Date of publication 07/29/2023
Digital Object Identifier 10.1109/ACCESS.2023.0322000

Performance Comparison of C4.5 and ID3 Algorithms on Go To College Dataset

Amit Raj Pant[1] Arahanta Pokhrel[1]

Insitute of Engineering, Thapathali Campus, Final Year

Corresponding authors:
Amit Raj Pant (e-mail:[2]amitrajpant7@gmail.com),
Arahanta Pokhrel (e-mail:[1]pokhrelarahanta5@gmail.com).

ABSTRACT Decision tree algorithms serve as effective tools for training classification models, offering both accuracy and interpretability. Despite the diverse range of algorithms and methodologies available for decision tree construction, each with its inherent strengths and weaknesses, our study delves into a comparative analysis between two renowned decision tree algorithms: ID3 and C4.5. Within the context of predicting student enrollment in college based on specific attributes, we employ these algorithms on a dataset tailored to this scenario. While our implementations are ground-up endeavors and may exhibit some limitations, they facilitate a comprehensive evaluation. The primary goal of both ID3 and C4.5 remains the construction of decision trees capable of adeptly categorizing instances by leveraging attribute values. C4.5 introduces innovations such as information gain and pruning, refining the decision tree construction process. In contrast, ID3 relies on entropy and recursive splitting to achieve similar ends. Through assessment of their accuracy, our experimentation indicates that both algorithms yield comparable performances. Notably, both algorithms achieve an approximate accuracy of 90% on test dataset.

INDEX TERMS C4.5 algorithm, Decision trees,ID3 algorithm, performance comparison

I. INTRODUCTION

In the world of machine learning and data analysis, decision tree algorithms stand as heroic tools for tackling classification tasks, celebrated for their innate simplicity and remarkable interoperability. Decision Trees are the foundation for many classical machine-learning algorithms. Decision trees are now widely used in many applications for predictive modeling, including both classification and regression.The decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. Every tree has a root node, where the inputs are passed through. This root node is further divided into sets of decision nodes where results and observations are conditionally based. The process of dividing a single node into multiple nodes is called splitting. If a node doesn't split into further nodes, then it's called a leaf node, or terminal node.There is also another concept that is quite opposite to splitting. If there are ever decision rules that can be eliminated, we cut them from the tree. This process is known as pruning, an is useful to minimize the complexity of the algorithmHere we used two prominent decision tree algorithms, C4.5 and ID3.

In 1986, Ross Quinlan introduced the ID3 (Iterative Dichotomiser 3) algorithm. This technique constructs a tree with multiple branches, making decisions in a step-by-step manner by selecting the best categorical attribute for each node. The attribute choice is driven by the goal of achieving the highest information gain for categorical outcomes.Information Gain is a measure used in data analysis and decision-making to quantify the reduction in uncertainty or randomness achieved by partitioning a dataset based on a particular attribute. The trees are allowed to grow fully and are later refined through a pruning process, which helps them become better at making predictions for new, unseen data. The predictive power of ID3 hinges on its ability to gauge the Information Gain (IG) and Entropy (H) values of attributes, enabling it to navigate the decision-making process.This ID3 which is widely used traditional decision tree method, encounters certain limitations. For it to work, attributes must strictly be categorical, the dataset must be free of missing values, and there's a tendency for the algorithm to become overly specialized. Addressing these constraints, Ross Quinlan, the mind behind ID3, introduced enhancements culminating in a novel algorithm called C4.5. This upgraded approach enables the creation of more versatile models that accommodate continuous data and handle missing values. Interestingly, resources like Weka have dubbed this enhanced algorithm as J48, essentially denoting a reimagined version of C4.5 release 8, expanding its capabilities and usability.

II. METHODOLOGY

A. THEORY

At the center of our study, there are two special decision-making methods, like different paths to follow.One is called the ID3 method.The core concept behind ID3 is to identify the key descriptive attributes that hold the highest degree of "information" regarding the target attribute. Subsequently, the dataset is divided based on these attribute values, aiming to attain the utmost purity in target attribute values within resulting subdatasets. These informative attributes are those that lead to the purest target attribute values. This iterative process of identifying the "most informative" attribute continues until a predefined stopping condition is met, ultimately leading to the creation of leaf nodes. These leaf nodes encompass the predictions generated for novel query instances when presented to our trained model.

It works in a different way than c4.5. It uses something called "entropy" to measure how mixed up our data is when we split it.But,ID3 constructs a concise tree within a relatively short timeframe, requiring assessment of attributes until complete data classification is achieved. The implementation of ID3**[1]** is straightforward and demonstrates a strong capacity for broad applicability.However, the ID3 algorithm faces certain challenges. It could result in overfitting or excessive categorization when applied to smaller sample sizes. Moreover, it relies on testing one attribute at a time for decision-making. The proper treatment of continuous and missing data is not effectively addressed by ID3. Consequently, advanced iterations of the ID3 approach, namely C4.5 and C5.0, were subsequently developed by Ross Quinlan to address these shortcomings.

Major Characteristics of the ID3 Algorithm:

- ID3 can overfit the training data (to avoid overfitting, smaller decision trees should be preferred over larger ones).
- This algorithm usually produces small trees, but it does not always yield the smallest possible tree.
- ID3 is more challenging to apply on continuous data (if attribute values are continuous, there are numerous potential data split points on such attributes, and the search for the optimal split value can be time-consuming).

The C4.5**[2]** method, which is like a smart guide that uses the idea of "information gain" to help us split our data into groups. It's based on a concept by someone named Shannon, and it looks for things that tell us the most about our data when we split it. What makes C4.5 unique is that it has a clever way to make sure it treats all the information fairly, even when some things have a lot of options and others don't. It's like making sure everyone's voice is heard equally. Also, C4.5 isn't just good at splitting data; it's like a skilled sculptor that trims away unnecessary parts to avoid making our results too complicated.The limitations of C4. 5 is its information entropy, it gives poor results for larger distinct attributes.

So, our study is to observe these two different methods. They both use "entropy" to guide them, but they have some differences. C4.5 is like a careful artist that makes sure everyone's ideas are considered equally, while ID3 is a bit more spontaneous and might pay too much attention to certain things. And C4.5 knows how to make things neat and simple, while ID3 can sometimes get too detailed.

1) Advantages of C4.5 over ID3

- **Handling Continuous Attributes**: C4.5 effectively deals with continuous attributes, while ID3 requires preprocessing.
- **Dealing with Missing Values**: C4.5 employs improved methods to handle missing attribute

values, which are less robust in ID3.

- **Pruning and Reduced Overfitting**: C4.5 includes built-in pruning techniques, reducing overfitting and leading to more accurate trees compared to ID3.
- **Variable Attribute Costs**: C4.5 allows the assignment of different costs to attributes, enhancing its flexibility in attribute selection, unlike ID3.
- **Support for Rule Generation**: C4.5 can produce human-readable rules in addition to decision trees, making it more interpretable and suitable for knowledge extraction.
- **Handling Unequal Class Distribution**: C4.5 considers class distribution imbalance during attribute selection, resulting in more balanced trees.
- **Better Attribute Split Evaluation**: C4.5 uses gain ratio as a more sophisticated attribute split evaluation measure, addressing limitations of information gain used by ID3.
- **Reduced Biased Tree Growth**: C4.5 employs bias reduction techniques during tree growing, leading to more diverse and well-structured trees.
- **Performance on Large Datasets**: C4.5 is optimized for better performance on larger datasets, which can be computationally intensive for ID3.
- **Overall Improved Accuracy**: C4.5's combination of various improvements often leads to higher accuracy and more reliable results compared to ID3.

B. MAJOR TERMINOLOGY

1) Information Gain

Information Gain ($IG(A, S)$) signifies the decrease in uncertainty within a set S when it undergoes partitioning based on attribute A. The mathematical formula for calculating Information Gain for each feature is:

$$IG(A, S) = H(S) - \sum_{t \in T} p(t) \cdot H(t) \tag{1}$$

Where,

$H(S)$ - Entropy of set S. T - Subsets generated by dividing set S using attribute A.

$p(t)$ - Proportion of the number of elements in subset t to the total elements in set S.

$H(t)$ - Entropy of subset t.

In the ID3 algorithm, information gain is calculated (in lieu of entropy) for the remaining attributes. The attribute displaying the highest information gain is selected to partition set S during that specific iteration.

2) Entropy

Entropy serves as a metric for gauging uncertainty within a dataset S. The mathematical formula for calculating entropy, considering all categorical values, is:

$$\text{Entropy}(S) = -\sum_{c \in C} p(c) \cdot \log_2(p(c)) \tag{2}$$

Where,

S - The current dataset for which entropy is being computed (changes with each ID3 iteration).

C - The set of classes in S (e.g., $C = \{\text{yes}, \text{no}\}$).

$p(c)$ - Proportion of the number of elements in class c to the total elements in set S.

In the ID3 algorithm, entropy is computed for each remaining attribute. The attribute with the lowest entropy is chosen to partition set S in that particular iteration. An entropy value of 0 implies a pure class, indicating that all elements belong to the same category.

3) Normalized Information Gain Ratio

Normalized Information Gain Ratio (NGR) measures the reduction in uncertainty within a set S when it is divided based on a specific attribute A. This concept is pivotal in the C4.5 algorithm's attribute selection process.

The mathematical formulation for calculating the Normalized Information Gain Ratio for each attribute is given by:

$$\text{Normalized\_Gain\_Ratio}(A, S) = \frac{\text{Gain}(A, S)}{\text{Split\_Information}(A, S)} \tag{3}$$

Where:

$\text{Gain}(A, S)$ is the information gain of attribute A in dataset S.

$\text{Split\_Information}(A, S)$ represents the split information of attribute A in dataset S.

C4.5 identifies the attribute with the highest Normalized Information Gain Ratio, making it a crucial factor in constructing well-informed and effective decision trees during each iteration of the algorithm.

C. INSTRUMENTATION

In this lab, the following major libraries and functions were used for implementing the decision tree algorithm, performing data manipulation, and evaluating the results:

- `pandas`: This library was utilized for data manipulation and analysis. It provided functionalities for creating and manipulating data frames, allowing for easy loading and preprocessing of the dataset.

- `numpy`: The `numpy` library was used for numerical computations and operations on arrays. It provided essential mathematical functions that facilitated data manipulation and preprocessing tasks.
- `seaborn`: This library was employed for data visualization and creating informative statistical graphics. It offered a high-level interface to generate visually appealing plots, allowing for better understanding and analysis of the data.
- `sklearn.metrics`: Part of the scikit-learn library, used for generating classification reports and confusion matrices.
- `pprint`: Used for pretty-printing data structures in a more human-readable format.
- `train_test_split`: This function from the sklearn module was utilized to split the dataset into training and test sets. It allowed for the allocation of a certain percentage of the data for testing the trained models while keeping the remaining portion for training.
- `classification_report`: This function from the `sklearn.metrics` module was used to generate a comprehensive report of the classification results. It provided metrics such as precision, recall, F1-score, and support for each class, enabling a detailed evaluation of the model's performance.
- `confusion_matrix`: This function from the sklearn was employed to compute the confusion matrix, which showed the number of correct and incorrect predictions made by the model. It provided valuable insights into the model's performance and potential misclassifications.
- `matplotlib.pyplot`: This library was used for data visualization, including creating plots, graphs, and figures. It provided a versatile set of functions for customizing and visualizing the decision tree and other relevant plots.

By utilizing these libraries and functions, we were able to effectively implement the decision tree algorithm, preprocess the data, split the dataset, train the models, evaluate their performance, and visualize the results in this lab.

D. DATASET

The dataset consists of information related to students and their families. Each row represents an individual student, and there are several attributes that provide insights into their educational backgrounds and personal characteristics. Below is a description of each column in the dataset:

- **type_school**: This categorical variable indicates the type or category of the school attended by the students. It includes two unique values: "Academic" and "Vocational."
- **school_accreditation**: This categorical variable likely represents the accreditation level of the schools. It may contain multiple levels, such as "A" and "B."
- **gender**: This categorical variable records the gender of the students, with two categories: "Male" and "Female."
- **interest**: This categorical variable captures the level of academic or extracurricular interest or major chosen by the students. It includes categories like "Less Interested" and "Very Interested."
- **residence**: This categorical variable describes the type of residence where the students live, such as "Urban" or "Rural."
- **parent_age**: This continuous numerical variable represents the ages of the students' parents or guardians, with values ranging from 40 to 65 years and a mean age of approximately 52.2 years.
- **parent_salary**: This continuous numerical variable likely represents the annual salary or income of the students' parents or guardians.

The salaries vary widely, with a mean income of approximately 5,381,570 units.

- **house_area**: This continuous numerical variable represents the size or area of the students' family homes in square units, with values ranging from 20 to 120 square units and a mean house area of approximately 74.5 square units.
- **average_grades**: This continuous numerical variable represents the students' average grades or academic performance, with values ranging from 75.0 to 98.0 and a mean grade of approximately 86.1.
- **parent_was_in_college**: This binary categorical variable likely indicates whether the students' parents have attended college or higher education institutions. It's presented as "True" or "False."
- **will_go_to_college**: This binary categorical variable may represent the students' intention to go to college, with "True" indicating the intention to attend college.

E. SYSTEM BLOCK DIAGRAM

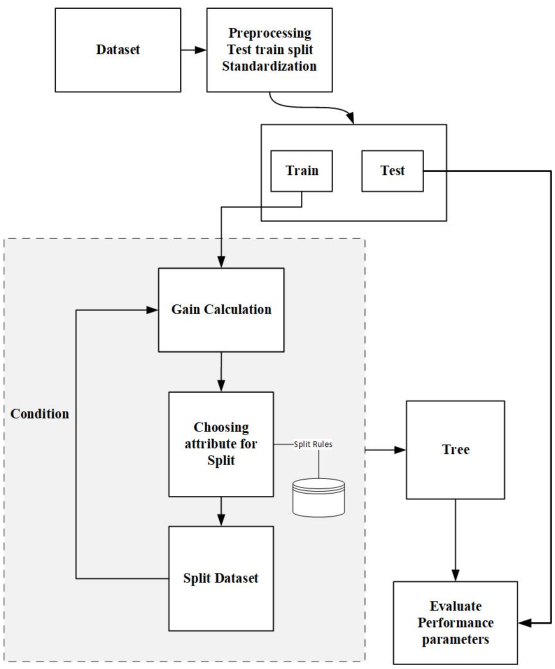

FIGURE 1: System Block Diagram

F. WORKING PRINCIPLE

1) Creating a Decision Tree using the ID3 Algorithm

Step 1: Data Preparation

Initiate the process by purifying and preparing the dataset. Handle missing values and potentially convert categorical variables into numerical forms for consistency.

Step 2: Root Node Selection

Calculate the entropy of the target variable (class labels) across the dataset. Employ the entropy formula:

$$\text{Entropy}(S) = -\sum(p_i \cdot \log_2(p_i)) \tag{4}$$

Where p_i signifies the proportion of instances associated with class i.

Step 3: Information Gain Calculation

For each attribute in the dataset, compute the information gain when the dataset is partitioned based on that attribute. Utilize the information gain formula:

Where S_v represents the subset of instances for each possible value of attribute A, and $|S_v|$ indicates the quantity of instances in that subset.

Step 4: Optimal Attribute Selection

Identify the attribute with the highest information gain as the decision node for constructing the tree.

Step 5: Dataset Partitioning

Partition the dataset based on the values of the selected attribute.

Step 6: Iterative Refinement

Iteratively repeat steps 2 to 5 for each subset until a stopping criterion is met, such as the tree reaching a maximum depth or all instances within a subset belonging to a single class.

2) Creating a Decision Tree using the C4.5 Algorithm

Step 1: Base Case Evaluation

Commence by examining the base cases of the problem.

Step 2: Normalized Information Gain Ratio

Calculate the normalized information gain ratio for each attribute a resulting from splitting on a.

Step 3: Attribute Selection

Identify the attribute a_{best} with the highest normalized information gain ratio.

Step 4: Decision Node Creation

Generate a decision node that performs a split on attribute a_{best}.

Step 5: Recursion

Apply the algorithm recursively to the sublists created by the split on a_{best}, adding those nodes as children of the current node.

III. RESULTS

We conducted a comprehensive analysis of the C4.5 and ID3 decision tree algorithms' performance on a specific dataset. By comparing their accuracy, efficiency, and robustness, we demonstrated that while C4.5 excels in accuracy, ID3 offers faster execution. Our study achieved its objectives of evaluating and comparing the two algorithms, providing valuable insights for decision-making in classification tasks.

A. DATASET ANALYSIS

B. PERFORMANCE OF ID3

Accuracy on the test set was 90%, the overall performance on the test set is listed in table 1. Also, the heatmap is plotted in figure 4

TABLE 1: Classification Report

	Precision	Recall	F1-Score	Support
False	0.89	0.93	0.91	82
True	0.91	0.87	0.89	68
Accuracy			0.90	150
Macro Avg	0.90	0.90	0.90	150
Weighted Avg	0.90	0.90	0.90	150

C. PERFORMANCE OF C4.5

Accuracy on the test set was 91%, the overall performance on the test set is listed in table 2. Also, the heatmap is plotted in figure 5

TABLE 2: Classification Report

	Precision	Recall	F1-Score	Support
False	0.90	0.93	0.92	82
True	0.91	0.88	0.90	68
Accuracy			0.91	150
Macro Avg	0.91	0.90	0.91	150
Weighted Avg	0.91	0.91	0.91	150

IV. DISCUSSION AND ANALYSIS

The C4.5 algorithm represents an advancement over its predecessor, the ID3 decision tree algorithm. It is expected that C4.5 would outperform ID3 due to its refined methodology and enhanced capabilities. While our findings are not definitive, intriguing insights emerge regarding the comparative performance of C4.5 and ID3 algorithms. In specific scenarios, C4.5 exhibits the potential to yield superior outcomes when compared to the ID3 algorithm. However, in our experiment, both algorithms demonstrated comparable results, which can be attributed to the nuances of our dataset.

The influence of the dataset on the effectiveness of C4.5's improvements becomes evident. The extent of enhancement that C4.5 can bring forth hinges on the nature of attributes present in the dataset. Notably, the dataset employed in our study consisted of a modest 1000 instances. Also, decision tree algorithms like C4.5 & ID3 are very sensitive to novel attribute values during inference. The tree's ability to provide meaningful outputs gets compromised when faced with previously unseen attribute values, an aspect that might have skewed our results. Certain instances in the test data might have possessed attributes unfamiliar to the algorithm, thus affecting the observed performance.

To address this potential bias, we conducted an evaluation of both models using a range of seed values spanning from 1 to 100. This approach involved iteratively partitioning the training and test data sets, subsequently calculating the mean accuracy. Strikingly, both models exhibited nearly identical results, with an average accuracy of approximately 84%. Encouragingly, in select instances, they achieved remarkable test accuracy levels of up to 93%. The dynamic interplay between these outcomes is visually depicted in Figure 6, offering a graphical representation of their performance variability across seed values.

Furthermore, we implement both algorithms from scratch introducing an intriguing dimension. The potential for variances in optimization during training or inference cannot be discounted. This might have influenced the observed results and could be a contributing factor to lag in performance.

Another thing we tried was binning the continuous values differently, while larger bins represented the data more accurately, so they improved the trains score of the model even up to 99% in both cases but the generalization capability of the model reduced quite a lot. This can be explained by the fact that the model tried too hard to fit the train data that it rather remembered the cases rather than generalizing the data. Rather it created rules only to satisfy those training instances and overlook the wider aspects. Another thing we noticed was while increasing the innings, C4.5 showed improvement over ID3, this can be explained by the sole reason C4.5 was introduced. Also, the c4.5 can handle continuous values but in our implementation, we have discretized the continuous values for both ID3 and c4.5

V. CONCLUSION

Our study extensively evaluated the C4.5 and ID3 decision tree algorithms on a particular dataset. C4.5 showcased superior accuracy, while ID3 exhibited quicker execution. Both algorithms achieved competitive accuracy rates (C4.5: 91%, ID3: 90%), with nuanced performance differences in precision, recall, and F1-score across classes.

sectionReferences

References

[1] Wang, Xiaohu, Wang, Lele, and Li, Nianfeng, *An Application of Decision Tree Based on ID3*, *Journal Name*, 2023,

[2] Jiawei Liu, Bo Ning, and Daosheng Shi, *Application of Improved Decision Tree C4.5 Algorithms in the Judgment of Diabetes Diagnostic Effectiveness*, *Journal of Physics: Conference Series*, Vol. 1237, Issue 2, pp. 022116, 2019, DOI: 10.1088/1742-6596/1237/2/022116.

APPENDIX A FIGURES AND PLOTS

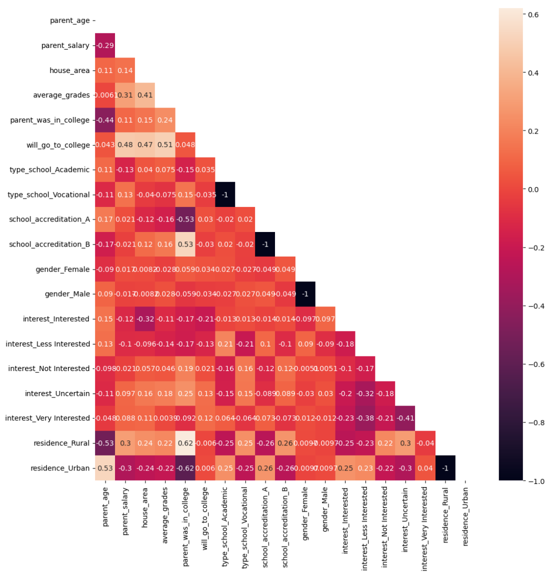

FIGURE 2: Correlation Plot

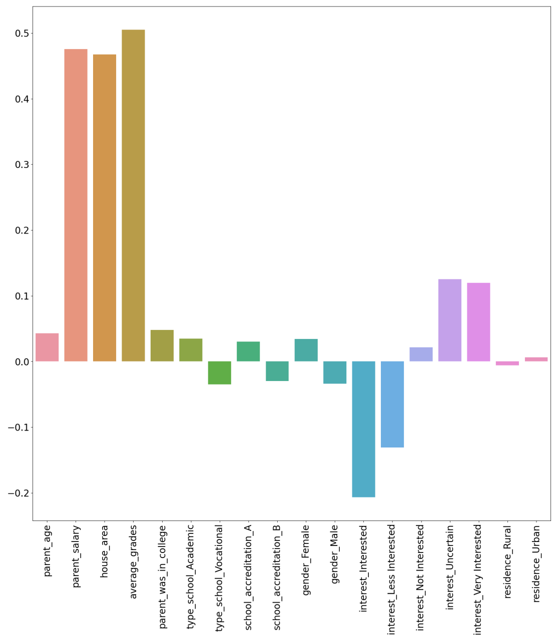

FIGURE 3: Correlation Histogram

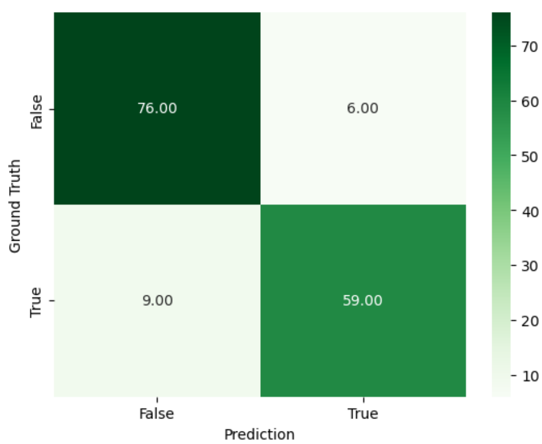

FIGURE 4: Confusion Matrix for ID3

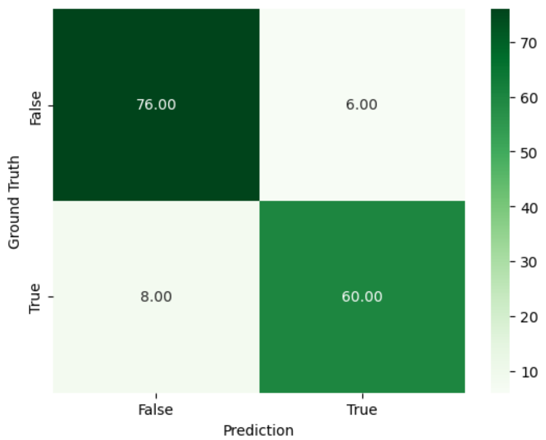

FIGURE 5: Confusion Matrix for C4.5

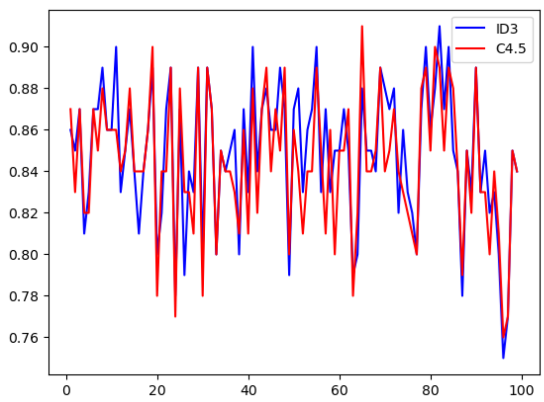

FIGURE 6: Performance with different seed values

APPENDIX B CODE

```python


# Implementing the ID3 and C4.5 decison tree algorithms from scratch, evaluating them and comparing their performance.

# In[52]:


import pandas as pd
import numpy as np
# from chefboost import Chefboost as chef
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
import pprint


# In[53]:


df = pd.read_csv('/kaggle/input/go-to-college-dataset/data.csv')
df_org = pd.read_csv('/kaggle/input/go-to-college-dataset/data.csv')


# # Dataset Analysis

# In[54]:


df.head(5)


# In[55]:


missing_values = df.isnull().sum()
print(missing_values)


# In[56]:


# One hot enocding
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_hot = pd.get_dummies(df)
df_scaled = pd.DataFrame(scaler.fit_transform(df_hot), columns = df_hot.columns)
df_hot.describe()


# In[57]:


df_scaled.head(5)


# In[58]:


corr = df_hot.corr()
m = np.triu(corr)
plt.figure(figsize = (12,12))
sns.heatmap(corr, annot= True, mask =m)


# In[59]:


plt.figure(figsize= (20, 20))
plt.xticks(rotation='vertical',  fontsize =20)
plt.yticks(fontsize=20)
correlation = corr['will_go_to_college']
correlation = correlation.drop(index= 'will_go_to_college', axis =0 )
sns.barplot(x = correlation.index , y = correlation.values)


# In[60]:


# Discretize all columns into 4 equal-width bins
for col in ['parent_age', 'parent_salary', 'house_area', 'average_grades']:
    df[col] = pd.cut(df[col], bins=4, labels=False)

df.head()


# In[61]:


for col in ['parent_age', 'parent_salary', 'house_area', 'average_grades']:
    plt.figure(figsize = (12, 7))
    data = df[col]
    data2 = df_org[col]
    plt.subplot(1,2,1)
    sns.histplot(data)
    plt.subplot(1,2,2)
    sns.histplot(data2, kde=True)
    plt.show()


# # Test train Split

# In[62]:


from sklearn.model_selection import train_test_split

X = df
Y = df.iloc[:,-1 ]
train, test, _ ,_ = train_test_split(X, Y, test_size = 0.15,random_state = 78 )

print('Train lenght', len(train))
print('Test lenght', len(test))


# # Implementing ID3 Tree

# ## Calculating parameters for split

# In[63]:


#function to calculate entropy

def entropy(data, label = 'will_go_to_college' ):
    counts = data[label].value_counts() #return values in each catagory
    total = len(data)
    entropy = 0

    for value in counts:
        prob = value/ total
        entropy = entropy - prob * np.log2(prob) #at the end postivie
#     for value in df{}
    return entropy


```

```
#Calculate information gain
def cal_info_gain(data):
    base_entropy = entropy(data)
    total = len(data)

    info_gain =[]  #stores info gain for each attrib when a tree is splitted

    for column in data.columns[:-1]:

        attribute_values = data[column].unique()  # get each attribute value
        new_entropy = 0.0


        for value in attribute_values:
            subset = data[data[column] == value]
            prob = len(subset)/total
            new_entropy += prob*entropy(subset)

        info_gain.append([column, base_entropy-new_entropy])

    return info_gain


#function check
print(cal_info_gain(df))


# ## Tree building

# In[64]:


#Recursively build the tree and store in the format {'col_name':{'split_attr':{}}

def build_tree_ID3(data, root=None):
    global split_info
    label = 'will_go_to_college'

    info_gain = cal_info_gain(data)
    info_gain = sorted(info_gain, key=lambda x: x[1], reverse=True)
    column_name = info_gain[0][0]    #best column to split on
#     print('Col:', column_name)

    root = {column_name: {}}

    for attr in data[column_name].unique():
#         print(attr)
        new_data = data[data[column_name] == attr]
        new_data = new_data.drop(column_name, axis=1)  # Drop the column used for splitting

        if len(new_data.columns) < 2:  #If the data is splitted in all columns and still is impure
            count = new_data[label].value_counts()
            count = count.sort_values(ascending=False)
            root[column_name][attr] = count.index[0]

        elif len(new_data) > 1 and len(new_data[label].unique()) > 1:
            new = build_tree_ID3(new_data, root)  #Recusive call for further splitting
            root[column_name][attr] = new

        else:
            output = new_data[label].unique()
            root[column_name][attr] = output[0]

    return root


# # Implementing C4.5

# ## Calculating parameters for split

# In[65]:


#Calculate gain
def Gain(data):
    base_entropy = entropy(data)
    total = len(data)

    gain =[]  #stores info gain for each attrib when a tree is splitted

    for column in data.columns[:-1]:

        attribute_values = data[column].unique()  # get each attribute value
        new_entropy = 0.0


        for value in attribute_values:
            subset = data[data[column] == value]
            prob = len(subset)/total
            new_entropy += prob*entropy(subset)

        gain.append([column, base_entropy-new_entropy])

    return gain


#function check
# print(cal_info_gain(df))


# In[66]:


# implementation of c4.5

def split_info(data):
    total = len(data)

    split_info_ =[]  #stores info gain for each attrib when a tree is splitted

    for column in data.columns[:-1]:

        attribute_values = data[column].unique()  # get each attribute value
        split_info_att = 0.0  #split info for an attribute


        for value in attribute_values:
            subset = data[data[column] == value]
            prob = len(subset)/total
            split_info_att -= prob * np.log2(prob)

```

```python
        split_info_.append([column, split_info_att])

    return split_info_


#gainratio = gain/splitinfo
def gain_ratio(data):
    gain = Gain(data)
    splitinfo = split_info(data)
    gain = np.array(gain)
    splitinfo = np.array(splitinfo)

    gain_ratio_ = list()   #stores gain ratio for every attribute

# #     for attr in gain[:, 0]:   #looping throuhg all attributes
# #         ratio = gain[:, 1]/split_info[:, 1]
#      try:
#         ratio = np.divide(gain[:, 1].astype(np.float64), splitinfo[:, 1].astype(np.float64))
#      except:
#         ratio =[]

    for a, b in zip(gain, splitinfo):
        if float(b[1])==0:  #handaling zero cases
            ratio =0
        else:
            ratio = float(a[1])/float(b[1])

        gain_ratio_.append([a[0], ratio])

    return gain_ratio_


# ## Tree Building

# In[67]:


#Recursively build the tree and store in the format {'col_name':{'split_attr':{}}

split_information = []  #Only to check, how tree is splitting, testing
def build_tree_C4_5(data, root=None):
    global split_info
    label = 'will_go_to_college'

    gain_ratio_ = gain_ratio(data)
    gain_ratio_ = sorted(gain_ratio_, key=lambda x: x[1], reverse=True)
    column_name = gain_ratio_[0][0]     #best column to split on
#     print('Col:', column_name)

    split_information.append(column_name)
    root = {column_name: {}}

    for attr in data[column_name].unique():
#         print(attr)
        new_data = data[data[column_name] == attr]
        new_data = new_data.drop(column_name, axis=1)   # Drop the column used for splitting

        if len(new_data.columns) < 2:  #If the data is splitted in all columns and still is impure
            count = new_data[label].value_counts()
            count = count.sort_values(ascending=False)
            root[column_name][attr] = count.index[0]

        elif len(new_data) > 1 and len(new_data[label].unique()) > 1:
            new = build_tree_C4_5(new_data, root)   #Recusive call for further splitting
            root[column_name][attr] = new

        else:
            output = new_data[label].unique()
            root[column_name][attr] = output[0]

    return root


# # Decision tree Inference

# In[68]:


#Make decision using the tree
def predict_decision_tree(data_point, decision_tree):
    node = decision_tree
    default_prediction = False

    while isinstance(node, dict):
        feature = list(node.keys())[0]
        value = data_point[feature]

        if value is None:
            return default_prediction

        try:
            node = node[feature][value]
        except KeyError:   #if key value is not present
            return default_prediction

    return node


# In[69]:


#Functions for evaluation of model

#take datrame input and returns report and confusion matrix
def generate_report(data, tree):
    predictions = []
    actual = data.iloc[:, -1]
    for index, row in data.iterrows():
        row = row.drop('will_go_to_college', axis = 0)
        pre = predict_decision_tree(row, tree)
        predictions.append(pre)

    #generate report
    report = classification_report(actual, predictions)
    matrix = confusion_matrix(actual, predictions)

    return report, matrix

#plots confusion matrix
def plot_matrix(matrix):
    labels= [False, True]
    sns.heatmap(matrix, annot = True, fmt=".2f",cmap='Greens',
```

```
        xticklabels = labels, yticklabels = labels)
    plt.xlabel('Prediction')
    plt.ylabel('Ground Truth')
    plt.show()


# # Training and Evaluation

# ### ID3

# In[71]:


#Training

id3_tree = build_tree_ID3(train)

# Train report
report, matrix = generate_report(train, id3_tree)
print('\nTrain Report\n')
print(report)
plot_matrix(matrix)


# Test report
report, matrix = generate_report(test, id3_tree)
print('\n Test Report\n')
print(report)
plot_matrix(matrix)


# ### C4.5

# In[72]:


#Training

c4_5_tree = build_tree_C4_5(train)

# Train report
report, matrix = generate_report(train, c4_5_tree)
print('\nTrain Report\n')
print(report)
plot_matrix(matrix)


# Test report
report, matrix = generate_report(test, c4_5_tree)
print('\n Test Report\n')
print(report)
plot_matrix(matrix)


# As the dataset is small, it's causing the decision tree to perform well somtimes when the seed if perfect but is performing poorly when the seed is not good.

# In[82]:


def accuracy(data, tree):
    predictions = []
    actual = data.iloc[:, -1]
    for index, row in data.iterrows():
        row = row.drop('will_go_to_college', axis = 0)
        pre = predict_decision_tree(row, tree)
        predictions.append(pre)

    predictions = np.array(predictions)
    accu = (predictions==actual).sum()/actual.shape[0]
    return accu




# In[102]:


seed_data = []
for i in range(1, 100):
    '''
    seed_data = ['seed', 'id3_acc', 'c4.5_acc']
    '''

    X = df
    Y = df.iloc[:,-1 ]
    train, test, _ ,_ = train_test_split(X, Y, test_size = 0.1,random_state = i )

    id3_tree = build_tree_ID3(train)
    c4_5_tree = build_tree_C4_5(train)

    accu_id3 = accuracy(test, id3_tree)
    accu_c4 = accuracy(test, c4_5_tree)
    seed_data.append([i, accu_id3,accu_c4 ])




# In[103]:


seed_data = np.array(seed_data)
plt.plot(seed_data[:, 0], seed_data[:, 1], color = 'blue')
plt.plot(seed_data[:, 0], seed_data[:, 2], color = 'red')
plt.legend(['ID3', 'C4.5'])
plt.show()


# In[106]:


seed_data[:, 1].mean(),  seed_data[:, 2].mean()


# In[99]:


plt.plot(seed_data[:, 0], seed_data[:, 1], color = 'blue')
plt.plot(seed_data[:, 0], seed_data[:, 2], color = 'red')
plt.legend(['ID3', 'C4.5'])
plt.show()


# In[104]:


seed_data[:, 1].mean()


# In[105]:


seed_data[:, 2].mean()
```

...

APPENDIX C AUTHORS

ARAHANTA POKHAREL was born in 1999 in Biratnagar, Nepal. He is a dedicated individual with a strong passion for learning and research. Currently pursuing a Bachelor's degree in Computer Technology at the Institute of Engineering, Thapathali Campus, he is in the final year of his studies. Throughout his academic journey, he has developed a keen interest in machine learning and data science. Additionally, he has a curious mind and actively engages in quizzes and current affairs to stay updated with the latest information and is committed to acquiring new skills.

AMIT RAJ PANT Amit Raj Pant is a dedicated student pursuing his studies in the Department of Electronics and Computers at Thapathali Engineering Campus, Tribhuvan University, located in Kathmandu, Nepal. With a strong passion for technology, his interests include computer vision and machine learning on resource-constrained edge devices, which involves performing computational tasks on local devices rather than relying solely on remote servers.