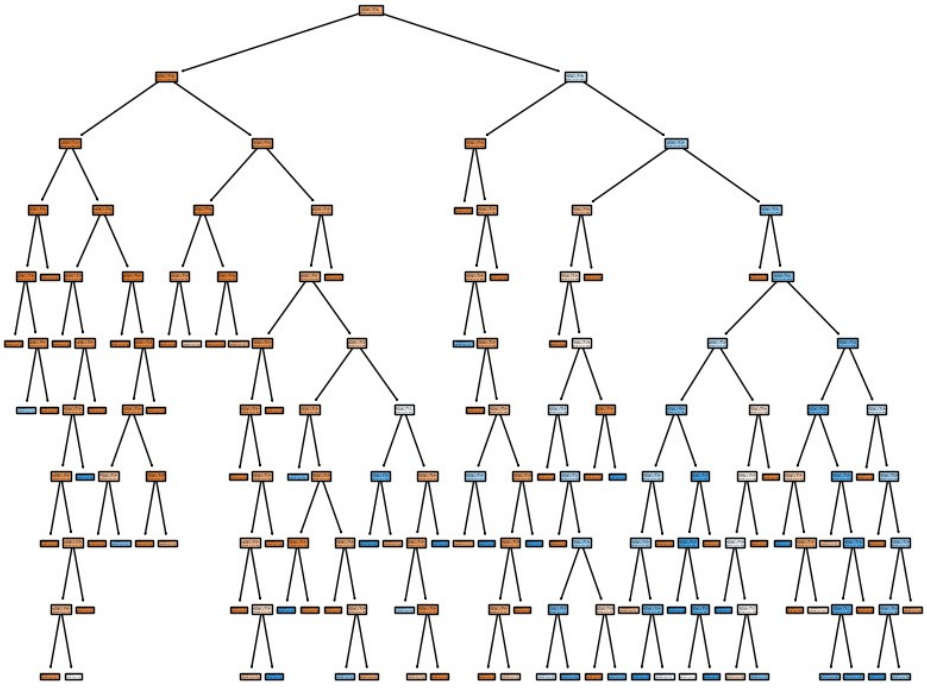
```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_c.py
[627 714 717 720 723 726 810 813 816 819]
Best hyperparameters for train data:  {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 9}
Training accuracy with best hyperparameters:  0.8584999999999999
Validation accuracy with best hyperparameters:  0.8700000000000001
Time taken to run grid search:  2.905208110809326
```

The time taken to run is 2.90 sec.
The best parameters for the train data is with criteria entropy, max_depth = 5 and min_samples_split = 9.
The training accuracy is 0.8584 and validation accuracy is 0.8700.

Confusion matrix :



Decision tree with the best new hyperparameters after taking top 10 features:

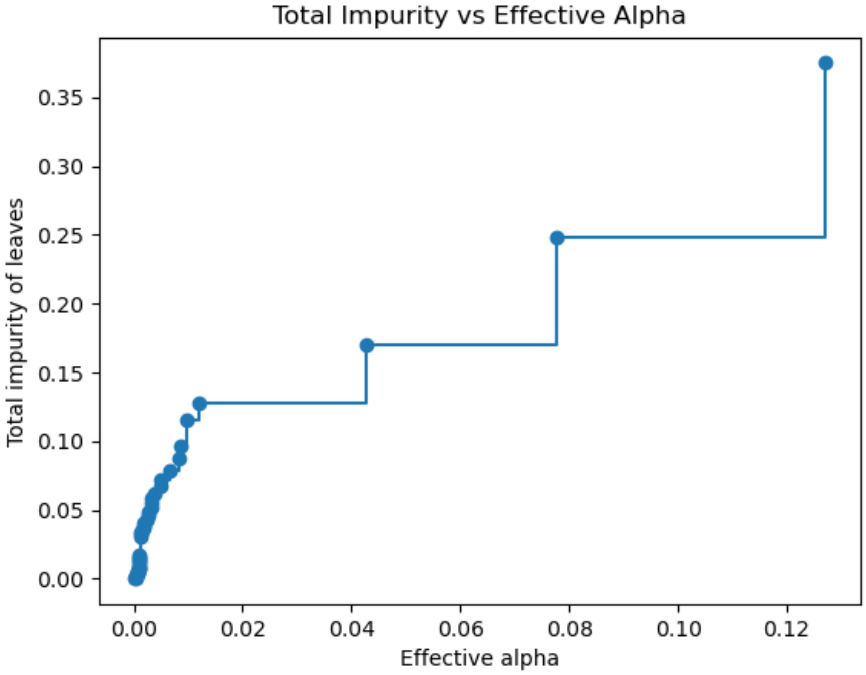


Comparison of trees with part a and part b: *The decision tree in this part is having more depth and is broader as compared to the decision trees in the previous parts. Its having more number of nodes too.*

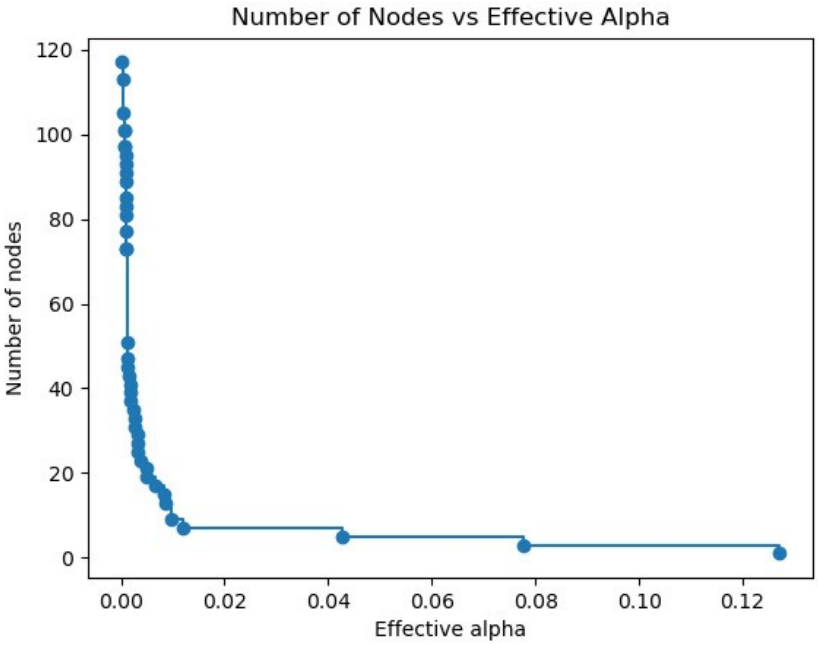
3.1(d) Decision Tree Post Pruning with Cost Complexity Pruning

We need to prune the tree inorder to reduce the complexity of the tree. We can reduce the size of the tree by reducing the number of links. For that, we first need to calculate the effective alphas and corresponding total leaf impurities at each step of the pruning process. We can do this using the **DecisionTreeClassifier.cost_complexity_pruning_path()**. *By checking the accuracy with the each ccp_alpha value obtained and creating a decision tree, we can find the optimal value of ccp_alpha.*

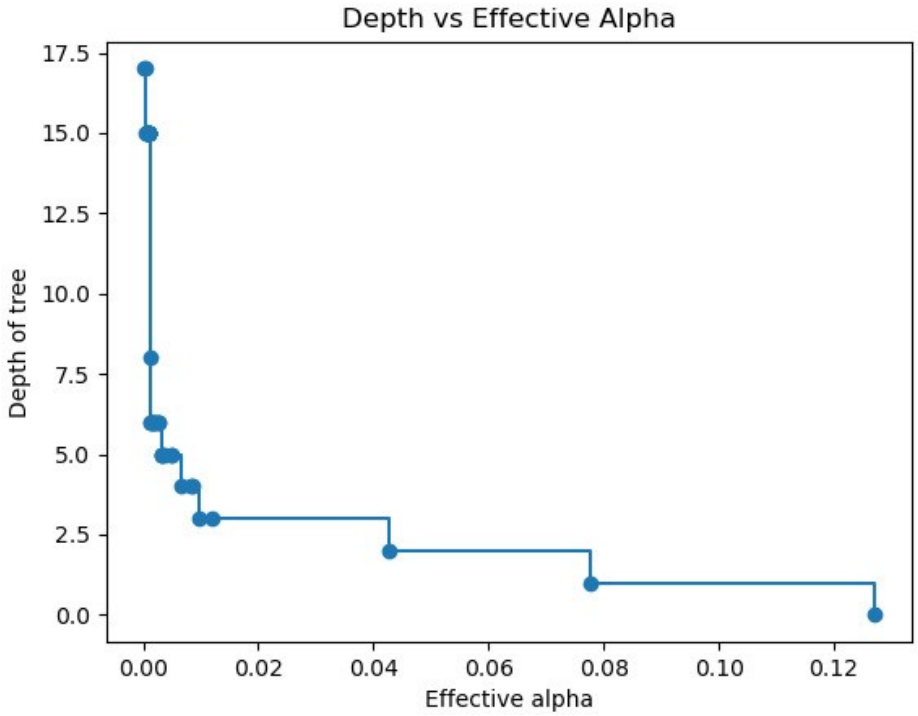
Graph of impurity of leaves vs effective alphas:



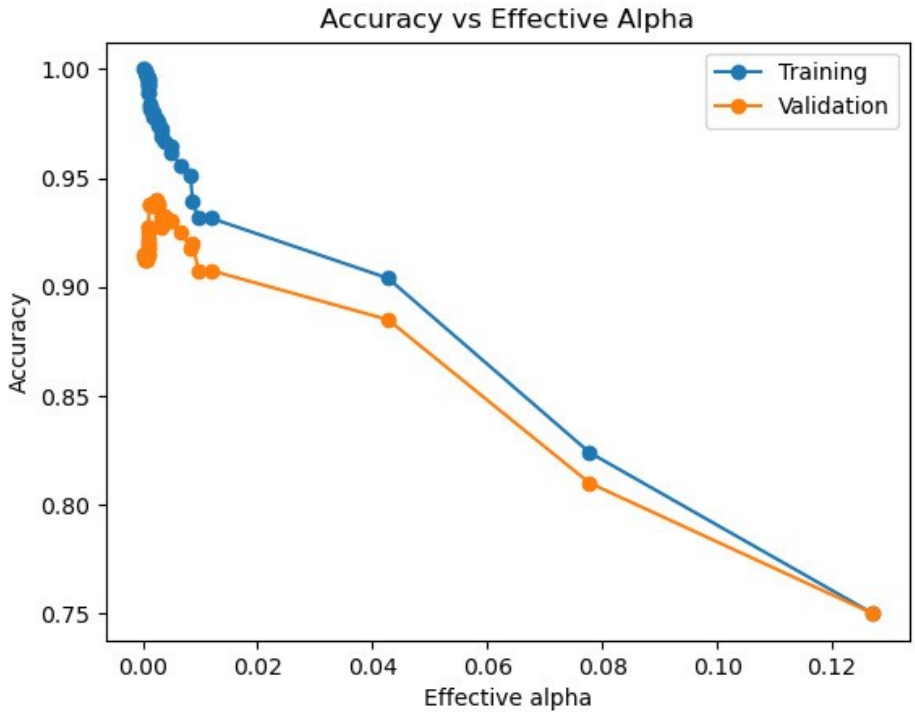
Graph of ccp_alpha vs number of nodes in a tree:



Graph of ccp_alphas vs depth of the tree :



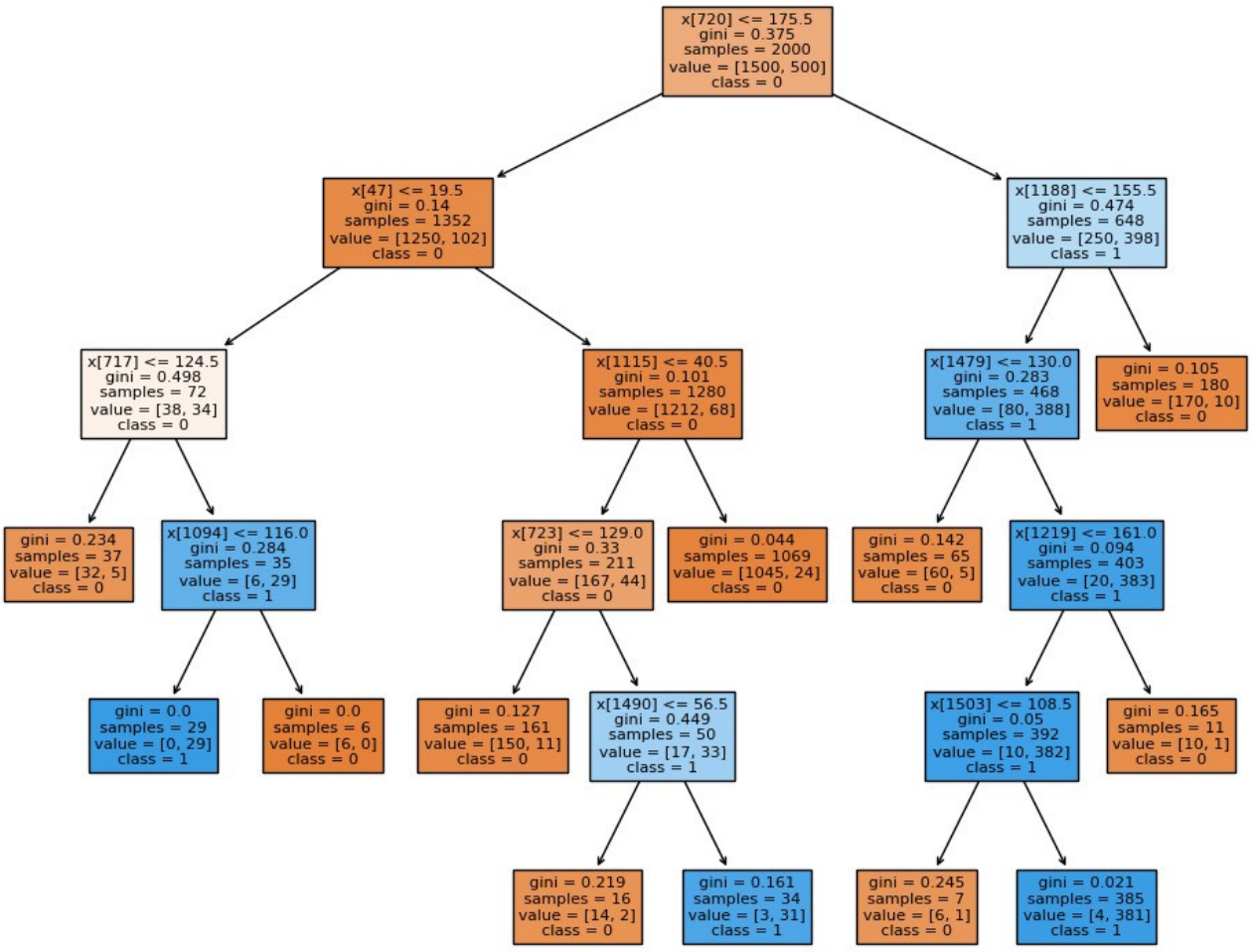
Graph of accuracy vs ccp_alphas for training and validation sets :



For the best pruned tree, the values are as follows:

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_d.py
Time taken to train the best tree: 383.7745282649994
Best ccp_accp_alpha_grid_search.best_params_lpha: {'ccp_alpha': 0.0037820512820512806}
Training accuracy: 0.967
Validation accuracy: 0.9325
```

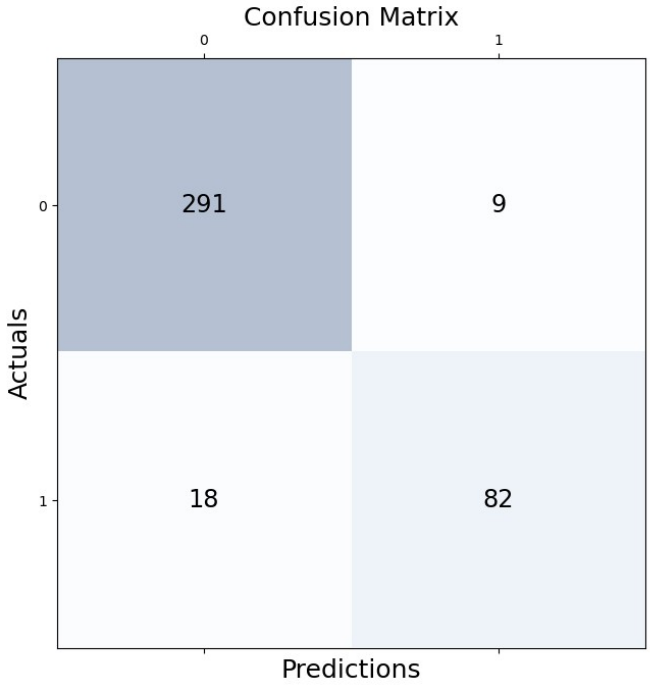
The time taken to find the best tree is 383 seconds.
The value of the best ccp_alpha = 0.00378205.
The training accuracy on the best pruned tree = 0.967.
The validation accuracy on the best pruned tree = 0.9325
<u>Best pruned decision tree :</u>



Observations :

The tree got pruned and the final decision tree is the perfect one with the higher accuracy after pruning with less overfitting.

Confusion matrix :



3.1(e) Random Forest

Random Forest is an extension of the Decision Tree Algorithm where multiple decision trees are grown in parallel with bootstrapped data. We will implement random forest using scikit-learn library.

Using the default hyperparameters, these are the accuracy values of what I got.

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_e.py
Default hyperparameters:
Training accuracy: 1.0
Training precision: 1.0
Training recall: 1.0
Validation accuracy: 0.9725
Validation precision: 1.0
Validation recall: 0.89
```

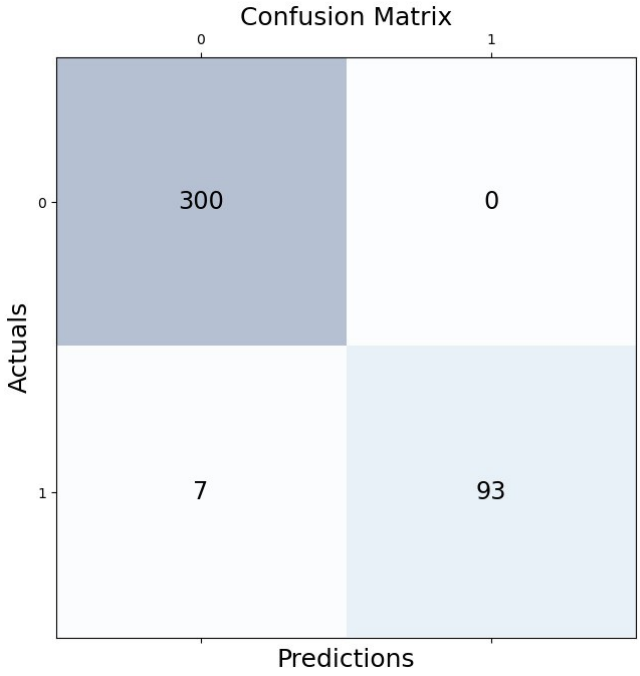
After performing Grid Search (with parameter cv=5) over given hyper-parameters list:
[n_estimators: {80,100,150,200}, criterion: {'gini', 'entropy'}, max_depth: {None,5,7,10}, min_samples_split: {5,7,10}], the values obtained are as follows:

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_e.py
Best hyperparameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 150}

Best hyperparameters performance:
Training accuracy: 1.0
Training precision: 1.0
Training recall: 1.0
Validation accuracy: 0.9825
Validation precision: 1.0
Validation recall: 0.93
```

The best hyperparameters are with criteria entropy, with max_depth = 10, min_samples_split=5 and with estimators = 150.

With the best set of hyperparameters,
The training accuracy is 1, the training precision is 1, training recall is 1.
The validation accuracy is 0.9825, validation precision is 1, validation recall is 0.93

Confusion matrix:



3.1(f) Gradient Boosted trees and XGBoost

Gradient Boost :

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_f.py
Best hyperparameters for gradient boost: {'max_depth': 5, 'n_estimators': 50, 'subsample': 0.4}
Gradient Boosting Classifier:
Best parameters: {'max_depth': 5, 'n_estimators': 50, 'subsample': 0.4}
Training accuracy: 1.0
Validation accuracy: 0.9675
Training precision: 1.0
Validation precision: 0.9675055164442576
Training recall: 1.0
Validation recall: 0.9675
Time taken for gb to execute: 2191.9909842014313
```

*The time taken for gradient boosting to get executed is **2191.99 seconds** which is approximately an hour **using n_jobs = -1.** Using n_jobs = -1, we will get the execution time decreased as it uses all the cores present in the CPU.*

The best set of hyperparameters for gradient boost is as follows:
Max_depth = 5, number of estimators = 50, subsample = 0.4

The validation accuracy is 0.9675.
The training accuracy is 1.
The validation precision is 0.9675055
The training precision is 1.
The validation recall is 0.9675.
The training recall is 1.

XGBoost:
XGBoost (Extreme Gradient Boosting) is functional gradient boosting based approach where an ensemble of "weak learners" (decision trees in our case) is used with the goal to construct a model with less bias, and better predictive performance.

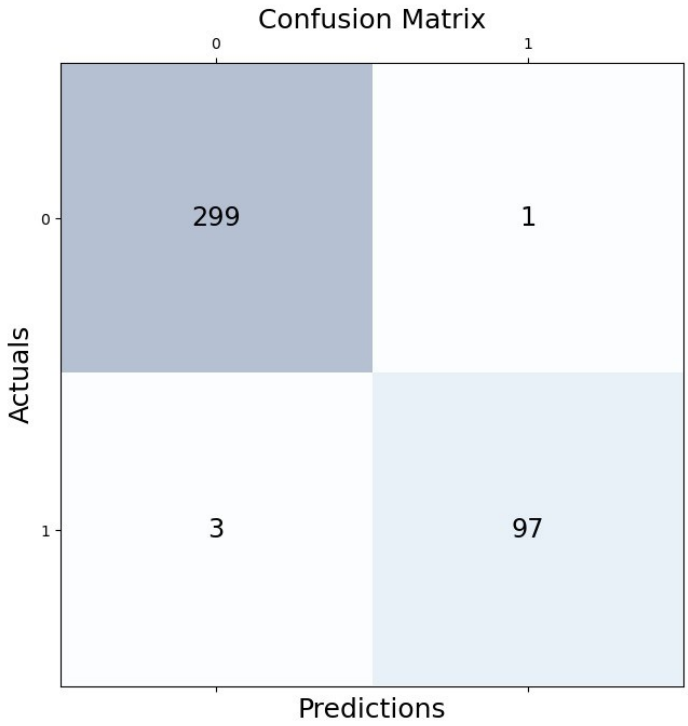
```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_1_f.py
Best hyperparameters for XGboost: {'max_depth': 10, 'n_estimators': 50, 'subsample': 0.6}

XGBoost Classifier:
Best parameters: {'max_depth': 10, 'n_estimators': 50, 'subsample': 0.6}
Training accuracy: 1.0
Validation accuracy: 0.99
Training precision: 1.0
Validation precision: 0.9899986484660088
Training recall: 1.0
Validation recall: 0.99
Time taken for xgb to execute: 659.971720457077
```

*The time taken for gradient boosting to get executed is **659.971 seconds using n_jobs = -1.** Using n_jobs = -1, we will get the execution time decreased as it uses all the cores present in the CPU.*

The best set of hyperparameters for gradient boost is as follows:
Max_depth = 10 , number of estimators = 50, subsample = 0.6

The validation accuracy is 0.99.
The training accuracy is 1.
The validation precision is 0.9899986.
The training precision is 1.
The validation recall is 0.99.
The training recall is 1.

Confusion matrix for XG boost:



3.1(g) Confusion matrices

All the confusion matrices in all the parts are shown above.

3.1(h) Best Implementation

The best implementation came for XGBoost. So, implemented that in competetive part.

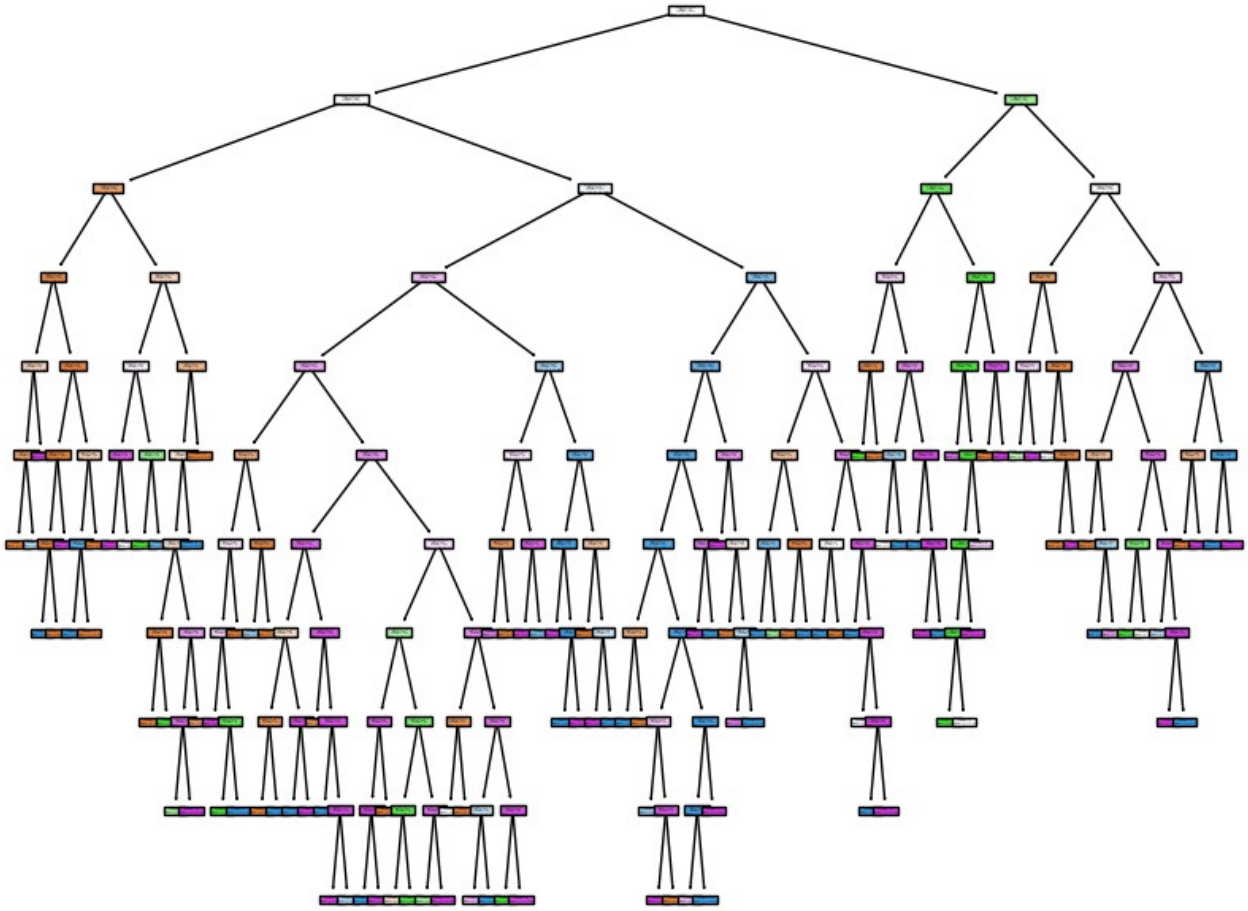
3.2 Multi Classification

3.2(a) Decision tree sklearn

With hyperparameters max depth set to 10 and min samples split set to 7, the decision tree classifier is implemented.

The time taken to train the model is 2.5590 seconds.

Decision tree :



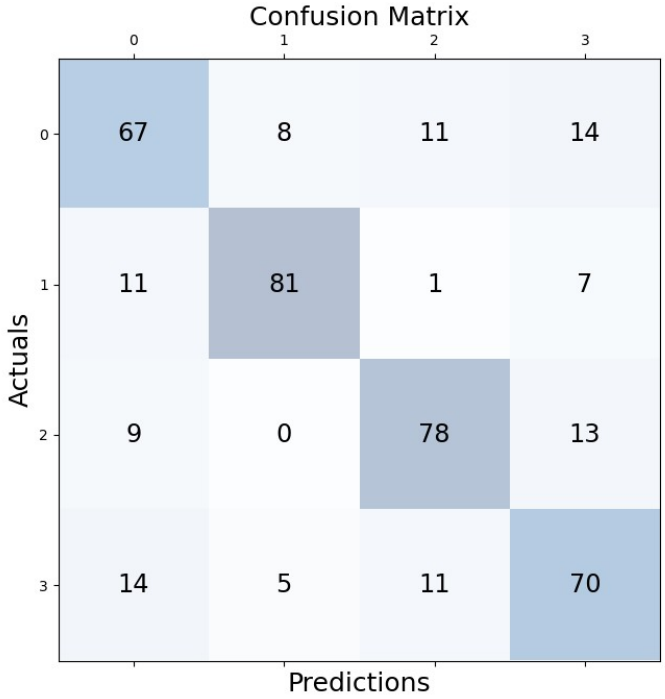
```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_2_a.py
2.5590384006500244
Training Time: 2.5590384006500244
Training Accuracy: 0.969
Training Precision: 0.9690976443728118
Training Recall: [0.97  0.972 0.962 0.972]
Validation Accuracy: 0.74
Validation Precision: 0.7426056537732333
Validation Recall: [0.67 0.81 0.78 0.7 ]
```

The training accuracy is 0.969.

The validation accuracy is 0.74.

The recall values are calculated using the average=None attribute in it.

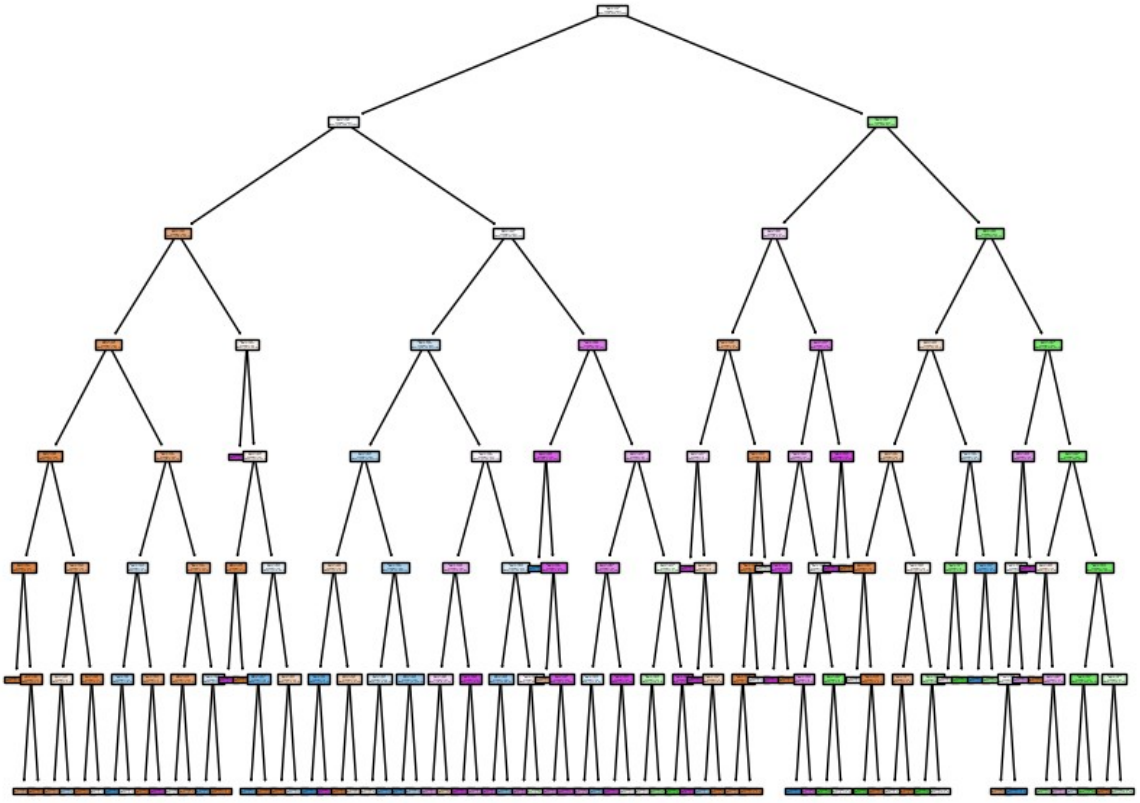
Confusion matrix:



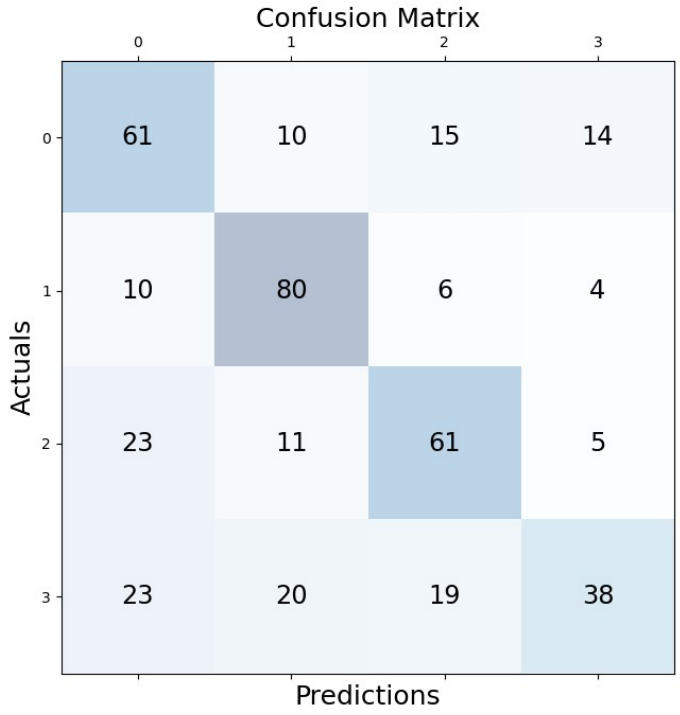
3.2(b) Decision Tree Grid Search and visualisation

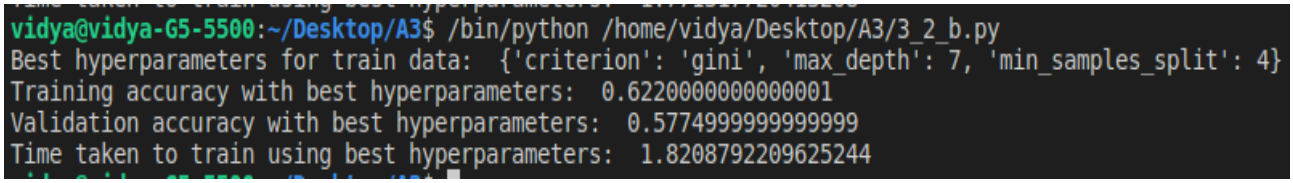
We select top 10 features from the data set. Using these features, we build a decision tree. This can be done by useing the **SelectKBest** class from scikit-learn's **feature selection.**
After that, we use grid search on the given parameters and find the best parameters.

Decision tree :



Confusion matrix:





The best set of hyperparameters is as follows :
Criterion : gini, max_depth : 7, min_samples_split = 4.
Training accuracy with best set of hyperparameters = 0.6220001
Validation accuracy with best set of hyperparameters = 0.57749999
The time taken to train = 1.8208 seconds.

<u>Comparison with part 3.2(a):</u>

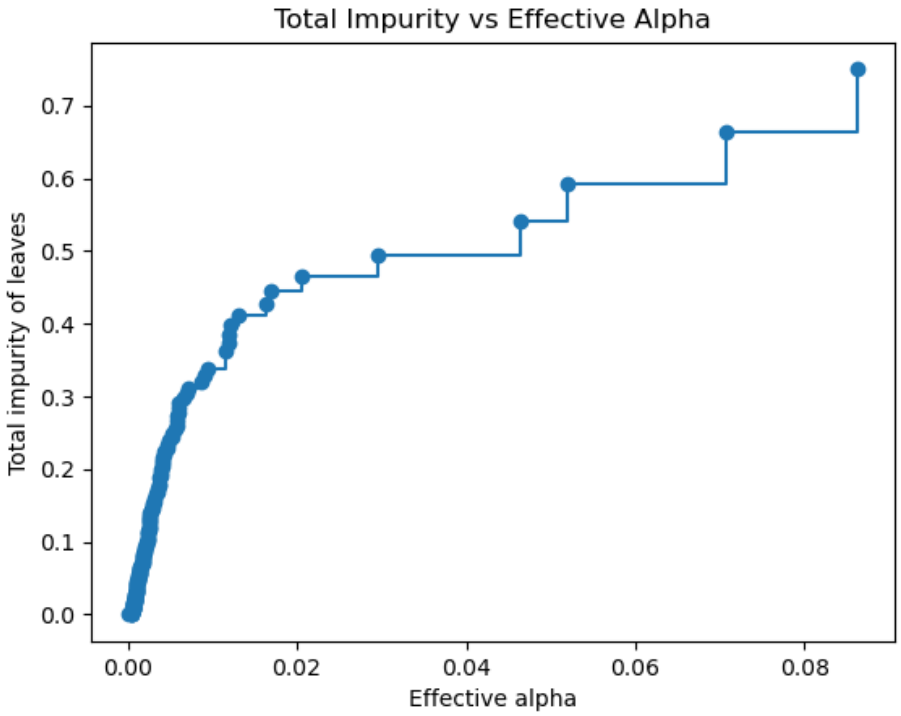
The decision tree in the a part is having more depth in the left side of the tree whereas in the present part, the depth is uniform overall in the tree.

The main difference is that the training and the validation accuracies got reduced.

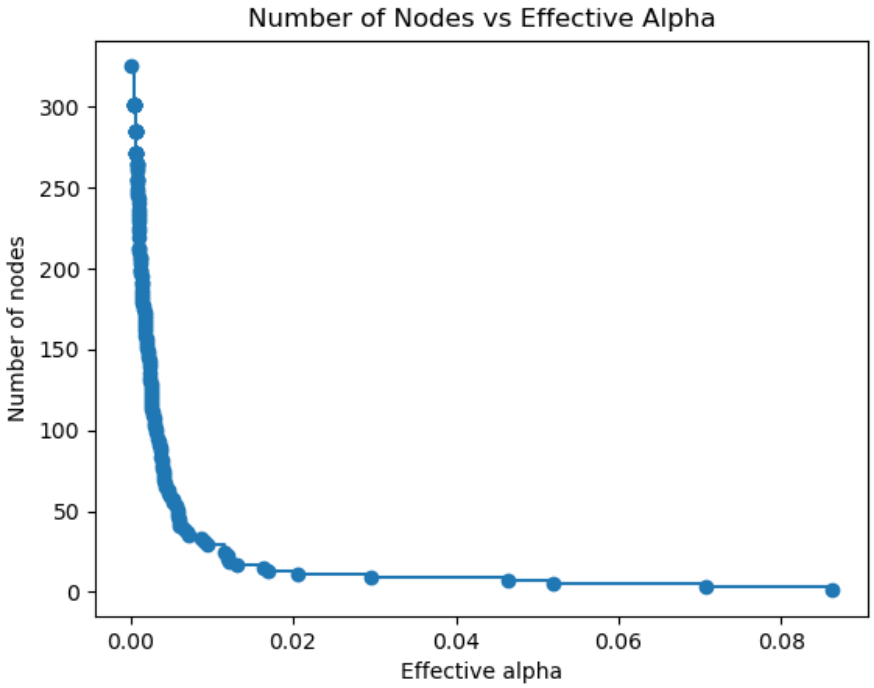
3.2(c) Decision Tree Post Pruning with Cost Complexity Pruning

We need to prune the tree inorder to reduce the complexity of the tree. We can reduce the size of the tree by reducing the number of links. For that, **we first need to calculate the effective alphas and corresponding total leaf impurities at each step of the pruning process. We can do this using the DecisionTreeClassifier.cost_complexity_pruning_path()**. *By checking the accuracy with the each ccp_alpha value obtained and creating a decision tree, we can find the optimal value of ccp_alpha.*

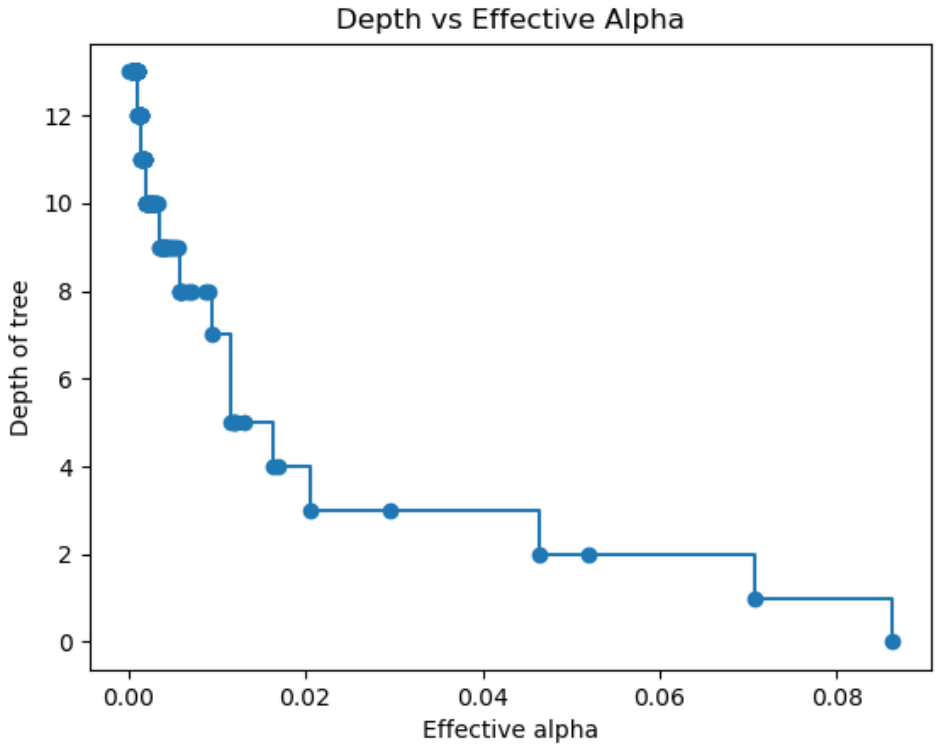
Graph of impurity of leaves vs effective alphas:



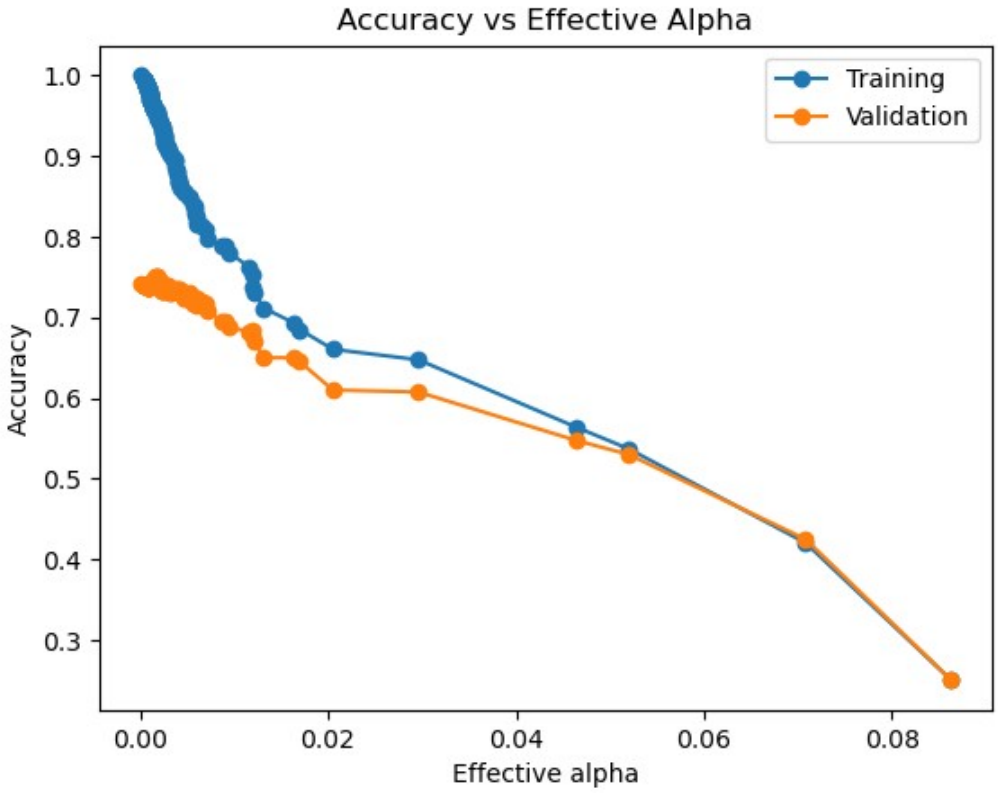
Graph of ccp_alpha vs number of nodes in a tree:



Graph of ccp_alphas vs depth of the tree :



Graph of accuracy vs ccp_alphas for training and validation sets :



For the best pruned tree, the values are as follows:

```
Time taken to train the best tree: 276.1533923149109
Best ccp_accp_alpha_grid_search.best_params_lpha: {'ccp_alpha': 0.004686666666666666}
Training accuracy: 0.8565
Validation accuracy: 0.725
```

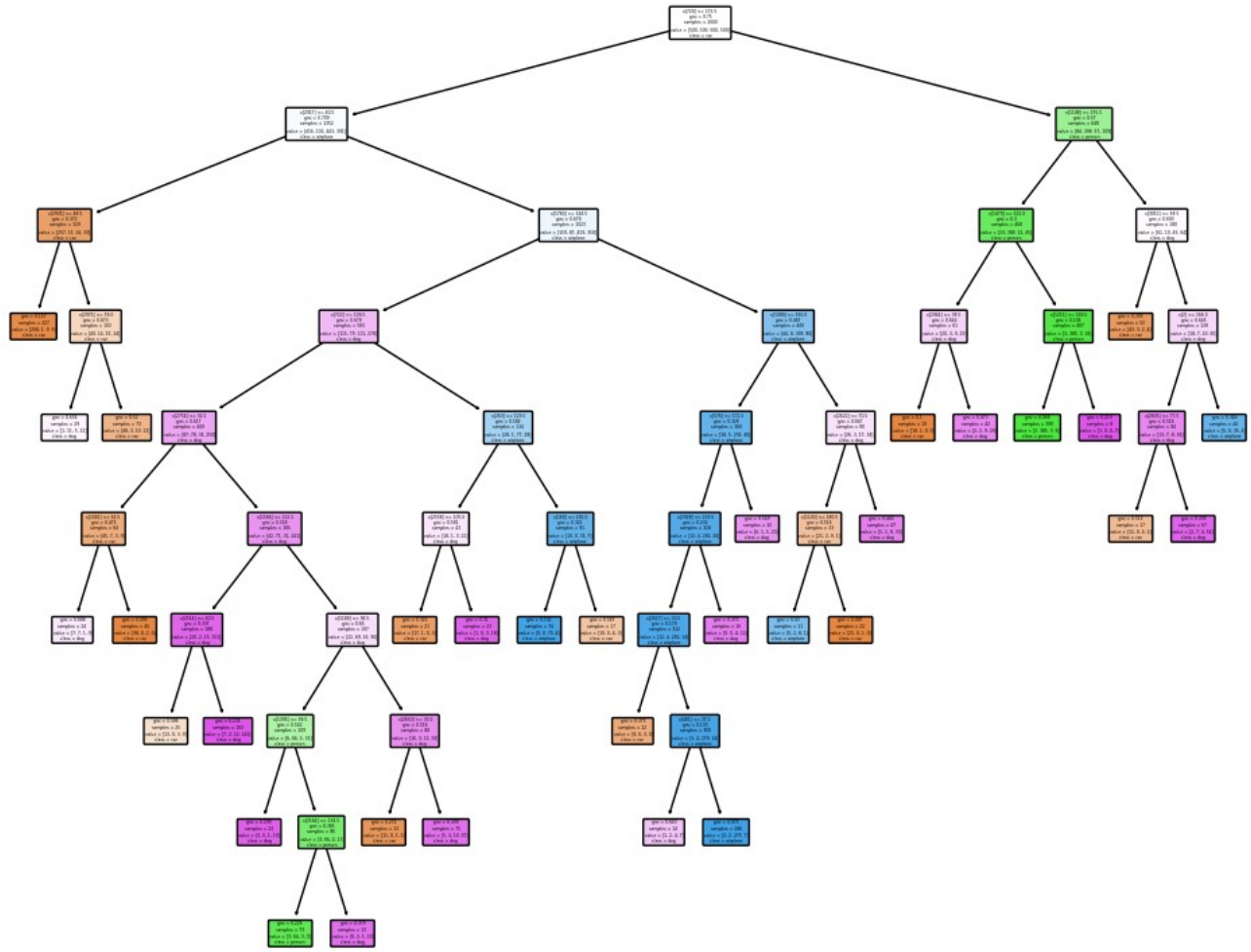
Time taken to find the best tree = 276.1533 seconds.

The value of the best ccp_alpha = 0.0046866

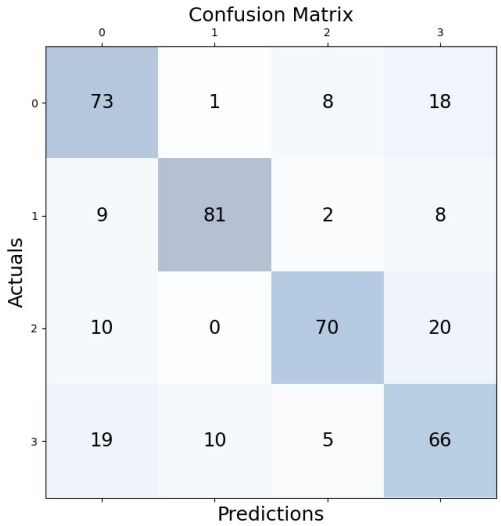
The training accuracy on the best pruned tree = 0.8565.

The validation accuracy on the best pruned tree = 0.725.

Best pruned decision tree :



Confusion matrix :



3.2(d) Random Forest

Random Forest is an extension of the Decision Tree Algorithm where multiple decision trees are grown in parallel with bootstrapped data. We will implement random forest using scikit-learn library.

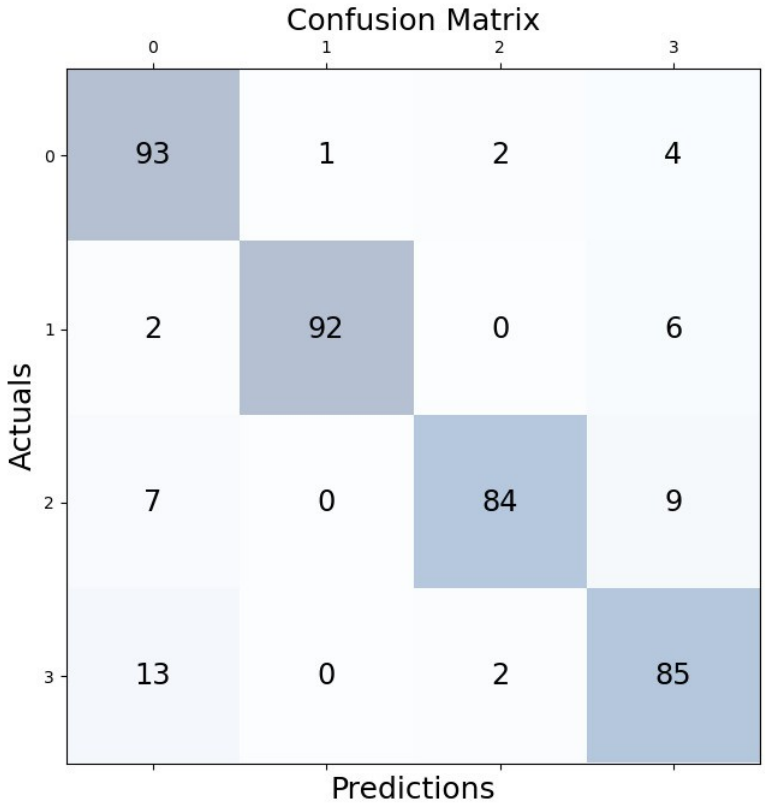
After performing Grid Search (with parameter cv=5) over given hyper-parameters list:
[n_estimators: {80,100,150,200}, criterion: {'gini', 'entropy'}, max_depth: {None,5,7,10}, min_samples_split: {5,7,10}], the values obtained are as follows:

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_2_d.py
Best hyperparameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 150}

Best hyperparameters performance:
Training Accuracy: 1.0
Training Precision: 1.0
Training Recall using macro: 1.0
Training Recall: [1. 1. 1. 1.]
Validation Accuracy: 0.885
Validation Precision: 0.8924490277137542
Validation Recall using macro: 0.885
Validation Recall: [0.93 0.92 0.84 0.85]
Time taken to execute: 277.82296991348267
```

The best hyperparameters are with criteria entropy, with max_depth = 10, min_samples_split=10 and with estimators = 150.

With the best set of hyperparameters,
The training accuracy is 1, the training precision is 1, training recall is 1.
The validation accuracy is 0.885, validation precision is 0.8924, validation recall is 0.93
*Time taken to train the best hyperparameter is **277.82 seconds using n_jobs = -1**. If n_jobs = -1, then the CPU uses all the cores present in the computer.*

Confusion matrix:



3.2(e) Gradient Boosted trees and XGBoost

XGBoost:
XGBoost (Extreme Gradient Boosting) is functional gradient boosting based approach where an ensemble of "weak learners" (decision trees in our case) is used with the goal to construct a model with less bias, and better predictive performance.

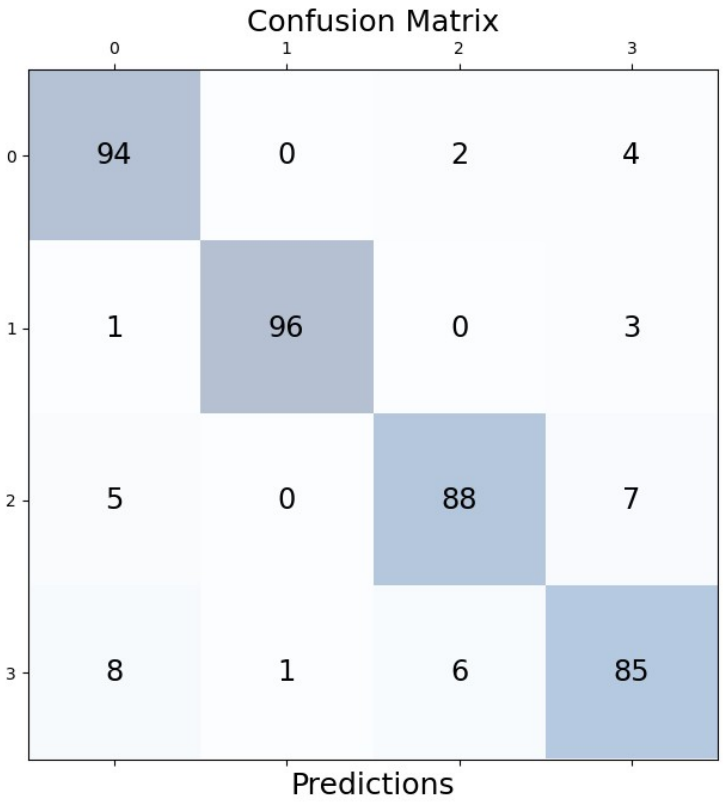
```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_2_e.py
Best hyperparameters for XGboost: {'max_depth': 7, 'n_estimators': 50, 'subsample': 0.6}

XGBoost Classifier:
Best parameters: {'max_depth': 7, 'n_estimators': 50, 'subsample': 0.6}
Training accuracy: 1.0
Validation accuracy: 0.9075
Training precision: 1.0
Validation precision: 0.908828404318095
Training recall: 1.0
Validation recall: 0.9075
Time taken for xgb to execute: 3854.2600376605988
```

*The time taken for gradient boosting to get executed is **3854.26 seconds using n_jobs = -1.** Using n_jobs = -1, we will get the execution time decreased as it uses all the cores present in the CPU.*

The best set of hyperparameters for gradient boost is as follows:
Max_depth = 7 , number of estimators = 50, subsample = 0.6

The validation accuracy is 0.9075.
The training accuracy is 1.
The validation precision is 0.908828.
The training precision is 1.
The validation recall is 0.9075.
The training recall is 1.

Confusion matrix for XG Boost :



Gradient Boost :

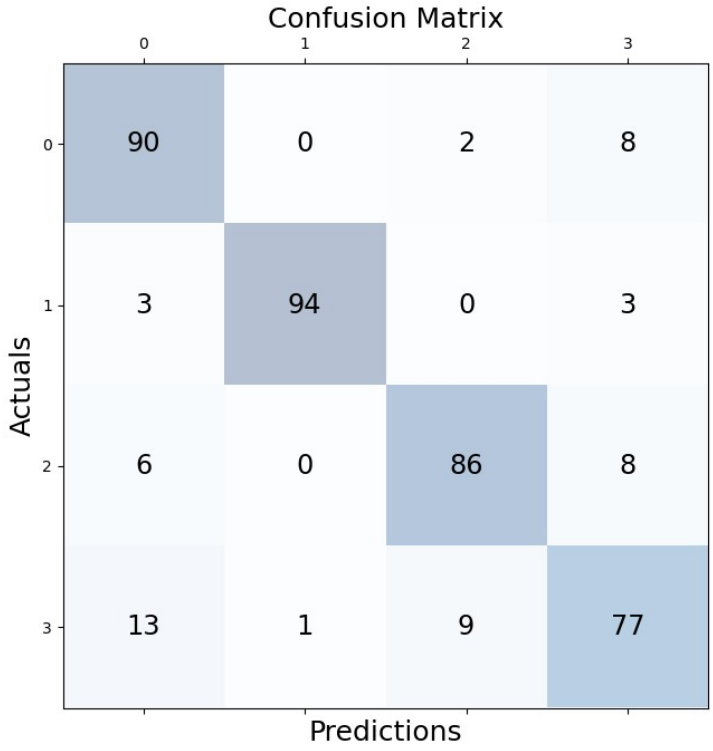
```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_2_e.py
Best hyperparameters for gradient boost: {'max_depth': 10, 'n_estimators': 50, 'subsample': 0.6}
```

*The time taken for gradient boosting to get executed is **4680 seconds using n_jobs = -1.** Using n_jobs = -1, we will get the execution time decreased as it uses all the cores present in the CPU.*

The best set of hyperparameters for gradient boost is as follows:
Max_depth = 10, number of estimators = 50, subsample = 0.6

The validation accuracy is 0.90.
The training accuracy is 1.
The validation precision is 0.958.
The training precision is 1.
The validation recall is 0.957.
The training recall is 1.

Confusion matrix :



3.2(f) Confusion matrices

I have written the confusion matrices in each and every part and also the decision tree diagram.

3.2(g) Real time application

Here, I have taken the images from my mobile phone which are jpeg. Then, I have converted them to 32*32 pixels. What I have got is 32*32*4 image. So, then I have converted that to 'RGB' image. Then, after training the data, I have predicted the classes of each and every face.

```
vidya@vidya-G5-5500:~/Desktop/A3$ /bin/python /home/vidya/Desktop/A3/3_2_g.py
The time taken to train :  2.7216718196868896
The predictions for the faces : [3 3 3 3 3 3 1 3 2 3]
Accuracy : 10.0
```

The time taken to fit the data is 2.721 seconds.
The predictions which I got for my faces are 8 dogs, 1 face and 1 airplane.
The prediction list is [3 3 3 3 3 3 1 3 2 3]
So, The accuracy for the predictions is 10%.

Its predicting the faces as dogs is what I have observed using different kinds of real images.

3.2(h) Best Implementation

The best implementation came for XGBoost. So, implemented that in competetive part.