CS60050
MACHINE LEARNING

Assignment 1

Animesh Jha - 19CS10070

Nisarg Upadhyaya - 19CS30031

1. Code

There are 4 files -

1. main.py

This file contains code to run the experiments. It has the following methods:

- select_best_tree: Finds the best tree over 10 random splits
- height_ablation: Finds the best depth limit for the tree by iterating over depths from 1 to 24
- run_gini_exp: Runs all experiments with measure set to gini
- run_entrop_exp: runs all experiments with measure set to entropy

2. utils.py

This file contains the utility functions for handling data and other calculations:

- get_column_names: Gets column names from the csv file
- get_X_y: Gets the data from the csv file, assumes last column contains target values
- train_test_split: Splits the data into train and test, can pass the size the split, whether to shuffle the data, and the random seed for splitting
- train_val_test_split: Similar to the above function but instead of two it splits into three parts
- check_purity: Checks if all the values in the passed array are the same
- classify_array: Finds the most common value in the array
- get_possible_breaks: Returns the possible break points for a given set of features
- create_children_np: For a given dataset, creates children based on a given attribute (column id) and a value for the same
- calc_entropy_np: Calculates entropy of a given array
- calc_info_gain: Given a dataset, attribute and its value to split based on, calculates the information gain by measuring the difference in entropies
- calc_gini_np: Calculates gini of a given array
- calc_gini_gain: Given a dataset, attribute and its value to split based on, calculates the gini gain by measuring the difference in gini
- get_best_split: For a given dataset and set of features returns the best possible split. Can also set impurity measure as "gini" or "entropy"
- assign_feature_type: checks if a feature is continuous or discrete based on a threshold. If the number of distinct values taken by that feature in the dataset is more than the threshold it is assigned a continuous type else discrete
- calc_accuracy: Calculates the accuracy of a prediction

- filter: This splits a given dataset into two parts given a target column and target value, in case of continuous variable this splits given dataset into a less_than_equal_to part and a greater_than part
- check_node: Checks if a node is pure, i.e., all target values are the same for this node

3. tree.py

This file contains two classes Node and DecisionTree and utility functions for printing the tree.

Class Node:

- Attributes:
    - node_id: A unique id for the node
    - attr_idx: The attribute this node is comparing
    - val: The value of the attribute it is comparing based on
    - attr_type: Whether the attribute is continuous or discrete
    - left: Left child of node
    - right: Right child of node
    - leaf: Whether the node is a leaf node or not
    - classification: The target value represented by the node
- make_leaf: Makes this node a **leaf** node and assigns the passed **classification**
- get_classification: Returns the **classification** of the node
- predict_node: Predicts the class of a new instance **X**. Does so recursively. If a leaf node is reached returns its classification else compares the **attr_idx** of **X** with **val** and accordingly calls **predict_node** on **left** child or **right** child
- dfs_count: counts the number of nodes in the subtree rooted at current node using depth first search
- prune_base: This is used to check if a node should be replaced by a leaf with the most common value as the label
- prune_rec: This function recursively prunes the tree nodes and returns the pruned node (and tree), this creates a new node and does not change the original node (and tree)

Class DecisionTree:

- Attributes:
    - measure: The impurity measure used in the tree, "gini" or "entropy"
    - root: Root of the tree
    - root_pruned: Root of the pruned tree
    - X and y: the training data
    - min_leaf_size: Minimum number of samples after which we should stop splitting and make the node a leaf. In the experiments this is set to 1
    - max_depth: Max depth of the decision tree
    - column_names: Names of the different features

    - type_arr: Array containing the type of the different features, "continuous" or "discrete"
- fit: Builds a decision tree which fits the training data X. Calls **build_tree** on the complete dataset **X** and assigns the returned node to the **root**.
- is_leaf: Given a dataset for a node checks if it should be a leaf node based on the purity of the target values and **min_leaf_size**
- build_tree: This function recursively builds the decision tree, it checks if the current node should be a leaf or not, if not it finds the best split for the node and then recursively computes its left and right children and returns the node.
- predict: Given new instances predicts their labels. Calls the **predict_node** function on the **root**.
- pruned_predict: Same as **predict** but calls **predict_node** function on the **root_pruned**
- calc_accuracy: Given test data, uses the **predict** function to predict the labels and compares them with the given labels to get the accuracy
- calc_pruned_accuracy: Same as **calc_accuracy** but predicts using **pruned_predict**
- print_tree: Prints the tree in a readable format
- count_nodes: Counts the number of nodes in the tree
- post_prune: Prunes the tree using **prune_rec** and assigns the returned node to **root_pruned**

4. diabetes.csv

The given dataset. It has the following attributed, all of which are continuous variables:

1) Pregnancies
2) Glucose
3) BloodPressure
4) SkinThickness
5) Insulin
6) BMI
7) DiabetesPedigreeFunction
8) Age
9) Outcome -- Label

Procedure followed

We use the standard ID3 algorithm, we only take binary splits, that is for discrete variables we check == and != and for continuous variables given a value we split the data on the basis of <= and >. For a node we first check if it should be leaf node, (check if depth is more than max depth, all variables belong to the same class or number of examples to split are less than the minimum specified leaf size). If not a leaf we calculate all possible splits given the remaining data, then we choose the best split given the specified metric (entropy or gini) and then recursively calculate the nodes left and right subtrees. We take 80:20 train test splits, and for validation we take 60:20:20 train test validation splits.

For 80:20 splits
X_train: (614, 8)
X_test: (154, 8)
y_train: (614,)
y_test: (154,)

For 60:20:20 splits
X_train: (460, 8)
X_test: (154, 8)
X_val: (154, 8)
y_train: (460,)
y_test: (154,)
y_val: (154,)

We do the following experiments

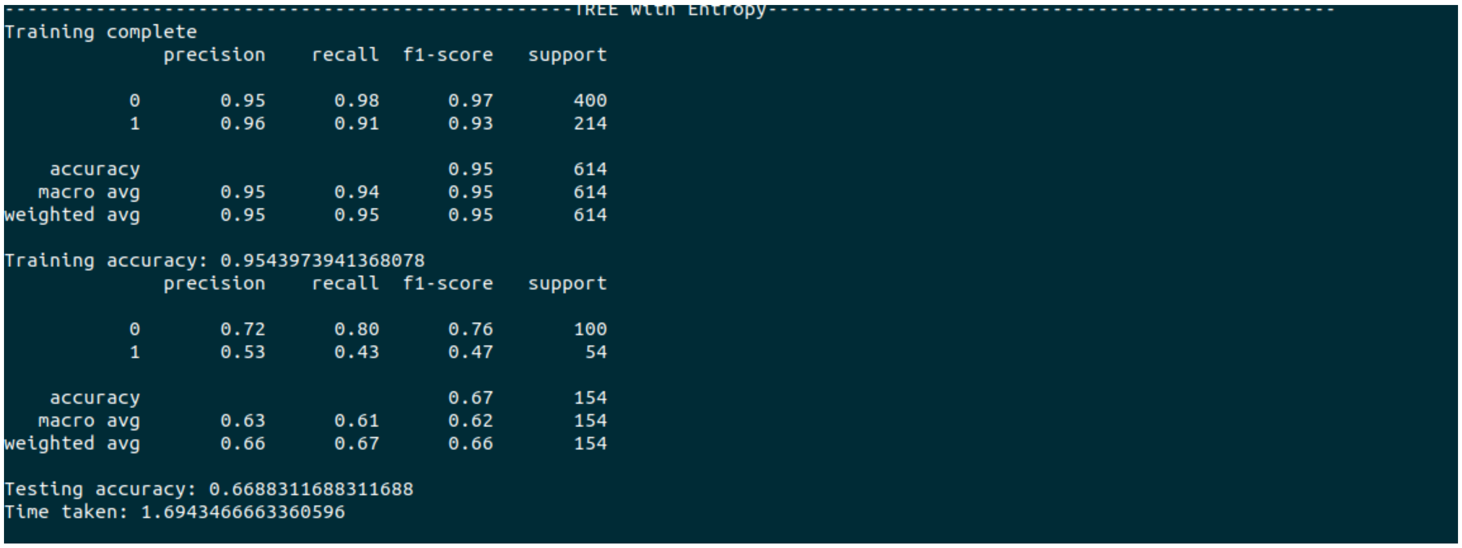
1. We construct a tree with depth=10 on a 80:20 split and print its classification report
2. We now create 10 random 60:20:20 splits, construct tree with depth=10 and choose the split with the best test accuracy
3. On the best split obtained in experiment 2, we do depth and number of nodes analysis
4. Now using the validation split obtained in experiment 2 we perform post pruning on the tree obtained in 2.

How to run the code?

After installing all dependencies just run `python main.py` and enjoy! README has more details.

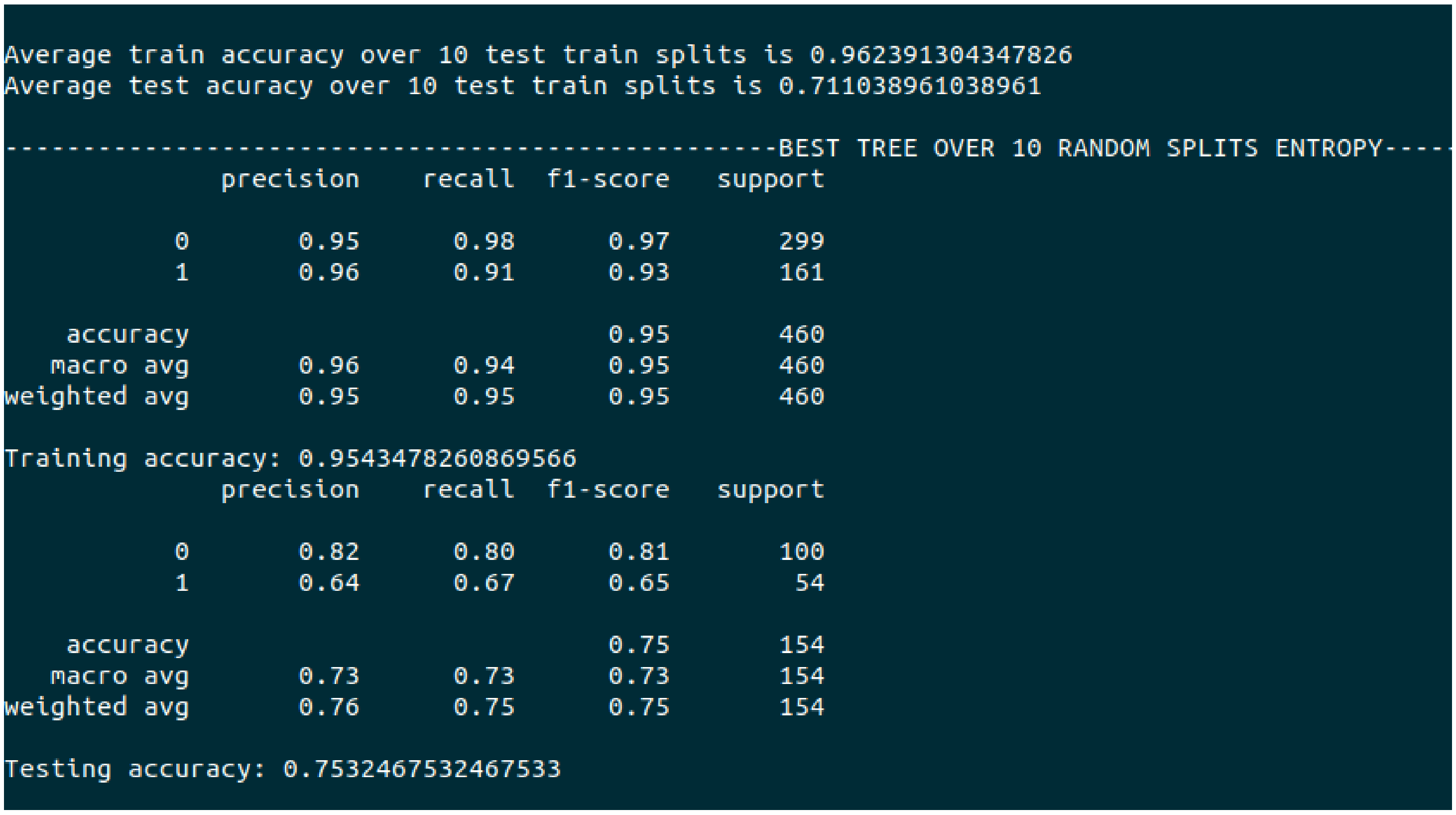
2. Experiments using Entropy

1) Tree constructed using 80:20 split, using entropy, max depth of the tree = 10



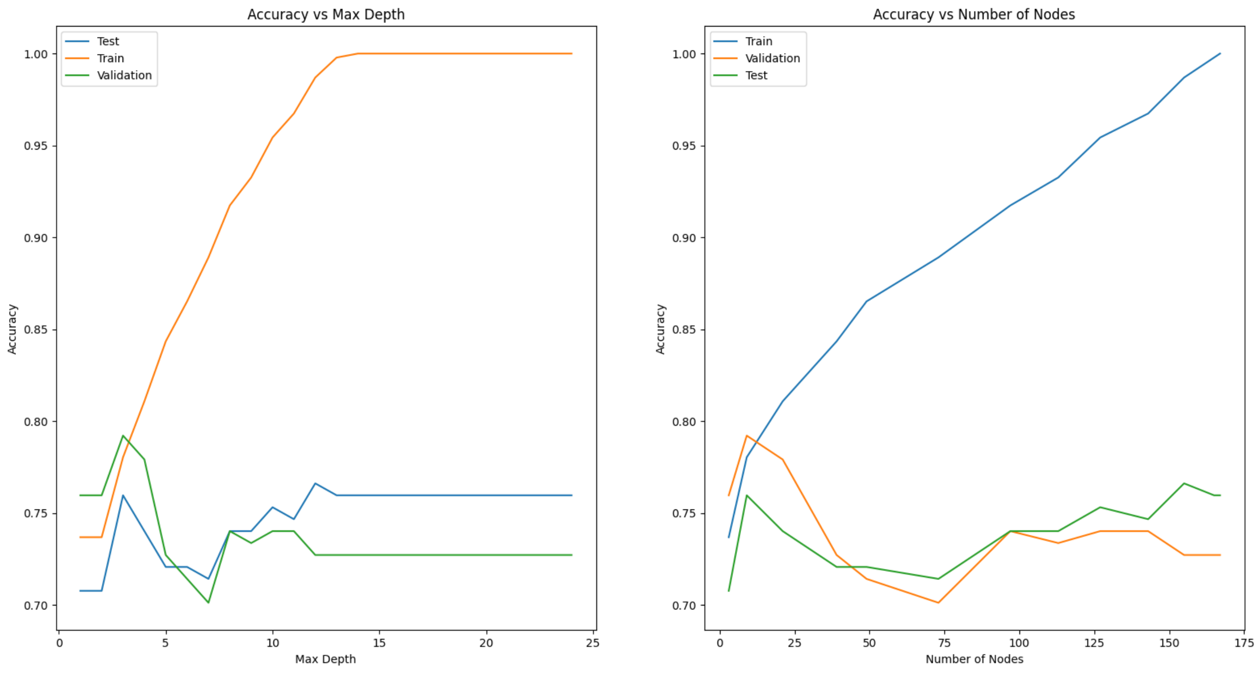
```
------------------------------------------------TREE with Entropy------------------------------------------------
Training complete
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       400
           1       0.96      0.91      0.93       214

    accuracy                           0.95       614
   macro avg       0.95      0.94      0.95       614
weighted avg       0.95      0.95      0.95       614

Training accuracy: 0.9543973941368078
              precision    recall  f1-score   support

           0       0.72      0.80      0.76       100
           1       0.53      0.43      0.47        54

    accuracy                           0.67       154
   macro avg       0.63      0.61      0.62       154
weighted avg       0.66      0.67      0.66       154

Testing accuracy: 0.6688311688311688
Time taken: 1.6943466663360596
```

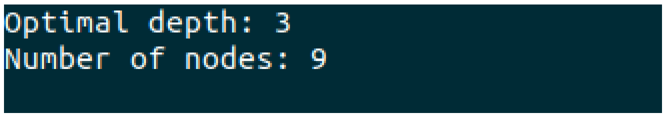
2) Accuracy over 10 random 60:20:20 splits, max depth = 10



```

Average train accuracy over 10 test train splits is 0.962391304347826
Average test acuracy over 10 test train splits is 0.711038961038961

-------------------------------------------------BEST TREE OVER 10 RANDOM SPLITS ENTROPY----
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       299
           1       0.96      0.91      0.93       161

    accuracy                           0.95       460
   macro avg       0.96      0.94      0.95       460
weighted avg       0.95      0.95      0.95       460

Training accuracy: 0.9543478260869566
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       100
           1       0.64      0.67      0.65        54

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.76      0.75      0.75       154

Testing accuracy: 0.7532467532467533
```

3) Accuracy vs Depth and Accuracy vs Number of Nodes analysis:

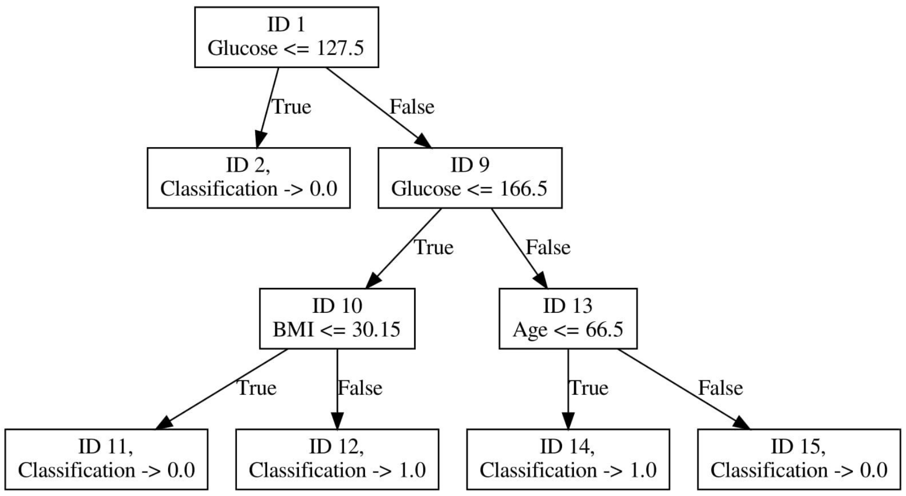




```
Optimal depth: 3
Number of nodes: 9
```

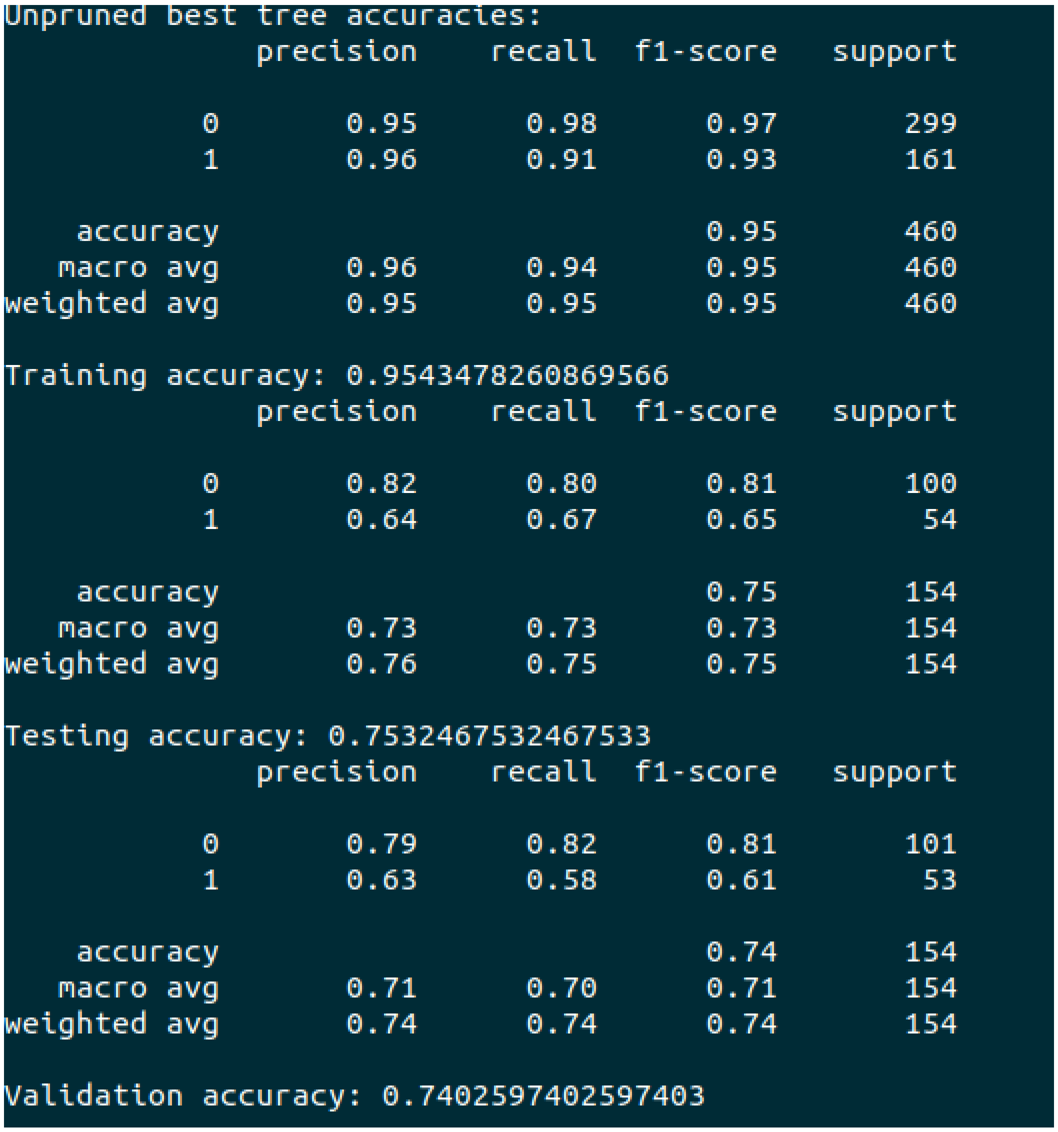
For this we use 60:20:20 splits

The optimal depth for validation accuracy is 3 and 9 nodes. At higher depths the training accuracy is higher, but the validation and test accuracies fall, this is due to the tree overfitting the data. The tree at the optimal depth looks like this:



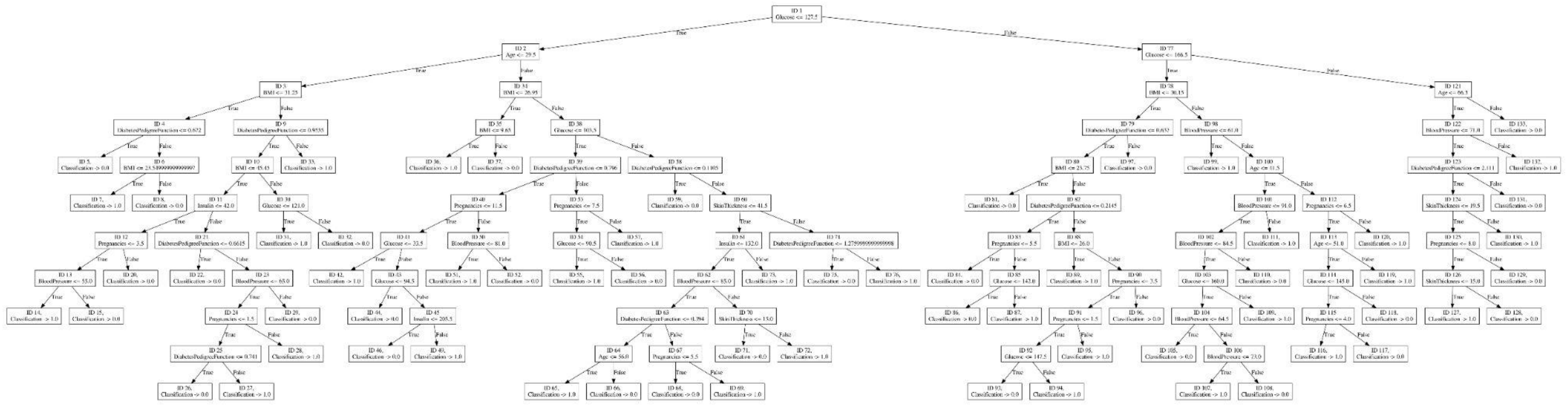
4) We now apply post pruning to the best tree obtained in experiment 2, at this point note that the max depth of the tree obtained in experiment 2 is 10 and the optimal depth is less than 10.

Before pruning the accuracies are as follow:

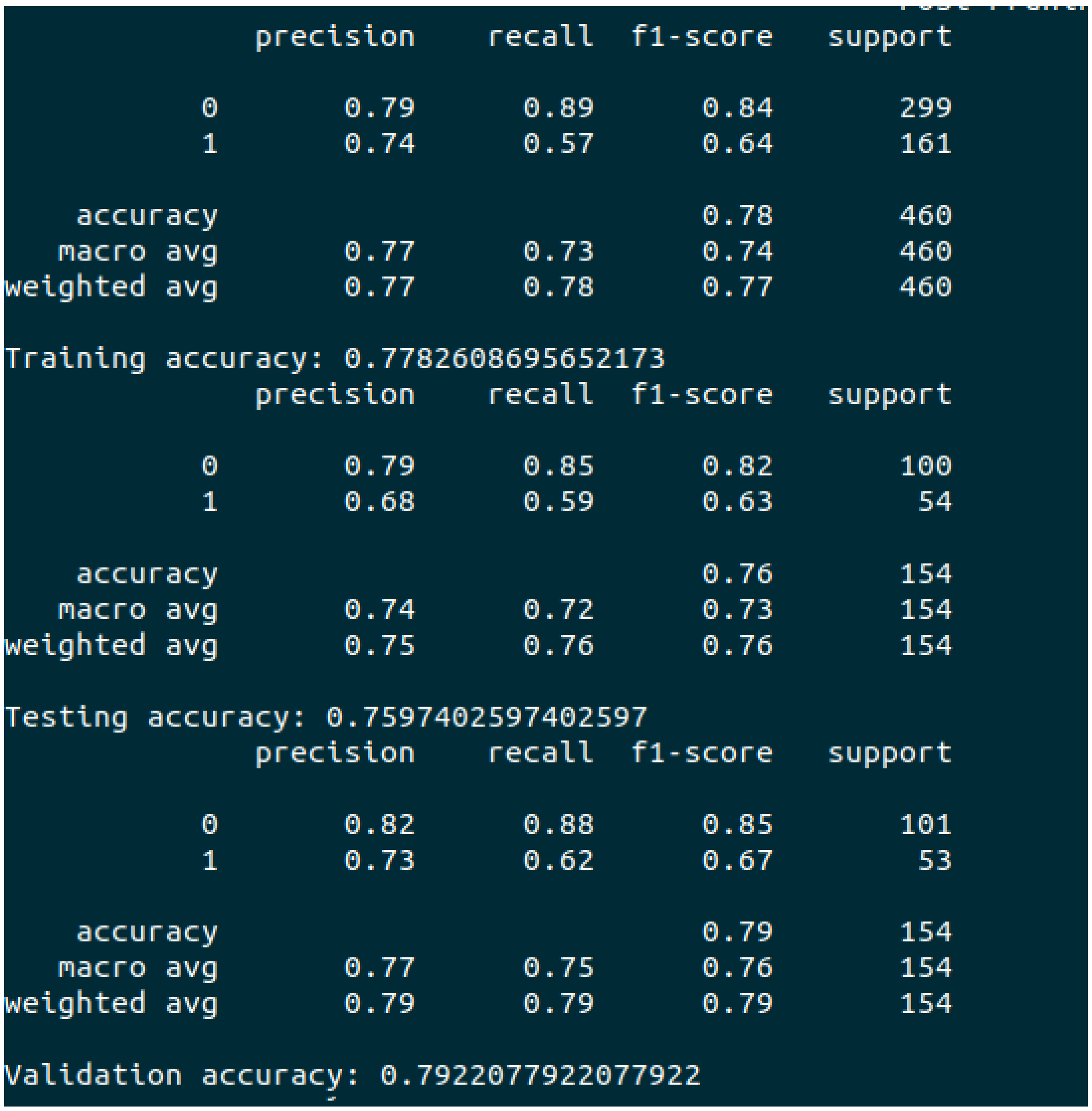


```
Unpruned best tree accuracies:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       299
           1       0.96      0.91      0.93       161

    accuracy                           0.95       460
   macro avg       0.96      0.94      0.95       460
weighted avg       0.95      0.95      0.95       460

Training accuracy: 0.9543478260869566
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       100
           1       0.64      0.67      0.65        54

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.76      0.75      0.75       154

Testing accuracy: 0.7532467532467533
              precision    recall  f1-score   support

           0       0.79      0.82      0.81       101
           1       0.63      0.58      0.61        53

    accuracy                           0.74       154
   macro avg       0.71      0.70      0.71       154
weighted avg       0.74      0.74      0.74       154

Validation accuracy: 0.7402597402597403
```

The tree looks like this (a clearer PDF can be found in the attached submission):

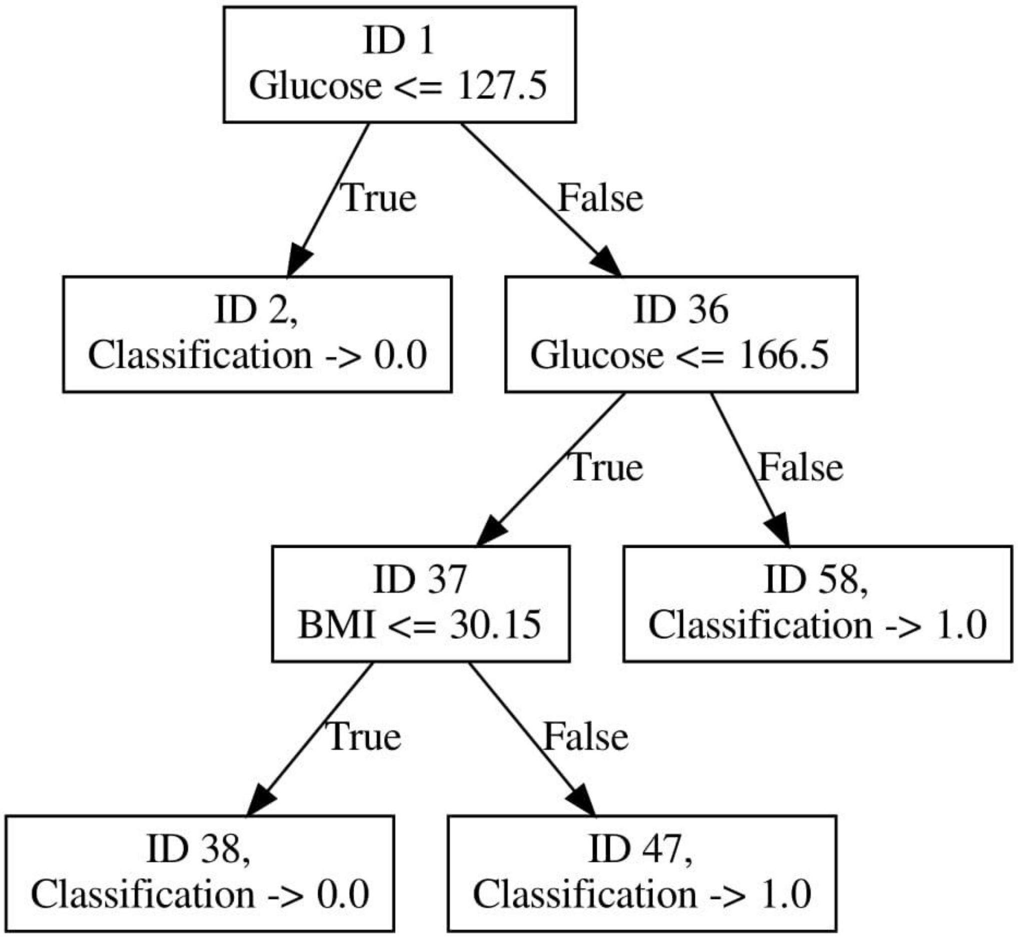


After pruning we observe that the validation and test accuracy increase, this is because pruning reduces the overfit of the depth=10 tree and reduces the model complexity. After pruning the accuracies are as follows:



```
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       299
           1       0.74      0.57      0.64       161

    accuracy                           0.78       460
   macro avg       0.77      0.73      0.74       460
weighted avg       0.77      0.78      0.77       460

Training accuracy: 0.7782608695652173
              precision    recall  f1-score   support

           0       0.79      0.85      0.82       100
           1       0.68      0.59      0.63        54

    accuracy                           0.76       154
   macro avg       0.74      0.72      0.73       154
weighted avg       0.75      0.76      0.76       154

Testing accuracy: 0.7597402597402597
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       101
           1       0.73      0.62      0.67        53

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.79      0.79      0.79       154

Validation accuracy: 0.7922077922077922
```

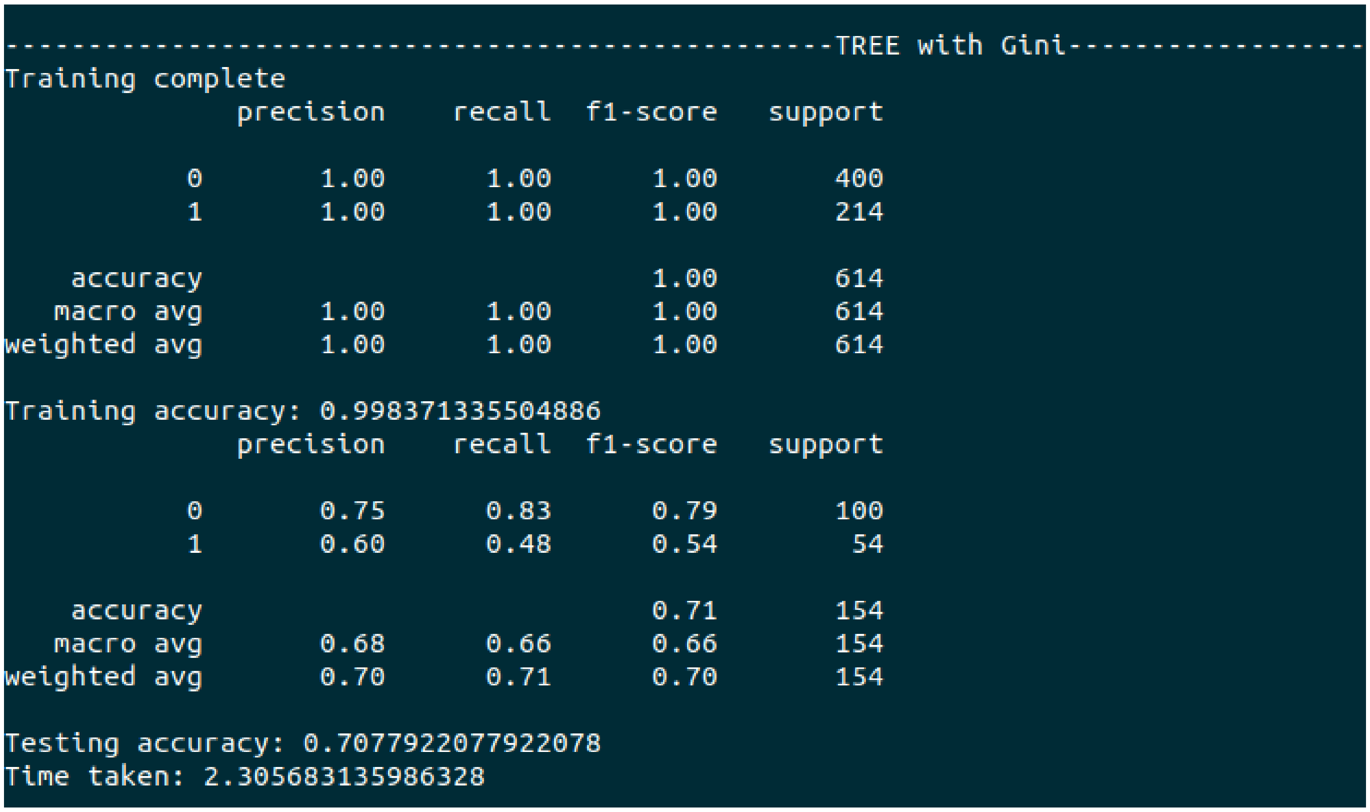
Post pruning the tree looks like this:



Observe that post pruning the tree with max depth 10, we get a tree which is very close to the optimal tree. This shows that pruning helped us to improve an overfitting tree.

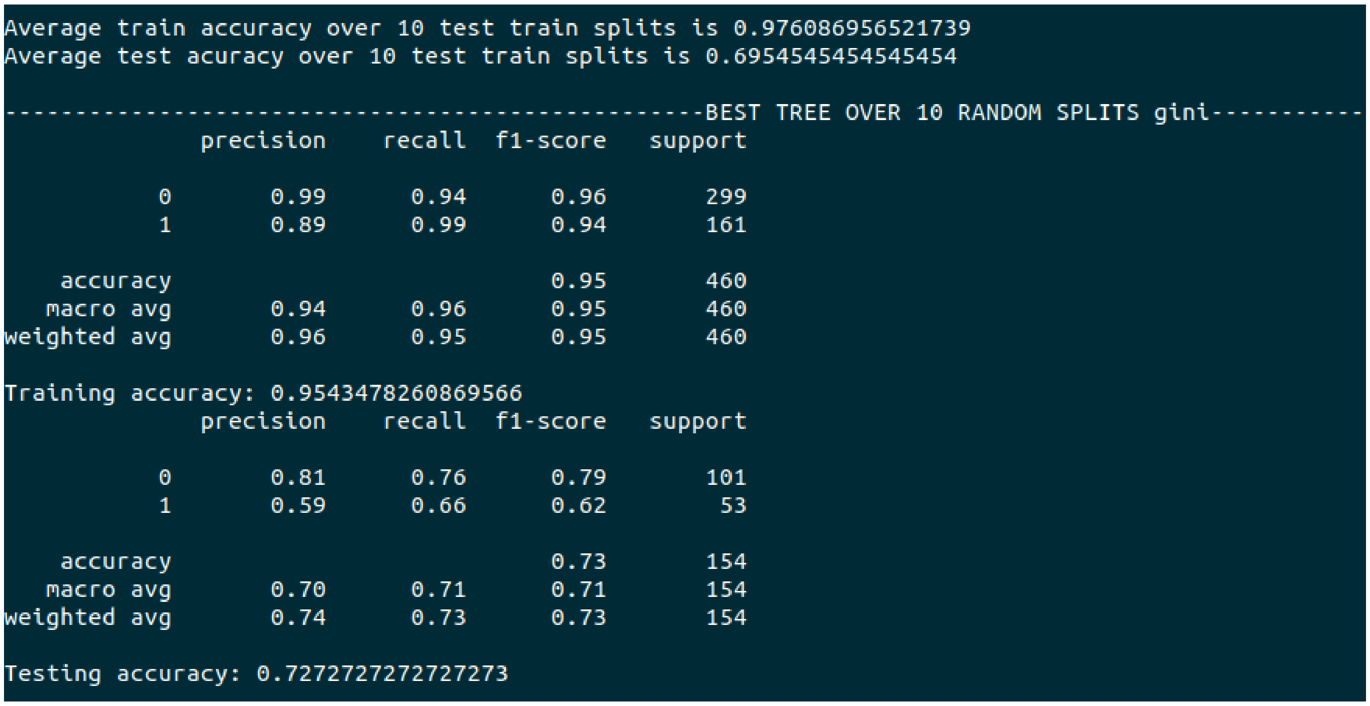
3. Experiments using Gini Index

1) Tree constructed using 80:20 split, using gini index, max depth of the tree = 10



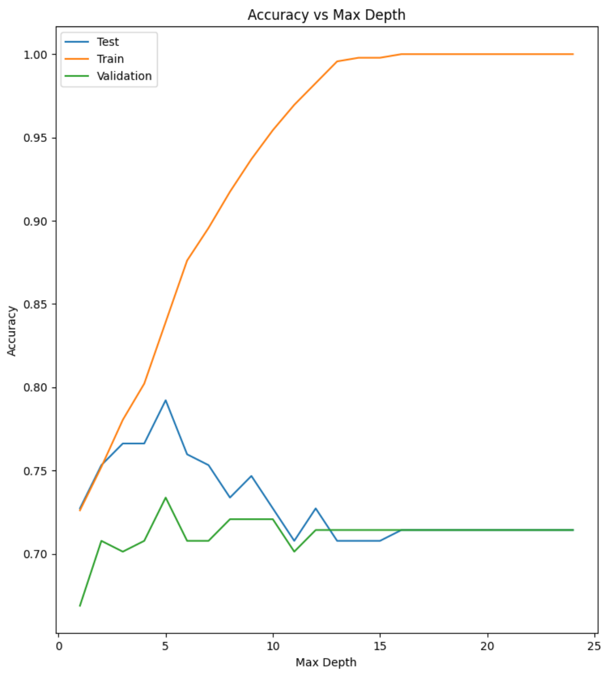
```
-------------------------------------------------TREE with Gini------------------
Training complete
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       400
           1       1.00      1.00      1.00       214

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614

Training accuracy: 0.998371335504886
              precision    recall  f1-score   support

           0       0.75      0.83      0.79       100
           1       0.60      0.48      0.54        54

    accuracy                           0.71       154
   macro avg       0.68      0.66      0.66       154
weighted avg       0.70      0.71      0.70       154

Testing accuracy: 0.7077922077922078
Time taken: 2.305683135986328
```

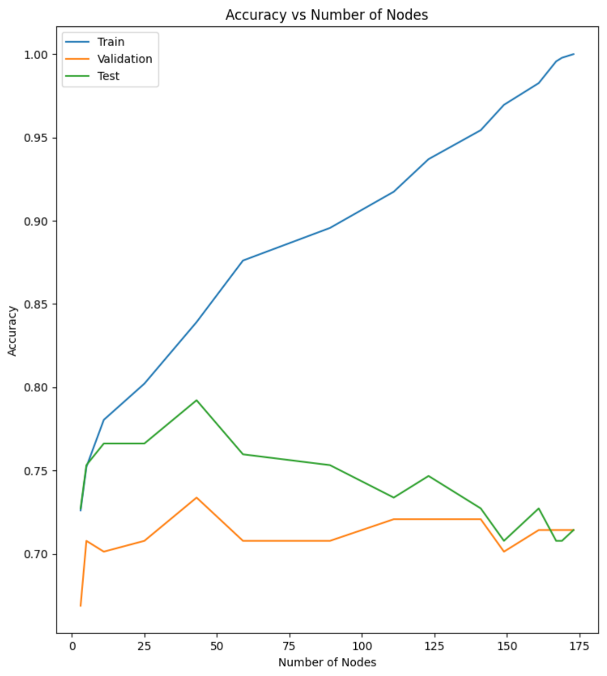
2) Accuracy over 10 random 60:20:20 splits, max depth = 10

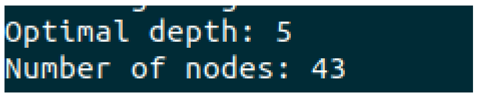


```
Average train accuracy over 10 test train splits is 0.976086956521739
Average test acuracy over 10 test train splits is 0.6954545454545454

-----------------------------------------------BEST TREE OVER 10 RANDOM SPLITS gini----------
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       299
           1       0.89      0.99      0.94       161

    accuracy                           0.95       460
   macro avg       0.94      0.96      0.95       460
weighted avg       0.96      0.95      0.95       460

Training accuracy: 0.9543478260869566
              precision    recall  f1-score   support

           0       0.81      0.76      0.79       101
           1       0.59      0.66      0.62        53

    accuracy                           0.73       154
   macro avg       0.70      0.71      0.71       154
weighted avg       0.74      0.73      0.73       154

Testing accuracy: 0.7272727272727273
```

3) Accuracy vs Depth and Accuracy vs Number of Nodes analysis

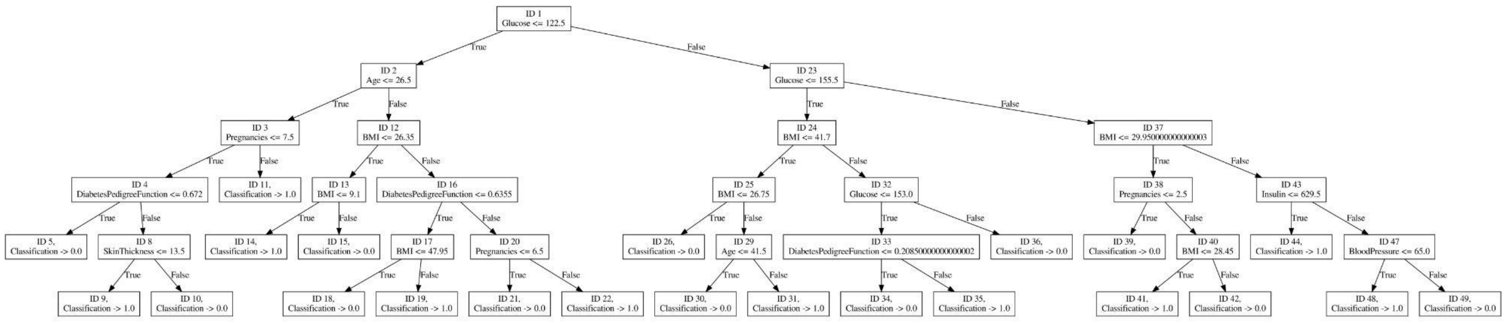






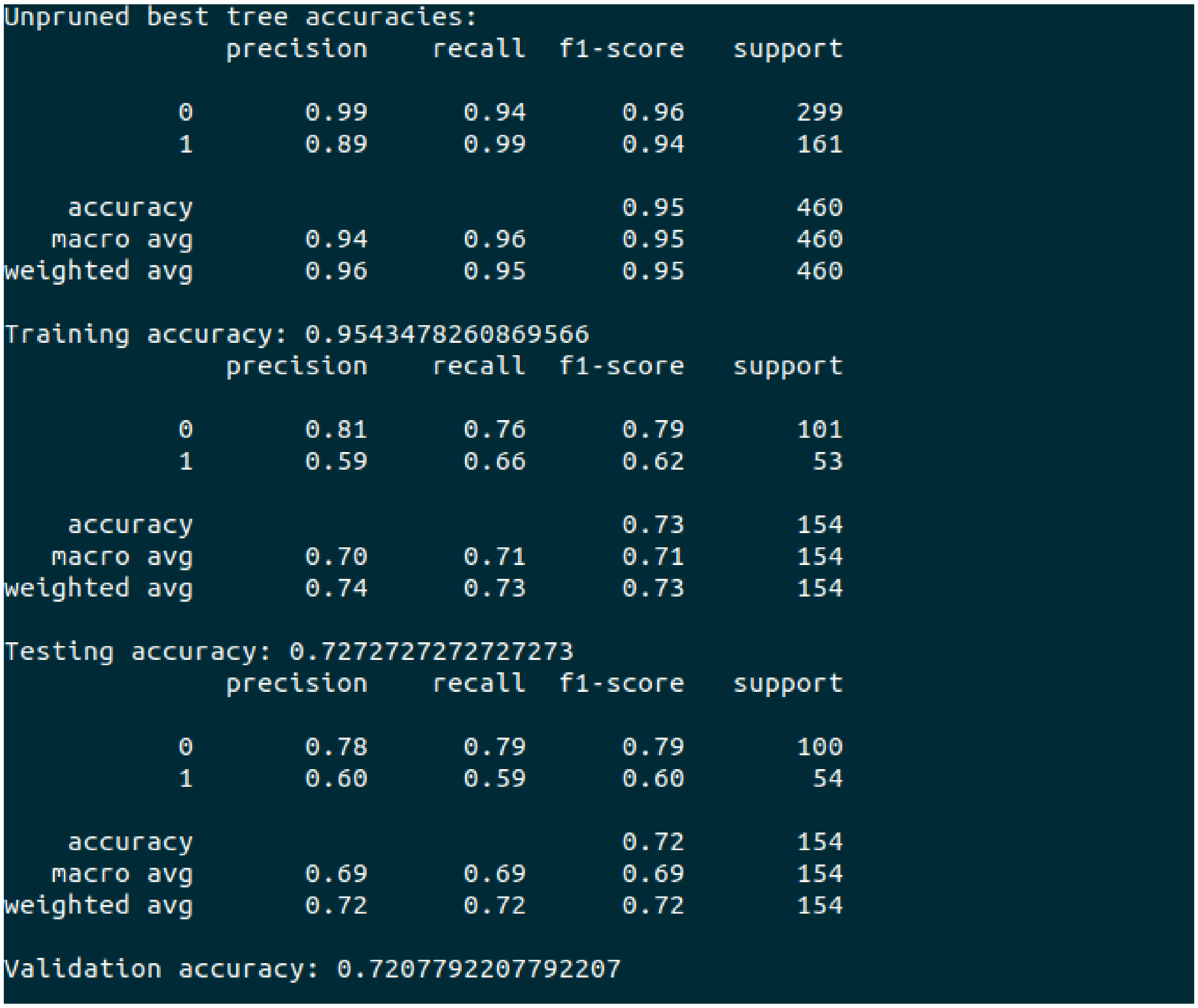
```
Optimal depth: 5
Number of nodes: 43
```

The optimal depth for validation accuracy is 5 and 43 nodes . At higher depths the training accuracy is higher, but the validation and test accuracies fall, this is due to the tree overfitting the data. The tree at optimal depth looks like this:


ID 1 Glucose <= 122.5
ID 2 Age <= 26.5
ID 3 Pregnancies <= 7.5
ID 4 DiabetesPedigreeFunction <= 0.672
ID 5, Classification -> 0.0
ID 8 SkinThickness <= 13.5
ID 9, Classification -> 1.0
ID 10, Classification -> 0.0
ID 11, Classification -> 1.0
ID 12 BMI <= 26.35
ID 13 BMI <= 9.1
ID 14, Classification -> 1.0
ID 15, Classification -> 0.0
ID 16 DiabetesPedigreeFunction <= 0.6355
ID 17 BMI <= 47.95
ID 18, Classification -> 0.0
ID 19, Classification -> 1.0
ID 20 Pregnancies <= 6.5
ID 21, Classification -> 0.0
ID 22, Classification -> 1.0
ID 23 Glucose <= 155.5
ID 24 BMI <= 41.7
ID 25 BMI <= 26.75
ID 26, Classification -> 0.0
ID 29 Age <= 41.5
ID 30, Classification -> 0.0
ID 31, Classification -> 1.0
ID 32 Glucose <= 153.0
ID 33 DiabetesPedigreeFunction <= 0.20850000000000002
ID 34, Classification -> 0.0
ID 35, Classification -> 1.0
ID 36, Classification -> 0.0
ID 37 BMI <= 29.950000000000003
ID 38 Pregnancies <= 2.5
ID 39, Classification -> 0.0
ID 40 BMI <= 28.45
ID 41, Classification -> 1.0
ID 42, Classification -> 0.0
ID 43 Insulin <= 629.5
ID 44, Classification -> 1.0
ID 47 BloodPressure <= 65.0
ID 48, Classification -> 1.0
ID 49, Classification -> 0.0
True
False

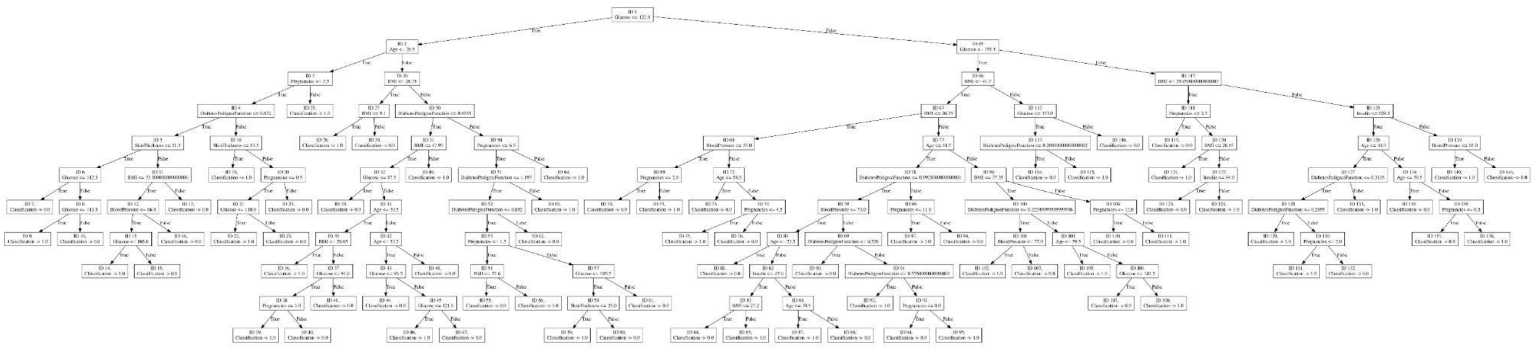
4) We now apply post pruning to the best tree obtained in experiment 2, note that the max depth of the tree obtained in experiment 2 is 10 and the optimal depth is less than 10. Pruning should reduce the complexity of the tree.

Before post pruning the accuracies are as follow:

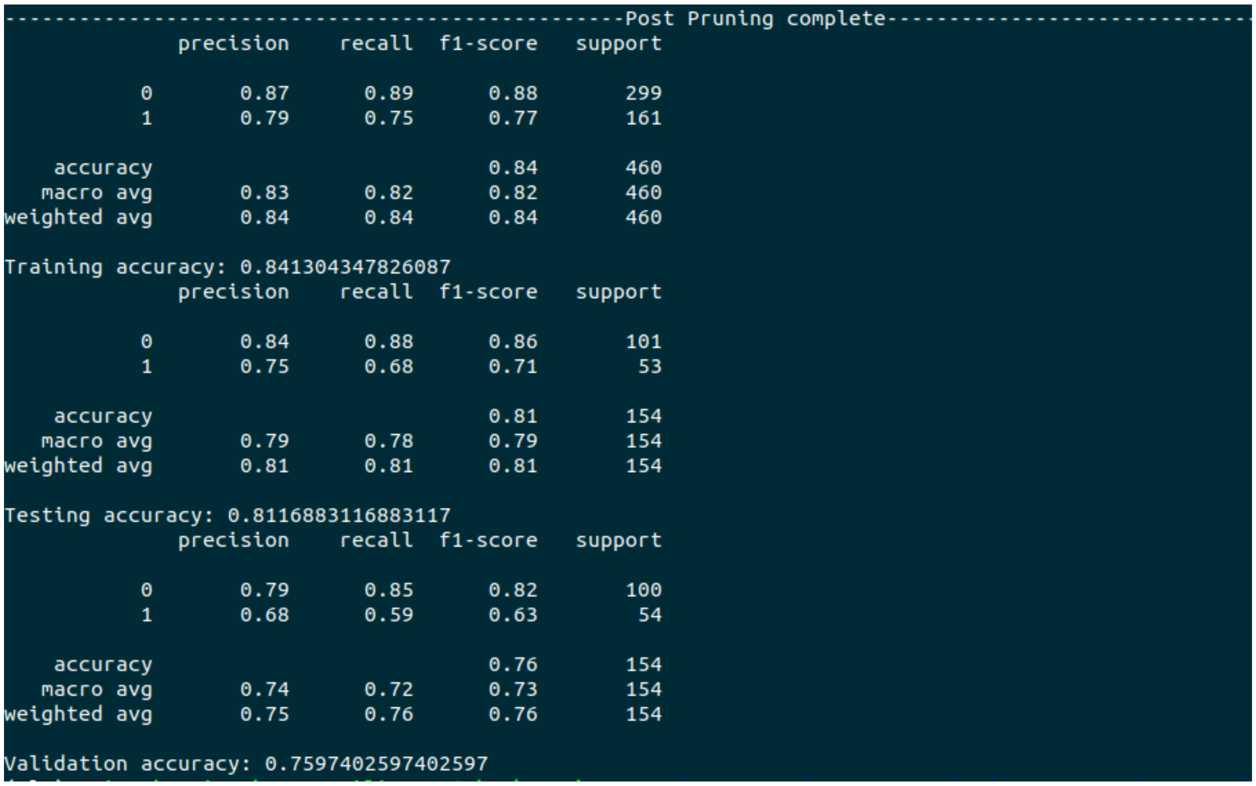


```
Unpruned best tree accuracies:
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       299
           1       0.89      0.99      0.94       161

    accuracy                           0.95       460
   macro avg       0.94      0.96      0.95       460
weighted avg       0.96      0.95      0.95       460

Training accuracy: 0.9543478260869566
              precision    recall  f1-score   support

           0       0.81      0.76      0.79       101
           1       0.59      0.66      0.62        53

    accuracy                           0.73       154
   macro avg       0.70      0.71      0.71       154
weighted avg       0.74      0.73      0.73       154

Testing accuracy: 0.7272727272727273
              precision    recall  f1-score   support

           0       0.78      0.79      0.79       100
           1       0.60      0.59      0.60        54

    accuracy                           0.72       154
   macro avg       0.69      0.69      0.69       154
weighted avg       0.72      0.72      0.72       154

Validation accuracy: 0.7207792207792207
```

The tree looks like this (a clearer PDF can be found in the attached submission):

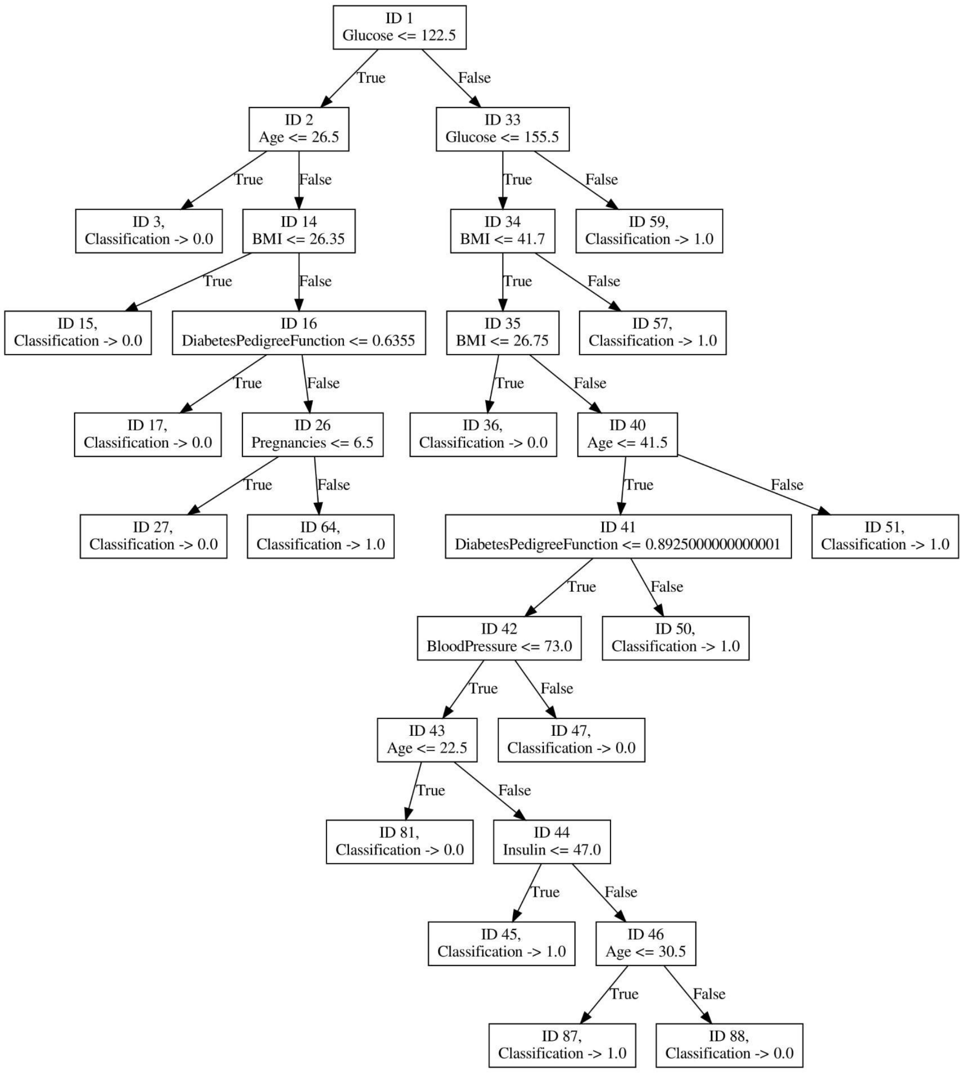


After post pruning we observe that the validation and test accuracy increase, this is because post pruning reduces the overfit of the depth=10 tree. After pruning the accuracies are as follows:



```
--------------------------------------------------Post Pruning complete------------------------------
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       299
           1       0.79      0.75      0.77       161

    accuracy                           0.84       460
   macro avg       0.83      0.82      0.82       460
weighted avg       0.84      0.84      0.84       460

Training accuracy: 0.841304347826087
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       101
           1       0.75      0.68      0.71        53

    accuracy                           0.81       154
   macro avg       0.79      0.78      0.79       154
weighted avg       0.81      0.81      0.81       154

Testing accuracy: 0.8116883116883117
              precision    recall  f1-score   support

           0       0.79      0.85      0.82       100
           1       0.68      0.59      0.63        54

    accuracy                           0.76       154
   macro avg       0.74      0.72      0.73       154
weighted avg       0.75      0.76      0.76       154

Validation accuracy: 0.7597402597402597
```

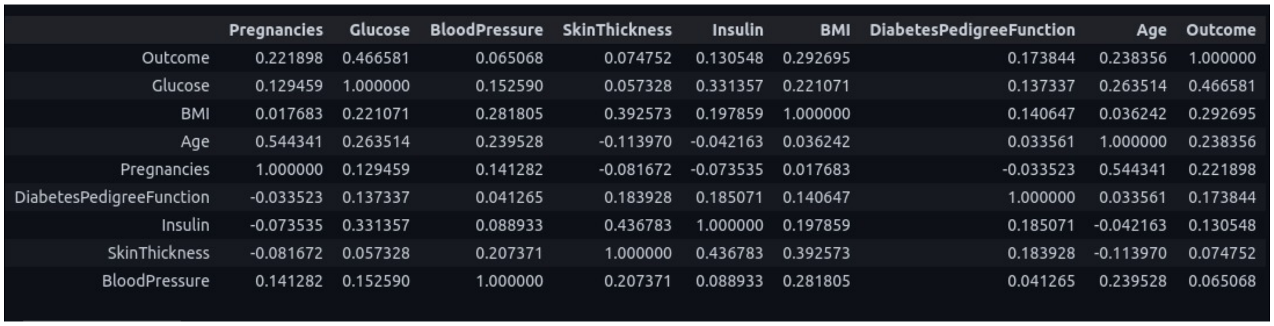
Post pruning most of the tree becomes shallow with depth around the optimal depth, i.e., 5. The overall complexity of the tree is reduced. The tree looks like this post pruning:



4. Analysis

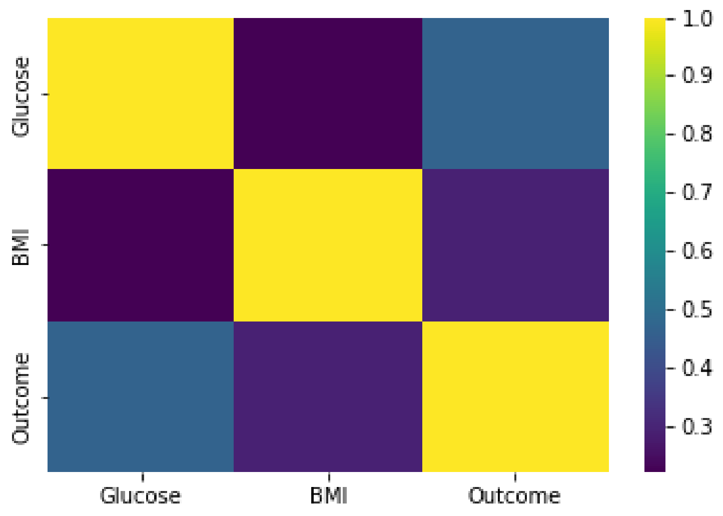
The decision tree obtained in all cases has two variables ‘**glucose**’ and ‘**BMI**’ in the top levels of the tree, in fact the accuracy of the tree with depth < 3 is within 5 percentage points of the optimal tree.

From this we hypothesize that ‘**Outcome**’, ‘**BMI**’ and ‘**Glucose**’ are correlated variables. To verify this we calculated the correlation matrix of the entire dataset.

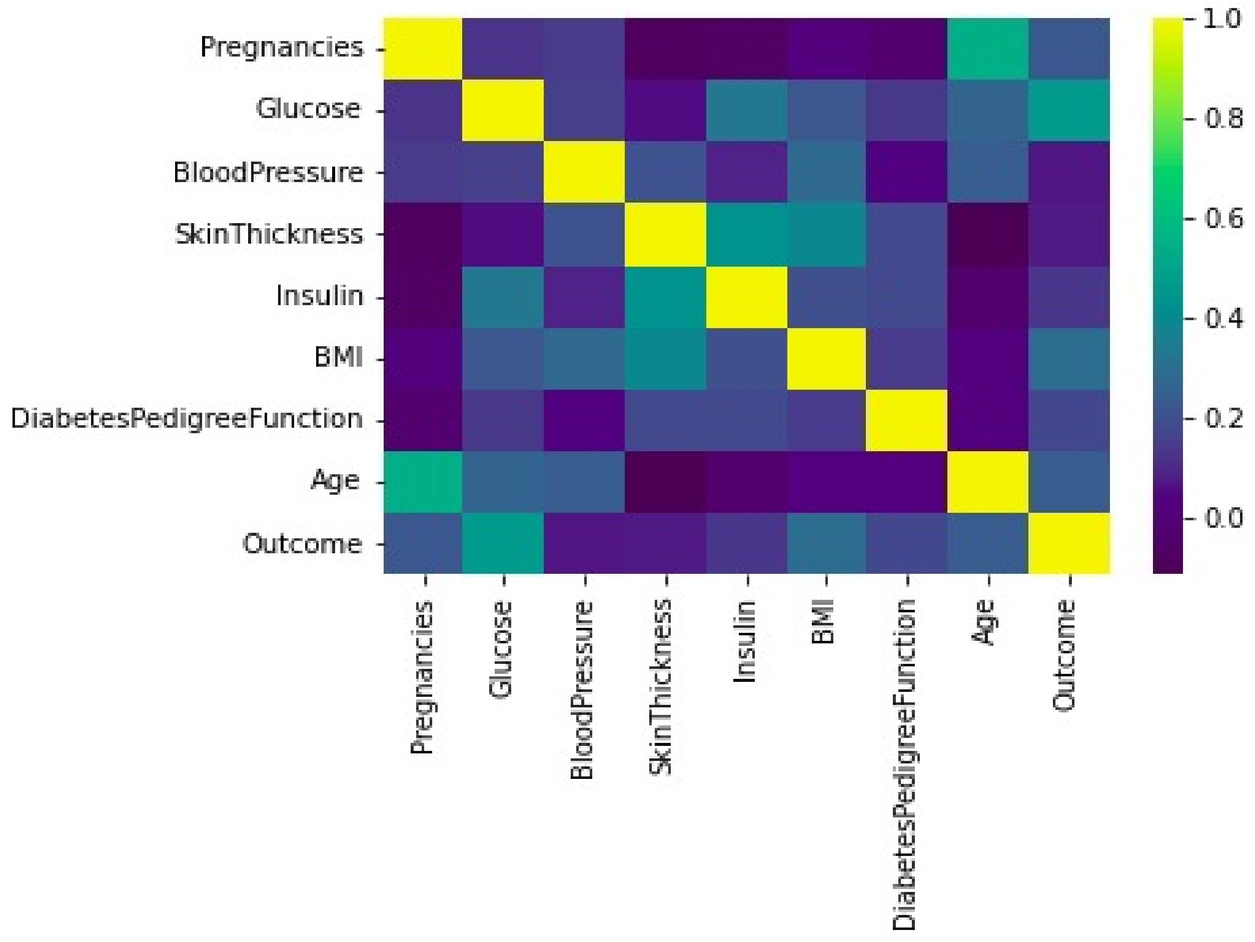


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068

The last column shows the correlation of variables with the **Outcome**, note that **Glucose** and **BMI** are the most correlated variables (apart from outcome itself), this confirms our hypothesis and the inferences made by the trees.



In fact we can extend this and compare the variables appearing in the top levels of the decision tree and their correlations with Outcome.



Note the brightly coloured cells in the last column all appear in the upper levels of our decision tree.

This tells us that for a patient to be diabetic, the best variables to look at are Glucose, BMI, Age and Pregnancies.