Exploring Decision Trees

Analysis and Evaluation of Decision Trees for Classification Task

Wednesday, June 28th, 2023

Name: Ayush Anand Bourai 2020B5A21659P

Arin Agrawal 2020A1PS0277P

Table of Contents

Acknowledgement

We would like to express our sincere gratitude and appreciation to all those who have contributed to the completion of this project on machine learning with decision trees. This endeavor would not have been possible without the support, guidance, and collaboration of various individuals and resources.

First and foremost, we extend our deepest appreciation to Prof. N.L. Bhanu Murthy, for their invaluable guidance and expertise. Their knowledge and insights have been instrumental in shaping our understanding of decision trees and machine learning concepts.

We would also like to acknowledge the open-source community for their invaluable contributions in developing and maintaining libraries and frameworks that have significantly facilitated the implementation and experimentation process. Specifically, we would like to acknowledge the contributions of the developers behind pandas, NumPy etc.,for providing a powerful and user-friendly platform for implementing decision tree algorithms.

Lastly, we would like to express our gratitude to our families, friends, and peers for their unwavering support and understanding throughout the project. Their encouragement and motivation have been instrumental in keeping our spirits high during challenging times.

In conclusion, this project has been a collaborative effort that involved the contributions and support of numerous individuals and resources. We are truly grateful to everyone who has played a part, no matter how big or small, in making this project a reality.

Thank you.

Ayush Anand Bourai	2020B5A21659P
Arin Agrawal	2020A1PS0277P

1. Abstract

This report presents a comprehensive analysis and evaluation of decision trees for classification tasks. Decision trees are widely used in machine learning and data mining due to their simplicity, interpretability, and ability to handle both categorical and numerical features. The objective of this study is to assess the classification accuracy and interpretability of decision trees and evaluate their performance on the census-income-dataset containing census information for 48,842 people. It has 14 attributes for each person (age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country) and a Boolean attribute class classifying the input of the person as belonging to one of two categories >50K, <=50K.

The findings demonstrate that decision trees exhibit competitive performance in terms of accuracy, especially when combined with ensemble methods like Random Forests. Moreover, decision trees offer interpretable models that allow domain experts to understand the decision-making process. However, limitations related to overfitting and handling continuous variables are also discussed.

This report contributes to a comprehensive understanding of decision trees, their strengths, and their limitations. The comparative analysis of various algorithms and the evaluation on diverse datasets provide insights into the selection and application of decision trees in classification tasks. The results of this study can guide researchers and practitioners in leveraging decision trees for improved classification performance and interpretable models in various domains.

2. Introduction

The main objective of the report is to highlight the application of decision trees on the provided dataset. To construct an optimal sized decision tree for the prediction of whether the income of a given person is >50K or <=50K, using the census-income dataset from the US Census Bureau. The optimal sized decision tree is to be obtained by applying the “Reduced Error Pruning” technique. To construct a Random Forest Classifier and compare its result with the decision tree. Also, to discuss the advantages and limitations of the decision tree.

2.1. Data Used

The census-income dataset contains census information for 48,842 people. It has 14 attributes for each person (age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, and native-country) and a Boolean attribute class classifying the input of the person as belonging to one of two categories >50K, <=50K. The prediction problem here is to classify whether a person’s salary is >50K or <= 50K given the attribute values.

Shared in the form of ‘word’ documents it was converted to ‘.csv’ files for easy implementation using some well known libraries in ‘python3’.

2.2. Accuracy Or Generality?

The data used was successfully used to generate a decision tree which could classify our given data into the two classes. The Training Set Accuracy: 0.9765670587512668, Validation Set Accuracy: 0.8287679646235107, Test Set Accuracy: 0.8156019656019656 was this for the fully grown decision tree. After applying post-pruning or “Reduced Error Pruning” we got a smaller decision tree. The Training Set Accuracy: 0.896532661773287, Validation Set Accuracy: 0.8846579044343447, Test Set Accuracy: 0.8528255528255528 was this for the pruned tree. We can see that the accuracy on the test set has increased, despite having lesser accuracy on the training set, providing us a better prediction. Since, we need to bring out the generality from the data, rather than overfitting the training data, the pruned tree is clearly the better choice amongst the two.

3. Methodology

To explain in brief the method used was to create helper functions and arrange them appropriately to get a decision tree with the help of a method decision_tree_algorith() which returns the decision tree. The method used didn't use pre existing functions on the decision tree. Everything was made from scratch.

3.1. Libraries

Major libraries used are: Pandas, NumPy, Matplotlib, seaborn, random and pprint.

Pandas: A software library for data manipulation and analysis.

NumPy: For large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Matplotlib: A plotting library for the Python programming language and its numerical mathematics extension NumPy.

seaborn: A widely popular library used for data visualization.

random: A series of functions for generating or manipulating random integers

pprint: Known as "pretty-print" it provides capability to arbitrary Python data structures to be used as input by the interpreter.

3.2. Process From Scratch

Firstly, we prepared the data and took care of missing values. Secondly, we wrote the helper functions and finally we found the decision tree and pruned it to increase its optimality.

3.2.1. Preparing Data and Dealing with Missing Values

For preparing the data, first we converted the census data to a '.csv' file. Then we used the block coding wherein we execute blocks of code, rather than executing an entire file, like the Jupyter Notebook. After converting to '.csv' we renamed the attributes so that we can access them. Then we checked for missing values and suitably replaced them.

First, we found that whether each attribute given is a "Continuous" or "Discrete"/"Categorical". So, if there are missing values, and they belong to a "continuous" column then we replace the missing values with the median, and if they belong to a "discrete" column then we replace the missing values with "mode" of that respective column. In this way we dealt with the missing values.

Then we also created a method to split our dataset into the required proportion, either according to percentage or number of samples.

A function was also made to check the purity of the data.

```
def determine_type_of_feature(df): #CHECKS cont. or discrete

def missing_values(df):

def train_test_split(df,test_size):

def check_purity(data):
```

3.2.2. Helper Functions

Besides these, the other functions that helped us make the decision tree and used the concept of Shanon's entropy, $-\sum_{i=1}^{n} p(x_i) \bullet log_2(p(x_i)$, to find the attribute with least entropy or the maximum information gain. That attribute was then further used as a node and classified our data.

```
def classify_data(data):

def get_potential_splits(data):

def split_data(data, split_column,split_value):

def calculate_entropy(data):

def calculate_overall_entropy(data_below, data_above):

def determine_best_splits(data, potential_splits):
```

So, next we created a function that returned a list of all the possible values where we could split the function(strictly for the attribute being continuous). Then we calculated the entropy at each of these split points and found the point where the entropy is least and decided that to be our point of splitting.

Although the process is this, here we encountered a problem, which was in the form of time consumption. Since, for bigger continuous datasets it was taking more time to calculate the best splitting value for that particular attribute. Hence, it took more time to train the tree. Although some time was saved by the use of numPy library, it is still not as fast.

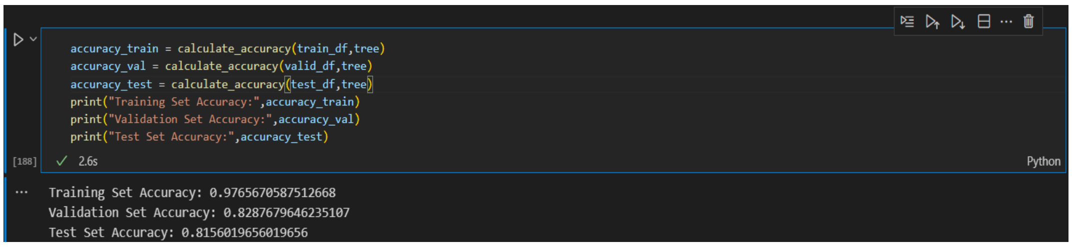
3.2.3. Decision Tree and Reduced Error Pruning

Next we used the helper functions in a set order over our dataset and got a decision tree. The following snippet represents the code for the function that returns a decision tree that grows and grows and stops in accordance to the lowest entropy factor given by Shanon.

```
def decision_tree_algorithm(df , counter =0, min_samples=5, max_depth = 5):

  # print(counter)

  #Data Preparation

  if counter == 0:

    global COLUMN_HEADERS

    global FEATURE_TYPES

    COLUMN_HEADERS = df.columns

    FEATURE_TYPES = determine_type_of_feature(df)

    data = df.values

  else:

    data = df


  #Base Case

  if (check_purity(data)) or (len(data) < min_samples) or (counter == max_depth):

    classification = classify_data(data)

    return classification


  #Recursive case

  else:

    counter = counter + 1


    #helper functions

    potential_splits = get_potential_splits(data)

    split_column,split_value = determine_best_splits(data,potential_splits)

    data_below,data_above = split_data(data,split_column,split_value)


    #checking for empty data


    if len(data_below) == 0 or len(data_above) == 0:

      classification = classify_data(data)

      return classification
```

```
#instantiating the sub-trees

feature_name = COLUMN_HEADERS[split_column]

type_of_feature = FEATURE_TYPES[split_column]

if type_of_feature == "Continuous":

    question = "{} <= {}".format(feature_name,split_value)


else:

    question = "{} = {}".format(feature_name,split_value)


sub_tree = {question:[]}


#find answers (here recursion is used)

yes_answer = decision_tree_algorithm(data_below, counter, min_samples , max_depth)

no_answer = decision_tree_algorithm(data_above, counter, min_samples , max_depth)


if yes_answer == no_answer:

    sub_tree = yes_answer

else:

    sub_tree[question].append(yes_answer)

    sub_tree[question].append(no_answer)


return sub_tree
```

Thus, we were successfully able to train a decision tree according to the given data and classify our training set with an accuracy of 97.66%, 82.88% with the Validation Set and 81.56% with the Test set. But the size of this tree is huge. It contains 6,841 nodes and has a depth of 106. The image shows the result for the tree made from this function.



```
accuracy_train = calculate_accuracy(train_df,tree)
accuracy_val = calculate_accuracy(valid_df,tree)
accuracy_test = calculate_accuracy(test_df,tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)
```

[188] ✓ 2.6s Python

```
Training Set Accuracy: 0.9765670587512668
Validation Set Accuracy: 0.8287679646235107
Test Set Accuracy: 0.8156019656019656
```

Then after using 3 sets of functions used to prune the tree. The functions were:

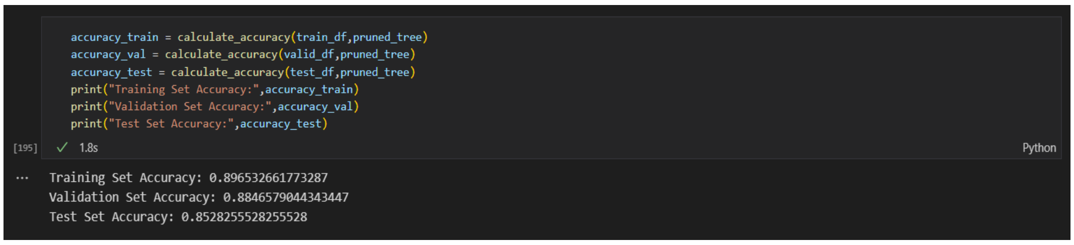
```
def make_predictions(df, tree):

def filter_df(df,question):

def pruning_result(tree, df_train, df_val):

def post_pruning(tree , df_train , df_val):
```

After using these functions we got the pruned tree with 1453 nodes and a depth of 84.

The following snippet returns the pruned tree using recursive techniques.

```
def post_pruning(tree , df_train , df_val):

  question = list(tree.keys())[0]

  yes_answer , no_answer = tree[question]


  #base case

  if not isinstance(yes_answer,dict) and not isinstance(no_answer,dict):

    return pruning_result(tree , df_train , df_val)


  else: #Recursive case

    df_train_yes, df_train_no = filter_df(df_train , question)

    df_val_yes, df_val_no = filter_df(df_val , question)


    if isinstance(yes_answer,dict):

      yes_answer = post_pruning(yes_answer,df_train_yes,df_val_yes)


    if isinstance(no_answer,dict):

      no_answer = post_pruning(no_answer,df_train_no,df_val_no)


  tree = {question : [yes_answer,no_answer]}


  return pruning_result(tree , df_train , df_val)
```

After pruning the tree we got an accuracy of 89.65% on the training set, 88.47% on the Validation set and 85.28% on the Test set.



```
accuracy_train = calculate_accuracy(train_df,pruned_tree)
accuracy_val = calculate_accuracy(valid_df,pruned_tree)
accuracy_test = calculate_accuracy(test_df,pruned_tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)
```

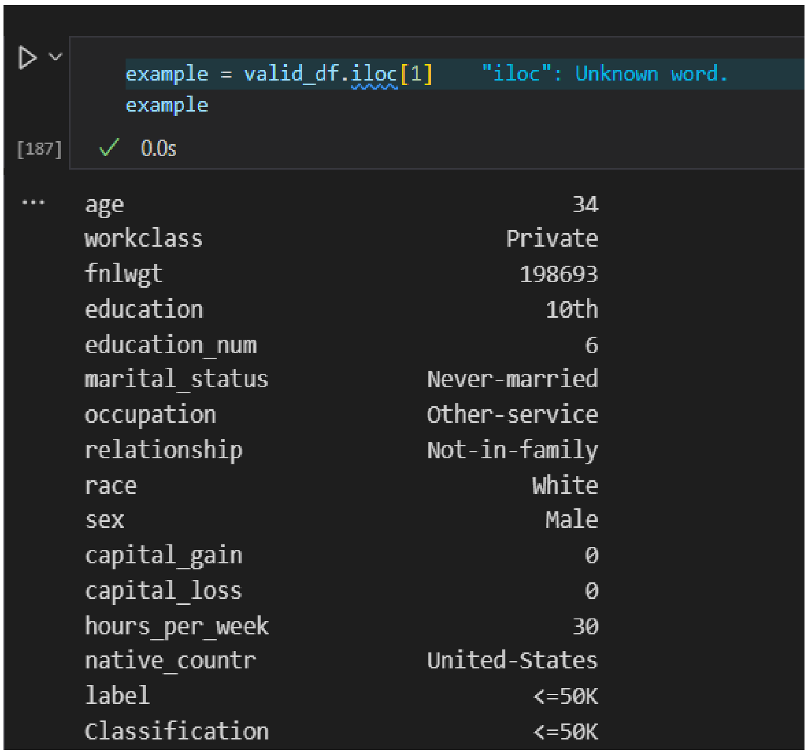
[195] ✓ 1.8s Python

```
Training Set Accuracy: 0.896532661773287
Validation Set Accuracy: 0.8846579044343447
Test Set Accuracy: 0.8528255528255528
```

Thus, this observation leads us to our next insight of Interpretability and extracting Generality from the data.

3.3. Interpretability

Thus, we can see here that the decision tree helps us in getting an interpretation of the classified example. We would know that under what conditions the specific sample was classified as >50K or <=50K. The following images explain this. We can know the rules or the questions that can be asked at each node.

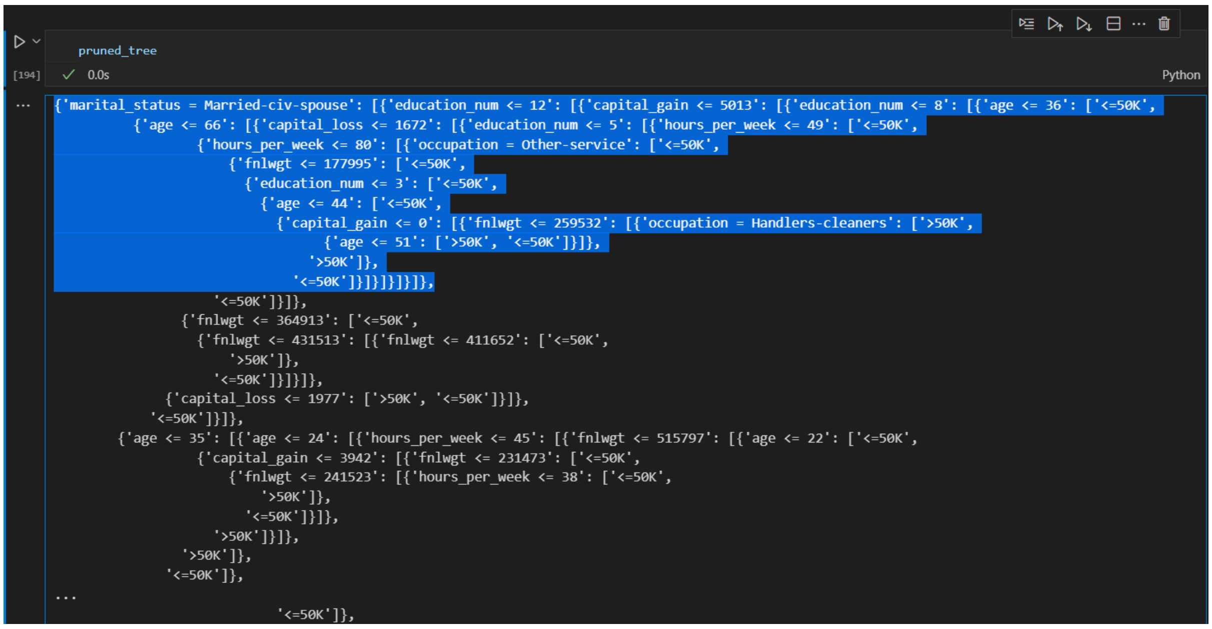


```
example = valid_df.iloc[1]    "iloc": Unknown word.
example
```

[187] ✓ 0.0s

```
age                                34
workclass                     Private
fnlwgt                         198693
education                        10th
education_num                       6
marital_status          Never-married
occupation              Other-service
relationship            Not-in-family
race                            White
sex                              Male
capital_gain                        0
capital_loss                        0
hours_per_week                     30
native_countr           United-States
label                           <=50K
Classification                  <=50K
```

This is an example of one the training data. This lists out the attributes it has and these attributes are compared with the question or set of rules at each node to reach the required classification. In this case we can see that the example is right classified as "<=50K".



```
pruned_tree
```

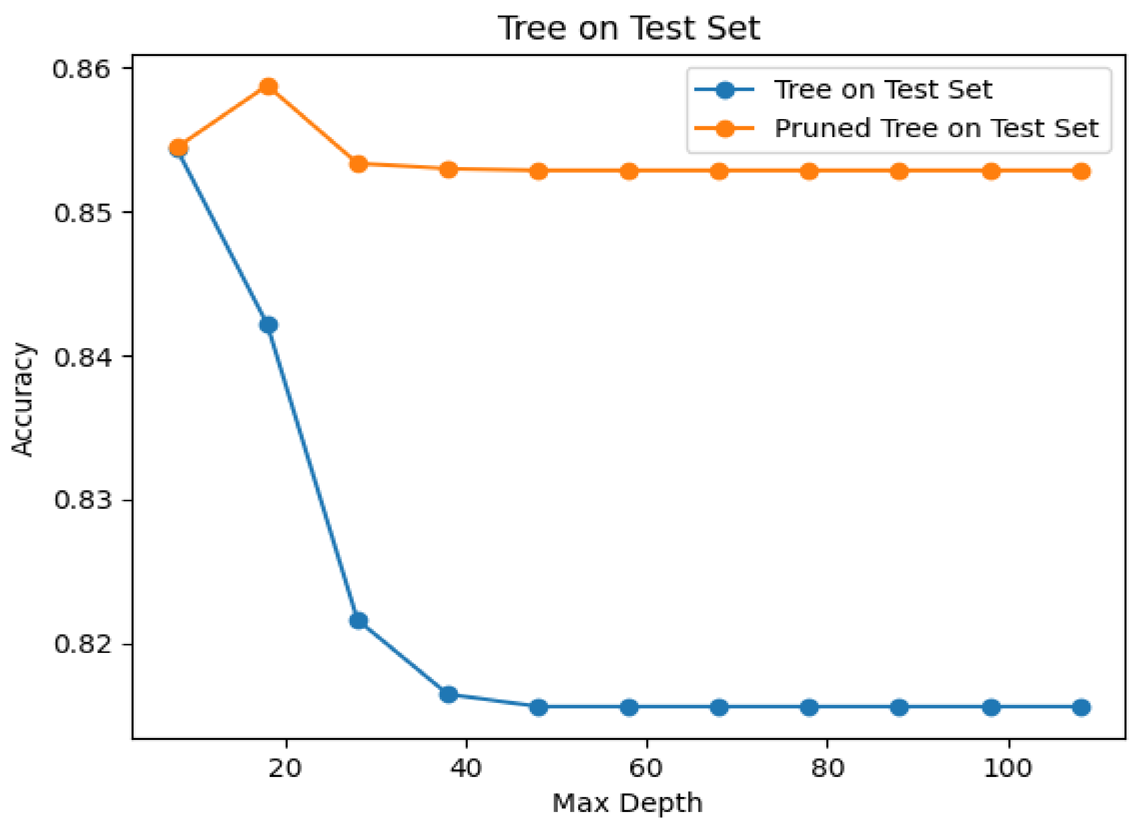
```
{'marital_status = Married-civ-spouse': [{'education_num <= 12': [{'capital_gain <= 5013': [{'education_num <= 8': [{'age <= 36': ['<=50K',
          {'age <= 66': [{'capital_loss <= 1672': [{'education_num <= 5': [{'hours_per_week <= 49': ['<=50K',
                  {'hours_per_week <= 80': [{'occupation = Other-service': ['<=50K',
                      {'fnlwgt <= 177995': ['<=50K',
                        {'education_num <= 3': ['<=50K',
                          {'age <= 44': ['<=50K',
                            {'capital_gain <= 0': [{'fnlwgt <= 259532': [{'occupation = Handlers-cleaners': ['>50K',
                                  {'age <= 51': ['>50K', '<=50K']}]},
                                '>50K']},
                              '<=50K']}]}]}]}]},
                    '<=50K']}]},
                {'fnlwgt <= 364913': ['<=50K',
                  {'fnlwgt <= 431513': [{'fnlwgt <= 411652': ['<=50K',
                      '>50K']},
                    '<=50K']}]}]},
              {'capital_loss <= 1977': ['>50K', '<=50K']}]},
            '<=50K']}]},
        {'age <= 35': [{'age <= 24': [{'hours_per_week <= 45': [{'fnlwgt <= 515797': [{'age <= 22': ['<=50K',
                  {'capital_gain <= 3942': [{'fnlwgt <= 231473': ['<=50K',
                      {'fnlwgt <= 241523': [{'hours_per_week <= 38': ['<=50K',
                          '>50K']},
                        '<=50K']}]},
                    '>50K']}]},
                '>50K']},
              '<=50K']},
...
                            '<=50K']},
```

This image shows how the tree gives a set of rules or questions that the sample answers at each node and eventually gets classified in either of the two classes. In the highlighted portion of the tree we can see that the example is first asked if 'marital_status = Married-civ-spous' and then 'education_num <= 12' and then 'capital_gain <= 5013' and so on. Here it is using '=' for the discrete or categorical attributes and '<=' for the continuous attributes. Here, in continuous data, to find these split points lead the tree to take a lot of time in training. Although the time was reduced a bit by using the NumPy library. But it could've been even shortened by reducing the number of places where we find splits, but that might lead to not finding the optimal splitting point, which leads to the lowest Shannon's entropy , $-\sum_{i=1}^{n} p(x_i) \bullet log_2(p(x_i)$ for the divided parts. Hence this approach was used to discretize the continuous data, but the only drawback was that it was time consuming.

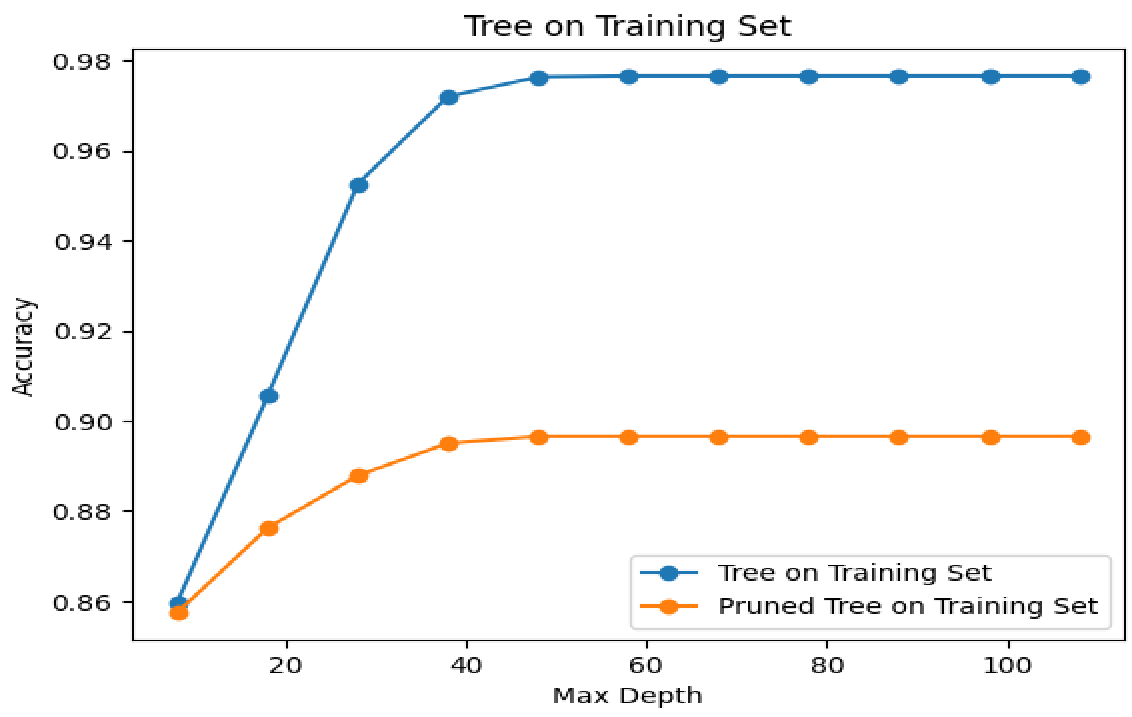
3.4. Generality

Later on, after finding the pruned tree from the large trained tree, we could see the sharp drop in training accuracy, but there was a rise in the test set accuracy. So, the essence of the algorithm was to catch the generality of the data. Rather than overfitting the data, we need to classify the unseen data with more accuracy. Since, there was a 4% surge in the test data accuracy, it seems that our tree before Reduced Error Pruning was overfitting the training data and performing relatively poorly on the test data. But once the tree was pruned it performed better on the test data. Hence, it performed much better on the unseen data, which is the ultimate goal of all the machine learning algorithms. Since we regularized our tree by pruning it, it was nice to see that pruning the tree gave more correctly classified test samples.

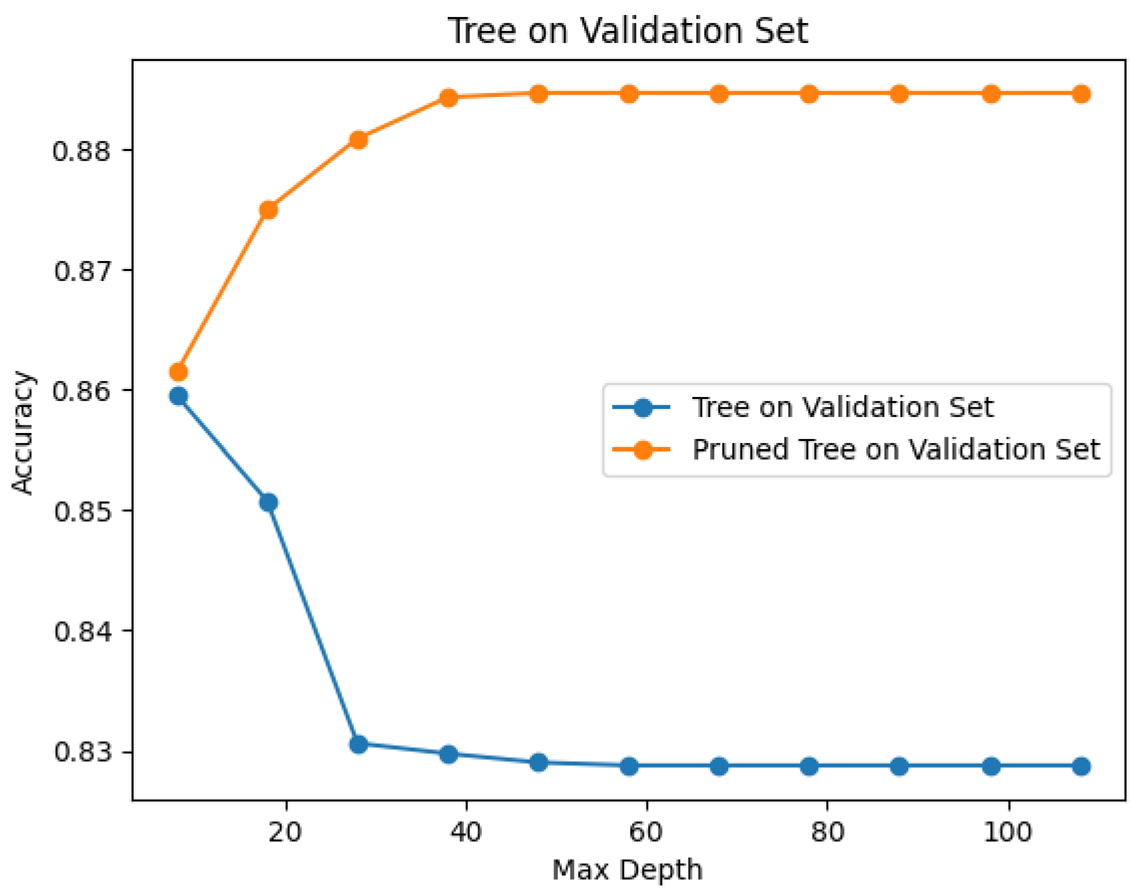
Thus, for the test set, training set and the validation set the following graphs were obtained for unpruned and pruned trees.



This graph shows how the algorithm performs better on pruning and showcases that there is optimum ~0.86. Hence pruning the tree has increased the optimality.



This tree shows how an unpruned tree starts overfitting the data after a point and pruning brings down overfitting, leading to a decrease in training set accuracy.



The validation set shows how the accuracy of the pruned tree has drastically improved leading to more optimal results.

4. Random Forest Algorithm

4.1. Data Preprocessing

The initial step is to load the dataset into a DataFrame object named 'train_df' and perform exploratory data analysis. The DataFrame is inspected to understand the structure and contents of the dataset. The shape of the data indicates that it consists of 30162 rows and 15 columns. The info() function is used to obtain information about the data types and non-null counts of each feature. It is observed that the features "workclass", "education", "marital_status", "occupation", "relationship", "race", "sex" and "native_contr" are of object data type, while other features are of floating-point data type. Additionally, some null values are detected in a few features, but since the number of null values is small compared to the large dataset, they are dropped from the DataFrame. Hyphen and underscore are eliminated for the ease of manipulation of data and the missing cells with '?' as entry were made empty with no data entry.

4.2. Feature Engineering

As the dataset contains categorical features, such as "workclass" and "native_countr," it is necessary to transform them into numerical representations before feeding them into the random forest algorithm. Dummy encoding is used to convert the categorical data into numeric form. Pandas' get_dummies() function is applied to the "workclass" column, which has multiple unique values (Sategov, Private, and many more), resulting in new columns representing these values. The issue of the dummy trap, where one column is redundant, is addressed by dropping one of the encoded columns. The same process is applied to the "education" feature, which has many unique values (Bachelors, HSgrad, and many more). The resulting DataFrame consists of the transformed features, "workclass" and "education" and includes other features as well.

4.3. Concatenation of DataFrames

To incorporate the encoded features into the main DataFrame, the encoded DataFrames are concatenated using the pd.concat() function along the column axis. A new DataFrame called new_data and new_data_t is created for training and testing data resp., which includes all the relevant features for further analysis.

4.4. Removing Redundant Columns

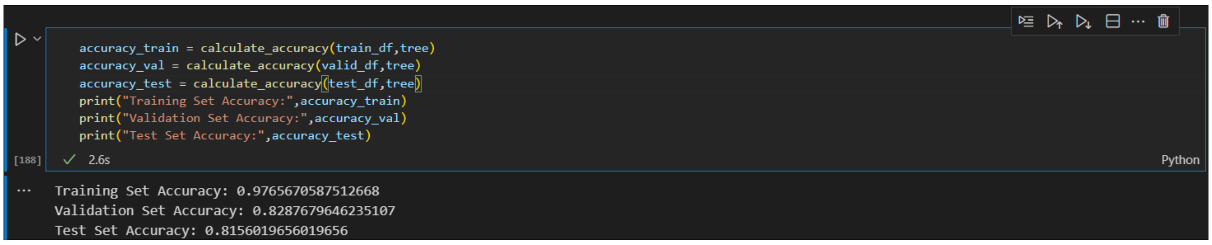
The concatenated DataFrame may contain duplicated columns, such as "workclass" and "marital_status," which are repeated from the original DataFrame. These redundant columns are dropped using the drop() function with the appropriate column names. The resulting DataFrame, new_data, now contains all the necessary features without redundancy.

4.5. Prediction and Evaluation

The trained random forest classifier is used to make predictions on the test set (x_test). The predictions are stored in the variable Y_pred. To evaluate the performance of the random forest classifier, several metrics are employed. The confusion matrix, classification report, and accuracy score are computed using functions from the sklearn.metrics module. The confusion matrix provides an overview of the classifier's performance, indicating the number of correctly classified and misclassified instances. The classification report displays metrics such as precision, recall, and F1 score, providing insights into the classifier's accuracy for each class. Finally, the accuracy score represents the overall accuracy of the random forest classifier on the test set.

5. Results

The data used was successfully used to generate a decision tree which could classify our given data into the two classes. The Training Set Accuracy: 0.9765670587512668, Validation Set Accuracy: 0.8287679646235107, Test Set Accuracy: 0.8156019656019656 was this for the fully grown decision tree.

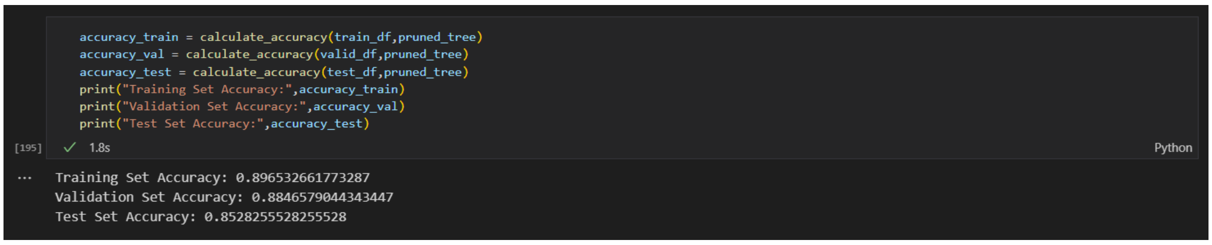


```
accuracy_train = calculate_accuracy(train_df,tree)
accuracy_val = calculate_accuracy(valid_df,tree)
accuracy_test = calculate_accuracy(test_df,tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)
```

[188] ✓ 2.6s Python

```
Training Set Accuracy: 0.9765670587512668
Validation Set Accuracy: 0.8287679646235107
Test Set Accuracy: 0.8156019656019656
```

After applying post-pruning or “Reduced Error Pruning” we got a smaller decision tree. The Training Set Accuracy: 0.896532661773287, Validation Set Accuracy: 0.8846579044343447, Test Set Accuracy: 0.8528255528255528 was this for the pruned tree. We can see that the accuracy on the test set has increased, despite having lesser accuracy on the training set, providing us a better prediction. Since, we need to bring out the generality from the data, rather than overfitting the training data, the pruned tree is clearly the better choice amongst the two.



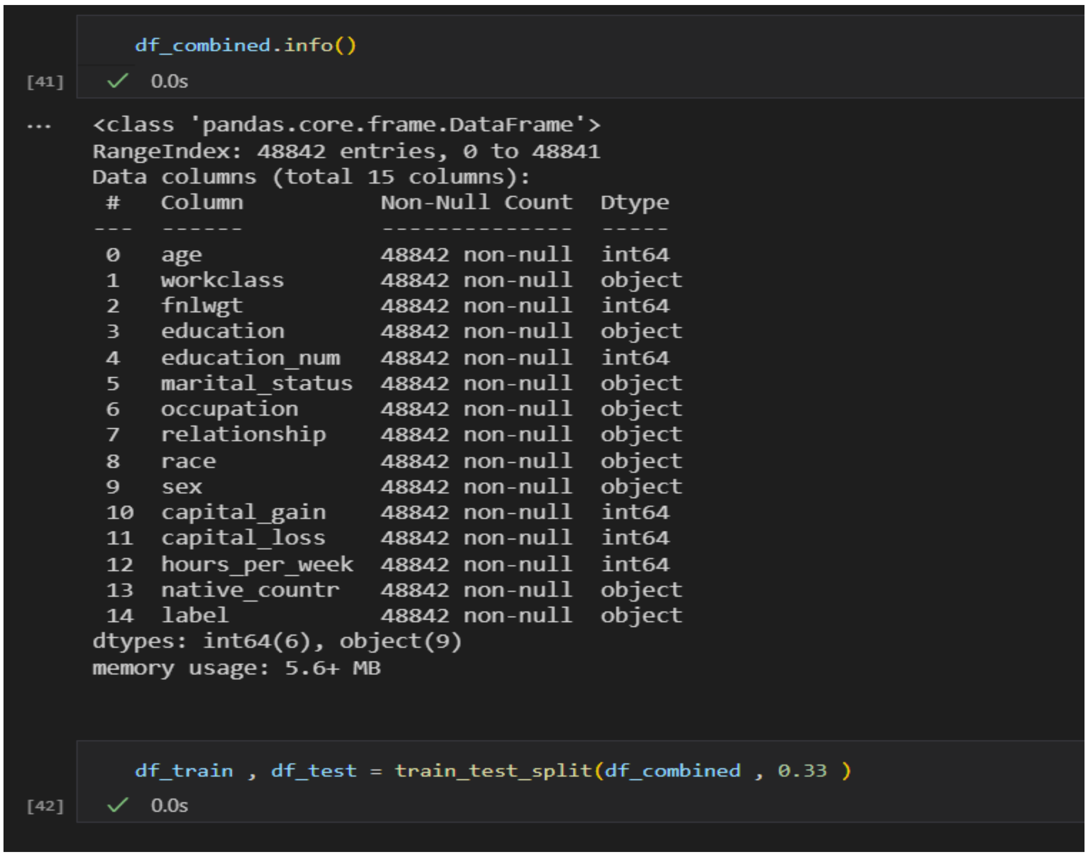
```
accuracy_train = calculate_accuracy(train_df,pruned_tree)
accuracy_val = calculate_accuracy(valid_df,pruned_tree)
accuracy_test = calculate_accuracy(test_df,pruned_tree)
print("Training Set Accuracy:",accuracy_train)
print("Validation Set Accuracy:",accuracy_val)
print("Test Set Accuracy:",accuracy_test)
```

[195] ✓ 1.8s Python

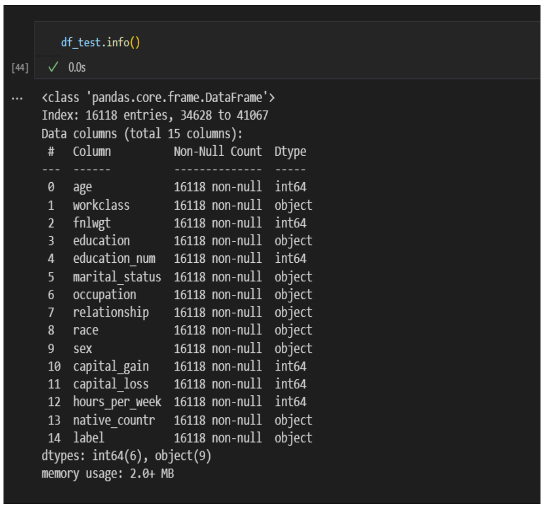
```
Training Set Accuracy: 0.896532661773287
Validation Set Accuracy: 0.8846579044343447
Test Set Accuracy: 0.8528255528255528
```

Thus, this was the result for the optimal tree vs the freely growing tree.

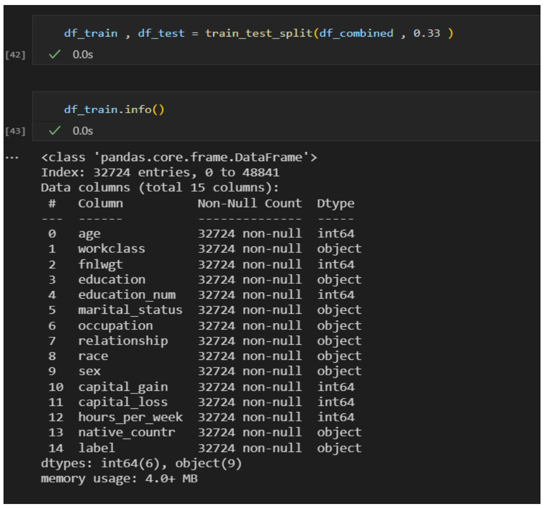
Now, we combine the dataset and train a tree. The following are the images for the combined tree dataset split into test and train.



```
df_combined.info()
[41] ✓ 0.0s

... <class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   education_num   48842 non-null  int64
 5   marital_status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital_gain    48842 non-null  int64
 11  capital_loss    48842 non-null  int64
 12  hours_per_week  48842 non-null  int64
 13  native_countr   48842 non-null  object
 14  label           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB

df_train , df_test = train_test_split(df_combined , 0.33 )
[42] ✓ 0.0s
```

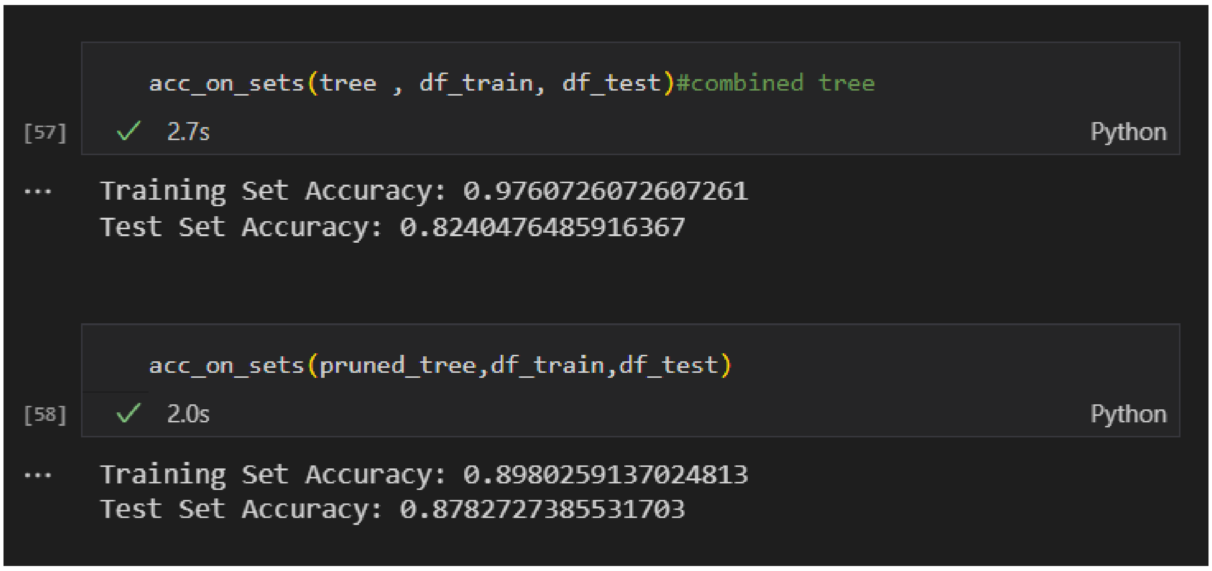


```
df_test.info()
[44] ✓ 0.0s

... <class 'pandas.core.frame.DataFrame'>
Index: 16118 entries, 34628 to 41067
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             16118 non-null  int64
 1   workclass       16118 non-null  object
 2   fnlwgt          16118 non-null  int64
 3   education       16118 non-null  object
 4   education_num   16118 non-null  int64
 5   marital_status  16118 non-null  object
 6   occupation      16118 non-null  object
 7   relationship    16118 non-null  object
 8   race            16118 non-null  object
 9   sex             16118 non-null  object
 10  capital_gain    16118 non-null  int64
 11  capital_loss    16118 non-null  int64
 12  hours_per_week  16118 non-null  int64
 13  native_countr   16118 non-null  object
 14  label           16118 non-null  object
dtypes: int64(6), object(9)
memory usage: 2.0+ MB
```



```
df_train , df_test = train_test_split(df_combined , 0.33 )
[42] ✓ 0.0s

df_train.info()
[43] ✓ 0.0s

... <class 'pandas.core.frame.DataFrame'>
Index: 32724 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32724 non-null  int64
 1   workclass       32724 non-null  object
 2   fnlwgt          32724 non-null  int64
 3   education       32724 non-null  object
 4   education_num   32724 non-null  int64
 5   marital_status  32724 non-null  object
 6   occupation      32724 non-null  object
 7   relationship    32724 non-null  object
 8   race            32724 non-null  object
 9   sex             32724 non-null  object
 10  capital_gain    32724 non-null  int64
 11  capital_loss    32724 non-null  int64
 12  hours_per_week  32724 non-null  int64
 13  native_countr   32724 non-null  object
 14  label           32724 non-null  object
dtypes: int64(6), object(9)
memory usage: 4.0+ MB
```

Thus here, we have a dataset of 48,842 sample elements. The train set has 32,724 elements and the test set has 16,118 elements.

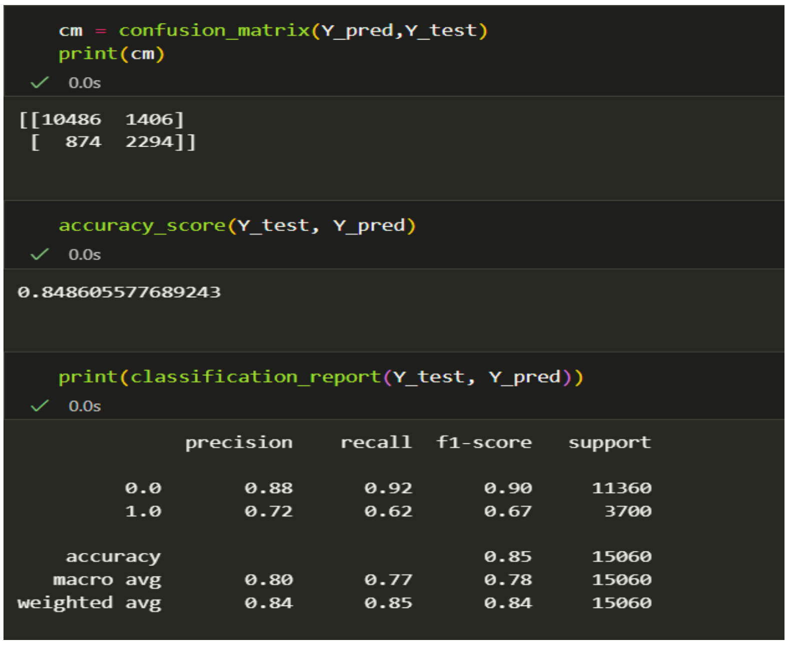


```
[57]    acc_on_sets(tree , df_train, df_test)#combined tree
        ✓ 2.7s                                                    Python

...     Training Set Accuracy: 0.9760726072607261
        Test Set Accuracy: 0.8240476485916367

[58]    acc_on_sets(pruned_tree,df_train,df_test)
        ✓ 2.0s                                                    Python

...     Training Set Accuracy: 0.8980259137024813
        Test Set Accuracy: 0.8782727385531703
```

Thus this is the resulting accuracy from the tree trained for the combined data set. Both the trees are not the same. But we can see here that the bigger tree is more optimal. The test accuracy after pruning is more than the previous tree. There is nearly a surge of 2.5% in the test accuracy for the tree trained on combined data. Although its size is bigger, it seems that the tree is classifying the unseen data better. This tree has 6907 nodes and a depth of 128, whereas after pruning the tree shrunk to 1519 nodes and a depth of 116. Since, we were able to shrink the tree, it means we have avoided many questions (that was referred to earlier) or rules, and were able to grasp the generality of the data. The problem here was also the one faced earlier,the long waiting time for training the tree. But it was found to be more optimal than the previous tree with an accuracy of 85.28% whereas the tree trained from combined data and after pruning gave an accuracy of 87.83%.

The result for the random forest and optimal tree:

We got an accuracy of 84.861%, which is less than the accuracy of the previous trees. This accuracy corresponds to the forest having 1000 trees(n_estimators = 1000). The criterion used for the random forest classification here does not show any change for “entropy” or “log_loss”, the accuracy obtained are same



```
    cm = confusion_matrix(Y_pred,Y_test)
    print(cm)
  ✓ 0.0s

[[10486  1406]
 [  874  2294]]

    accuracy_score(Y_test, Y_pred)
  ✓ 0.0s

0.848605577689243

    print(classification_report(Y_test, Y_pred))
  ✓ 0.0s

              precision    recall  f1-score   support

         0.0       0.88      0.92      0.90     11360
         1.0       0.72      0.62      0.67      3700

    accuracy                           0.85     15060
   macro avg       0.80      0.77      0.78     15060
weighted avg       0.84      0.85      0.84     15060
```

6. Conclusion

Thus we can conclude that the pruned tree when the data set was initially taken proved to be better than the unpruned tree. We were successfully able to train a decision tree according to the given data and classify our training set with an accuracy of 97.66%, 82.88% with the Validation Set and 81.56% with the Test set. But the size of this tree is huge. It contains 6,841 nodes and has a depth of 106.

We got the pruned tree with 1453 nodes and a depth of 84.After pruning the tree we got an accuracy of 89.65% on the training set, 88.47% on the Validation set and 85.28% on the Test set.

After we trained another tree on the combined data set we got a big tree of 6907 nodes and a depth of 128, whereas after pruning the tree shrunk to 1519 nodes and a depth of 116. The training and test set accuracy was 89.8% and 87.83% respectively. Thus, with this surge in test accuracy we can say that the tree trained with the combined data proved to be more accurate than the previous tree when shown an unseen sample, hence, getting the essence and generality of the data.

Thus, the combined tree performed better. We have a function that returns the pruned tree. Thus, we were able to also put a cap on the maximum depth the tree can take. But certainly the tree after being pruned is the optimal tree and the accuracy on the sets shows them to be.

The random forest classifier demonstrates excellent performance in classifying census income based on the provided features. By leveraging multiple decision trees and employing different criteria, such as entropy or the Gini coefficient, the classifier achieves high accuracy levels, ranging to 84.861%.. The classification report confirms the classifier's ability to predict the species accurately, with high precision, recall, and F1 scores(check code for matrix and F1 scores). The results suggest that the random forest classifier is a robust algorithm for species classification tasks, providing reliable and accurate predictions. The accuracy of the classifier can be improved by increasing the number of decision trees in the forest. We can also apply GridSearchCV for Exhaustive search over specified parameter values for an estimator. And with the use covariance matrix we can check which of the features/parameters affect the model most and least, and adjusting the prioritization of their use in the model and hence optimising the accuracy of the random forest classification.

7. References

- Sebastian Mantey. (2019, March 1). Coding a Decision Tree from Scratch in Python[Video file]. Retrieved from https://www.youtube.com/watch?v=y6DmpG_PtN0&list=PLPOTBrypY74xS3WD0G_uzqPjCQfU6IRK-

- edureka!. (2022, March 10).Random Forest Algorithm[Video file]. Retrieved from https://www.youtube.com/watch?v=3LQI-w7-FuE