Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2023.0322000*

# Branching Towards Creditworthiness: A Decision Tree Approach for Credit Score Classification

**KHADKA PILOT[1*] AND POUDEL KSHITIZ[2*]**

[1]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: pilot.076bct025@tcioe.edu.np)
[2]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: kshitizpoudel18@gmail.com)

Corresponding author: Khadka Pilot (e-mail: pilot.076bct025@tcioe.edu.np).

* Authors contributed equally

**ABSTRACT** Decision trees are a popular machine learning algorithm used for both classification and regression tasks. They provide a straightforward and interpretable way of making decisions based on a series of features or attributes.This report explores the application of the decision tree classifier from the scikit-learn library for credit score classification and compares the performance of the algorithm using entropy and GINI index.We have used decision tree for prediction and classification of Credit score based on an individual's personal and financial information.

**INDEX TERMS** Decision Tree, Credit score classification

## I. INTRODUCTION

CREDIT refers to the practice of borrowing money. A credit score, from a statistical perspective, represents the probability of an individual repaying borrowed funds. A credit score holds significant importance in various scenarios, such as purchasing a home, starting a business venture, and more.Having a favorable credit scoregives easier access to loans, as it portrays you as a lower risk borrower. This project aims to delve into the classification of credit scores using decision trees.

Classification and regression are two fundamental tasks involved in Machine learning. Decision tree is one of the supervised learning method , which automatically learns rules based on the different values of the attributes for these tasks. Decision trees involve creating a flowchart like tree where each node of the tree denotes test on a attribute and splits the tree into more branches.The selection of the attributes to be tested on determines the size , complexity and performance of the tree. All attributes may or may not be used based on weather they are relevant or not. Thus a decision tree can provide useful insights from tabular data.We can assign any attribute as target and see the relation of other attributes with it. Decision trees not only acts as a very efficient and powerful rule based classifier but is also very easy to understand .Unlike some models like neural networks where it is difficult to interpret the parameters, decision tree are more easier to understand.

## II. METHODOLOGY

### A. THEORY

Decision trees work similarly to a large set of if-then rules. Geometrically it uses hyperplanes which run parallely to axes to cut the coordinate system into hyper cuboids to seperate the data into different classes. A decision tree consists of root nodes , internal nodes, branches and leaf nodes. The root node represents the entire dataset and consists of a root node from which split is made.This creates branch to internal nodes, where similarly a attribute is chosen and split is made. But the internal nodes do not represent the entire dataset . After splitting from the root node , the dataset is also divided and the new dataset does not consists the root attribute . finally the leaf nodes are the class labels .

#### 1) HUNTS Algorithm

**Input:**

- $D$: Training dataset with features and corresponding class labels.
- $A$: Set of attributes/features.

**Output:**

- $T$: Decision tree.

**Algorithm**

```
1) Create a root node for the decision tree, denoted as $T$.
2) If all examples in $D$ belong to the same class, then:
     • Label the root node with that class.
     • Return $T$.
3) If $A$ is empty (i.e., no attributes remaining), then:
     • Label the root node with the majority class in $D$.
```

```
        • Return $T$.
4) Calculate the information gain (IG) of each attribute in $A$:
        • $IG(A) = H(D) - H(D|A)$
   where $H(D)$ is the entropy of the class labels in $D$ and $H(D|A)$ is the conditional entropy of $D$ given attribute $A$.
5) Select the attribute $A$ with the highest information gain.
6) Label the root node with attribute $A$.
7) For each possible value $v$ of $A$, do:
        • Create a new branch from the root node for value $v$.
        • Let $D_v$ be the subset of $D$ with examples that have value $v$ for attribute $A$.
        • If $D_v$ is empty, then:
            -- Create a leaf node with the majority class in $D$.
        • Else:
            -- Recursively apply the algorithm to $D_v$ with attributes $A - A$.
            -- Attach the resulting subtree to the corresponding branch.
8) Return $T$.
```

### 2) Handling Continuous and Categorical Attributes

**Continuous Attributes:** For continuous attributes, decision tree algorithms use threshold values to create binary splits. Different algorithms employ different techniques to determine the best thresholds, such as exhaustive search or approximations.

**Categorical Attributes:** Categorical attributes can be handled by creating separate branches for each possible attribute value and evaluating the resulting impurity or entropy.

### 3) Pruning Techniques:

Decision trees are prone to overfitting, where they memorize the training data and fail to generalize well to unseen examples. Pruning is a technique used to reduce overfitting by removing unnecessary nodes from the tree.

**Pre-Pruning:** Pre-pruning involves stopping the tree construction early based on a predefined condition, such as limiting the maximum depth of the tree, minimum number of instances per leaf, or minimum information gain threshold.

**Post-Pruning:** Post-pruning, also known as tree trimming, involves growing the tree to its maximum size and then pruning it by recursively replacing subtrees with leaf nodes if it improves the overall accuracy or error rate on a separate validation set.

## B. SYSTEM BLOCK DIAGRAM

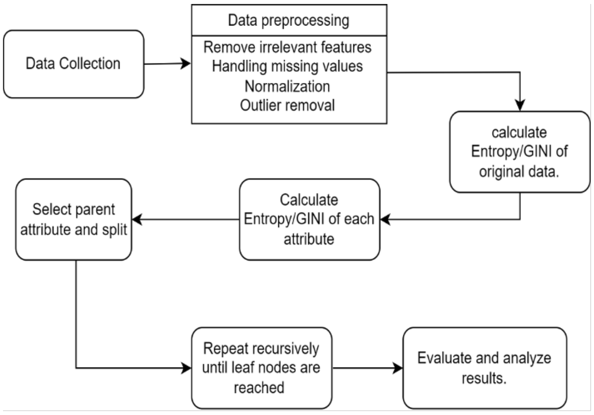

Data Collection
Data preprocessing
Remove irrelevant features
Handling missing values
Normalization
Outlier removal
calculate Entropy/GINI of original data.
Calculate Entropy/GINI of each attribute
Select parent attribute and split
Repeat recursively until leaf nodes are reached
Evaluate and analyze results.

**FIGURE 1.** System Block Diagram

## C. INSTRUMENTATION

The following libraries were instrumental in completion of this project :

- **Pandas**: Pandas is a powerful library in Python used for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data. It was utilized to read and preprocess the dataset, perform exploratory data analysis, and prepare the data for training the decision tree model. Functions such as `pd.read_csv()` were used to read the dataset from a CSV file, `pd.DataFrame()` to create data frames for data manipulation.
- **NumPy**: NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and a collection of mathematical functions to operate on these arrays efficiently. NumPy was employed for numerical computations and operations on arrays. Functions such as `np.array()` were used to create NumPy arrays, `np.random.shuffle()` to shuffle the data, and `np.unique()` to extract unique elements from an array.
- **Scikit-learn**: Scikit-learn is a widely used machine learning library in Python. It provides a collection of tools for various machine learning tasks, including classification, regression, clustering, and more. Scikit-learn was used to build and evaluate the decision tree model. Functions such as `sklearn.tree.DecisionTreeClassifier()` were used to create the decision tree classifier, `sklearn.model_selection.train_test_split()` for splitting the dataset into training and testing sets, and `sklearn.metrics.accuracy_score()` to calculate the accuracy of the model.
- **Matplotlib**: Matplotlib is a popular data visualization library in Python. Matplotlib was employed to visualize box plots and histograms, attribute correlation and confusion matrix.

## III. WORKING PRINCIPLE

Suppose we have dataset with k attributes and target variable. [1] Our goal is to predict target label from the information of attributes. The first is to calculate the entropy or GINI of the class labels of the dataset as:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{1}$$

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2 \tag{2}$$

Then we need to calculate entropy of each attribute. For a single attribute, we split the dataset on the basis of each value an attribute can take. Then we calculate resulting entropy/GINI of the divided datasets and calculate the weighted sum to calculate the entropy of that attribute.

$$\text{Entropy(Attribute)} = \sum_{i=1}^{n} \left( \frac{|S_i|}{|S|} \right) \cdot H(S_i) \tag{3}$$

where:

**Attribute** refers to the attribute being evaluated.

**S** is the original dataset.

$\mathbf{S_i}$ represents the subset of **S** after dividing it based on the values of **Attribute**.

**n** is the number of distinct values of **Attribute**.

$|S_i|$ represents the number of examples in subset $\mathbf{S_i}$.

$|S|$ is the total number of examples in dataset **S**.

$\mathbf{H(S_i)}$ is the entropy of subset $\mathbf{S_i}$.

The attribute with the least entropy or highest information gain is taken as the root node. This process is repeated again with remaining split data without the values of root node. The process is continued until all instances are classified or the tree reaches a certain depth.

## IV. RESULT AND ANALYSIS

### A. CREDIT SCORE CLASSIFICATION DATASET

The Credit score classification is a collection of data samples representing individuals and their respective credit profiles. It consists of 27 features and total of 150,000 instances. The dataset has been further divided into 100,00 instances for training and 50,000 instances for testing. The dataset is designed for the task of predicting the credit score of an individual based on their financial and personal information.

Certain attributes were excluded from the dataset based on their characteristics:

- **Nominal attributes**: `ID`, `Customer_ID`, `Month`, `Name`, and `SSN` were solely used for identification purposes and did not offer any additional information about the individuals. Hence, they were removed to streamline the dataset.
- **Derived attributes**: The attribute `Monthly inhand salary` provided information about an individual's monthly salary. However, since the dataset also contained the `Annual income` attribute, which can be used to calculate the monthly salary, the `Monthly_inhand_salary` attribute was deemed redundant and eliminated from the dataset.

Next, features were converted from categorical values into numerical:

**TABLE 1. Information Table**

| Field | Value |
|---|---|
| **ID** | 5634 |
| **Customer_ID** | 3392 |
| **Month** | 1 |
| **Name** | Aaron Maashoh |
| **Age** | 23.0 |
| **SSN** | 821000265.0 |
| **Occupation** | Scientist |
| **Annual_Income** | 19114.12 |
| **Monthly_Inhand_Salary** | 1824.843333 |
| **Num_Bank_Accounts** | 3.0 |
| **Num_Credit_Card** | 4.0 |
| **Interest_Rate** | 3.0 |
| **Num_of_Loan** | 4.0 |
| **Type_of_Loan** | auto loan,credit-builder loan |
| **Delay_from_due_date** | 3.0 |
| **Num_of_Delayed_Payment** | 7.0 |
| **Changed_Credit_Limit** | 11.27 |
| **Num_Credit_Inquiries** | 4.0 |
| **Credit_Mix** | Good |
| **Outstanding_Debt** | 809.98 |
| **Credit_Utilization_Ratio** | 26.82262 |
| **Credit_History_Age** | 265.0 |
| **Payment_of_Min_Amount** | No |
| **Total_EMI_per_month** | 49.574949 |
| **Amount_invested_monthly** | 21.46538 |
| **Payment_Behaviour** | High_spent_Small_value_payments |
| **Monthly_Balance** | 312.494089 |
| **Credit_Score** | Good |

- **Occupation**: This refers to an individual's chosen profession such as Law, Engineering, and Mechanics. A total of 15 distinct categories have been identified to capture the diversity of occupations.
- **Type of Loan**: This feature represents the specific type of loan an individual has obtained. Where no data is available, it indicates that the individual has not taken out any loans. The types of loans considered include Personal, Credit Builder, Debt Consolidation, among others.
- **Credit Mix**: The Credit Mix reflects the various types of loan accounts held by an individual, such as mortgages, general loans, and credit cards. To represent the state of an individual's credit mix, three categories have been established: Standard, Good, and Bad.
- **Payment Behavior**: It consists of patterns associated with purchasing goods and services which reveals individuals' spending habits. This includes their tendencies towards low spending on small value goods/services, high spending on large value goods/services, and more, providing valuable insights into their consumption behavior.

Outliers refer to data points that deviate significantly from the normal distribution or the expected pattern of the dataset. These outliers can have a substantial impact on the analysis and modeling process, potentially leading to skewed results and inaccurate predictions. For the dataset, IQR was used to identify the columns containing the outliers. Then, boxplot was used to visualize the columns containing the outliers.

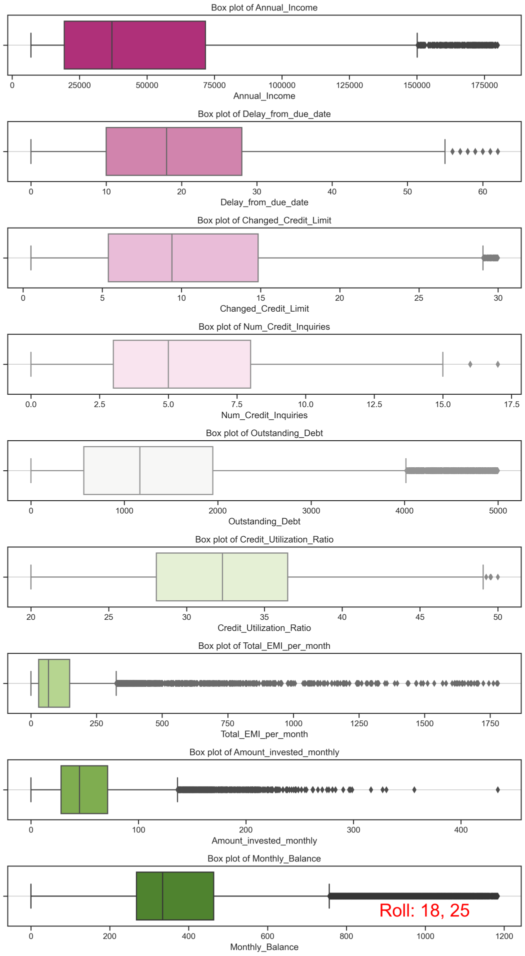


**FIGURE 2.** **Box plot of columns containing outliers**

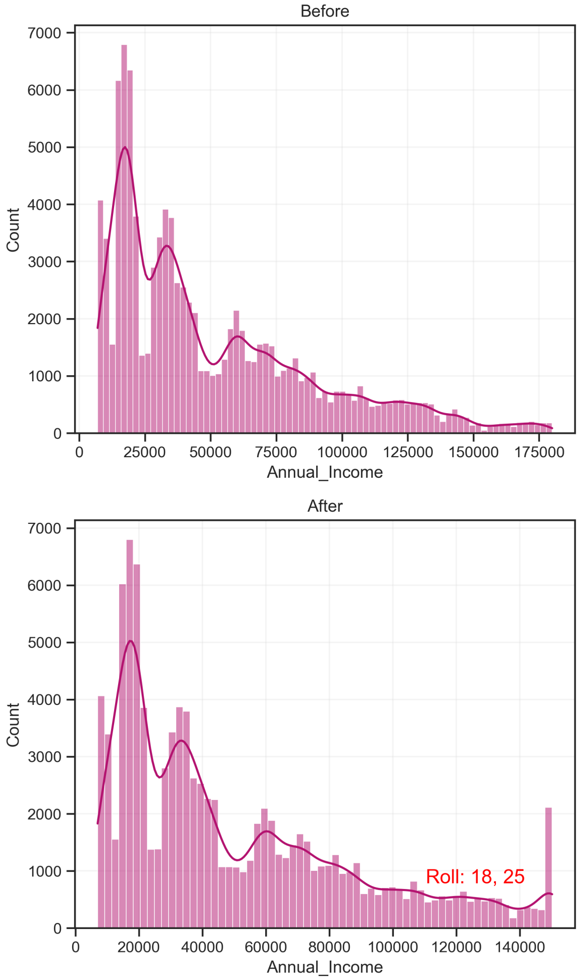


**FIGURE 3.** **Distribution with limited outlier values towards the upper whisker.**

The Interquartile Range (IQR) was used to identify the columns containing outliers. For that, we used the following formulas:

$$\text{lower\_limit} = q1 - \text{threshold} \times \text{IQR} \tag{4}$$

$$\text{upper\_limit} = q3 + \text{threshold} \times \text{IQR} \tag{5}$$

The outliers were then handled by capping or limiting them within the IQR range. As an example, a figure was provided showing the distribution with limited outlier values towards the upper whisker.

Correlation heatmap was then used to observe the relationship between the features.

From the figure, we can see a strong positive correlation between credit mix, bank account, interest rate, and the number of delayed payments.

**Credit mix and Delayed Payment:** It could indicate that individuals with a more diverse credit mix may have difficulty managing multiple credit obligations, leading to a higher likelihood of delayed payments.

**Bank Account and Delayed Payment:** The correlation between the bank account and the number of delayed payments suggests that individuals with a higher number of delayed payments may have a higher chance of having insufficient funds in their bank accounts to meet their payment obligations. This could be indicative of financial instability or poor financial management.

**Interest Rate:** The correlation between the interest rate and the number of delayed payments may imply that individuals with higher interest rates on their credit accounts are more likely to experience financial difficulties, making it challenging for them to make timely payments.

Regarding the negative correlation between credit history age and outstanding debt, credit mix, number of loans, and interest rate, it can be explained by several factors.

Individuals with a longer credit history age tend to exhibit responsible financial behavior, including lower outstanding debt, a more balanced credit mix, a lower number of loans, and potentially lower interest rates. Lenders and financial institutions consider credit history age when assessing credit-

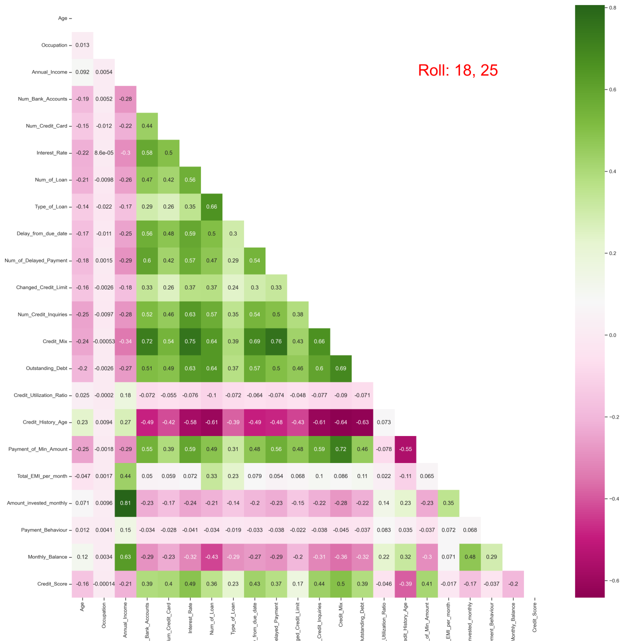


**FIGURE 4.** Correlation Matrix

worthiness and risk, favoring individuals with a longer credit history as they demonstrate a track record of managing credit responsibly.

Furthermore, a longer credit history age suggests greater financial stability and the ability to effectively manage credit obligations, leading to lower debt levels and better loan terms. Therefore, individuals with a longer credit history age tend to have lower credit utilization, which contributes to reduced outstanding debt and a more favorable credit mix.

To classify credit scores, the dataset was split into a 77% training set and a 33% testing set. The `DecisionTree Classifier()` algorithm from the `scikit-learn` library was employed. The model was trained using the training dataset and subsequently evaluated using the testing dataset. To assess the model's performance, a confusion matrix was generated and visualized in a figure. This matrix provides an overview of the model's predictions, highlighting the number of correct and incorrect classifications.

From the figure(5), we can observe the following percentages of correctly identified credit scores:

- Good: 68%
- Standard: 72%
- Poor: 75%

However, there are misclassifications evident in the confusion matrix. Notably, 31% of Poor credit scores were incorrectly classified as Good, and 26% were misclassified as Standard.

Similarly, a decision tree was created using the Gini index. The results showed a slightly improved accuracy of 74%, as evident from the heatmap visualization in figure (6).

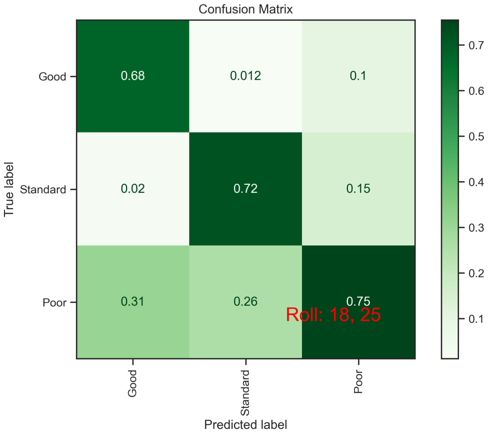


**FIGURE 5.** Confusion Matrix for entropy

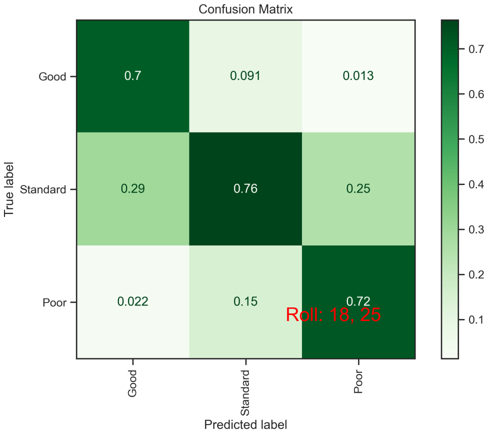


**FIGURE 6.** Confusion Matrix for Gini index

## V. DISCUSSION

The classification report for the decision tree using entropy is as follows:

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.67 | 0.69 | 0.68 | 5864 |
| 1 | 0.76 | 0.75 | 0.75 | 17529 |
| 2 | 0.73 | 0.72 | 0.73 | 9607 |

Accuracy: 0.73
Macro Avg: 0.72
Weighted Avg: 0.73

The F1-score considers both precision and recall. For class 0, the F1-score is 0.68, reflecting a balance between precision and recall. The F1-scores for class 1 and class 2 are 0.75 and 0.72, respectively.

The support column shows the number of samples in each class. Class 0 has a support of 5864, class 1 has a support of 17529, and class 2 has a support of 9607.

The macro average calculates the average precision, recall, and F1-score across all classes, treating each class equally. In this case, the macro average is 0.72.

The weighted average calculates the average precision, recall, and F1-score, taking into account the support of each class. It provides a weighted average that considers the imbalance in class distribution. The weighted average is 0.73 in this case.

The classification report for decision tree using Gini index is as follows: In this report, class 0 has an F1-score of 0.70, class 1 has an F1-score of 0.76, and class 2 has an F1-score of 0.72.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.69 | 0.70 | 0.70 | 5864 |
| 1 | 0.76 | 0.76 | 0.76 | 17529 |
| 2 | 0.73 | 0.72 | 0.72 | 9607 |

Accuracy: 0.74
Macro Avg: 0.73
Weighted Avg: 0.74

Support indicates the number of occurrences of each class in the dataset. In this report, class 0 has a support of 5864, class 1 has a support of 17529, and class 2 has a support of 9607. Similar to the previous report, macro average is 0.73 and weighted average is 0.74.

Comparing the two approaches, there appears to be a marginal performance improvement when utilizing the Gini index. However, this difference could potentially be due to the random state used. To obtain a more definitive understanding of the performance disparity between the Entropy and Gini index, conducting cross-validation could provide more reliable and conclusive results.

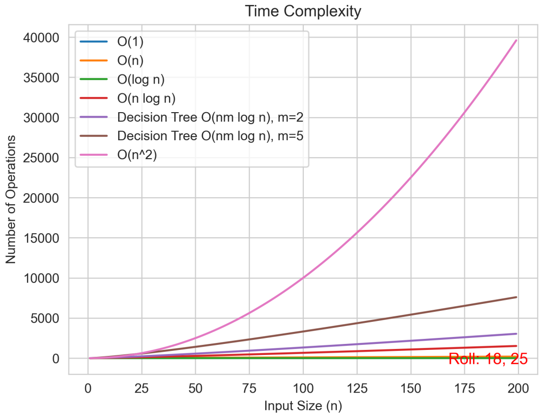


**FIGURE 7.** **Time complexity**

The time complexity of decision trees typically depends on various factors, including the number of samples (n) and the number of features (m) in the dataset, as well as the specific implementation of the decision tree algorithm. Figure shows the time complexity of decision tree of depth 2 and 5 along with time complexity of linear, logarithmic and quadratic time complexity.

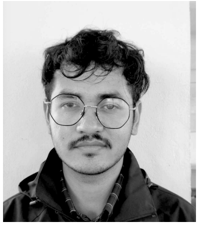
## VI. CONCLUSION

The application of decision trees in credit score classification provided valuable insights into their effectiveness and suitability for this task. Decision tree models demonstrated their ability to handle datasets with features that are relevant for assessing creditworthiness. The decision tree algorithm effectively analyzed factors such as credit history, outstanding debt, income, payment history, and employment status to classify credit scores. By creating a tree-like structure with decision nodes based on various features, decision trees could accurately predict and classify credit scores for individuals. Moreover, decision trees showcased their interpretability and explainability in the credit score classification task. By analyzing the decision tree structure, It was possible to understand the rules and conditions that contributed to credit score predictions.

## REFERENCES

[1] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 3rd edition. Morgan Kaufmann Publishers, 2012.

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *"Deep Learning."* MIT Press.

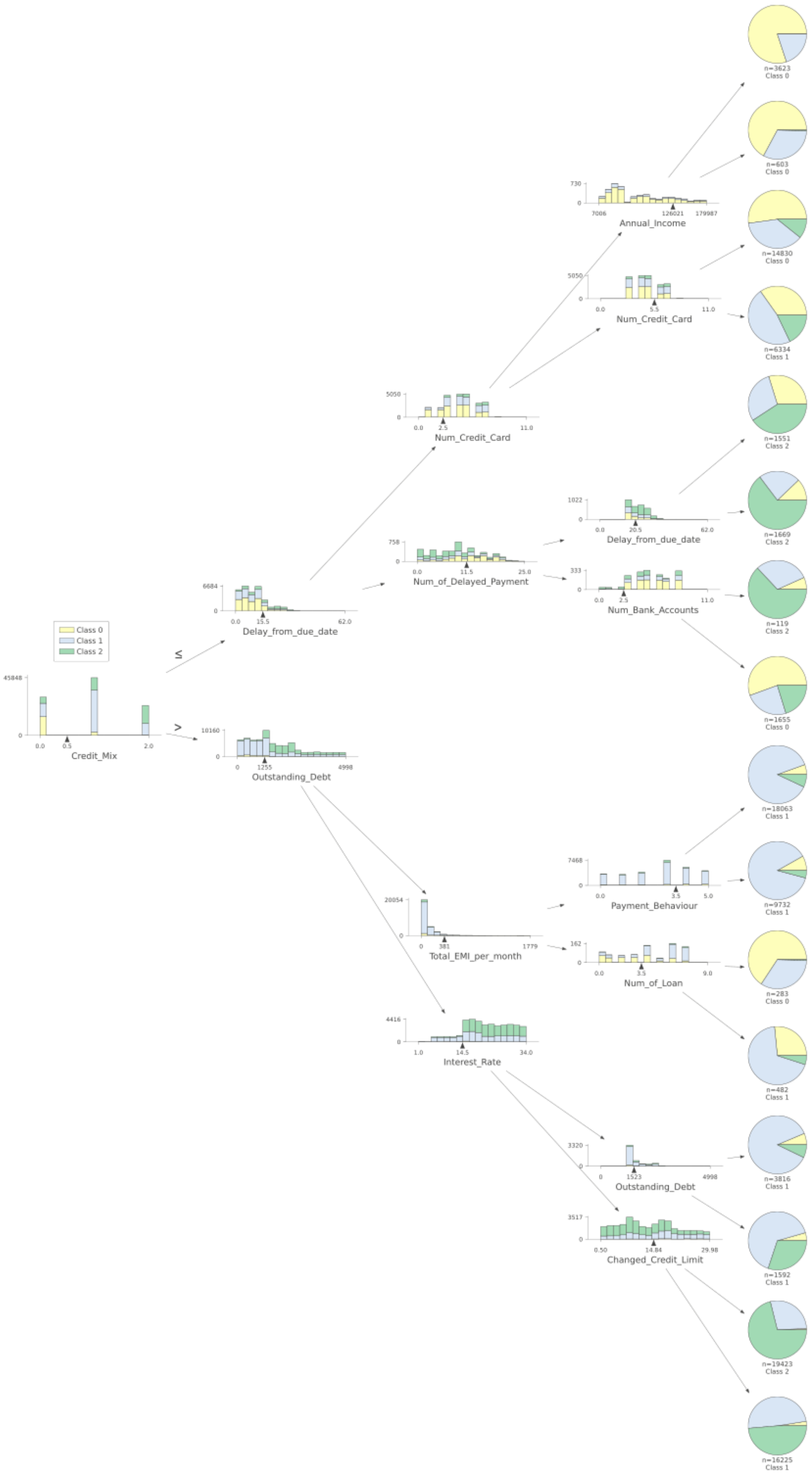


**PILOT KHADKA** is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. During his time at Thapathali Campus, Pilot has actively engaged in various academic and extracurricular activities. He has participated in coding competitions, collaborated on software development projects, and demonstrated a keen interest in exploring new technologies.

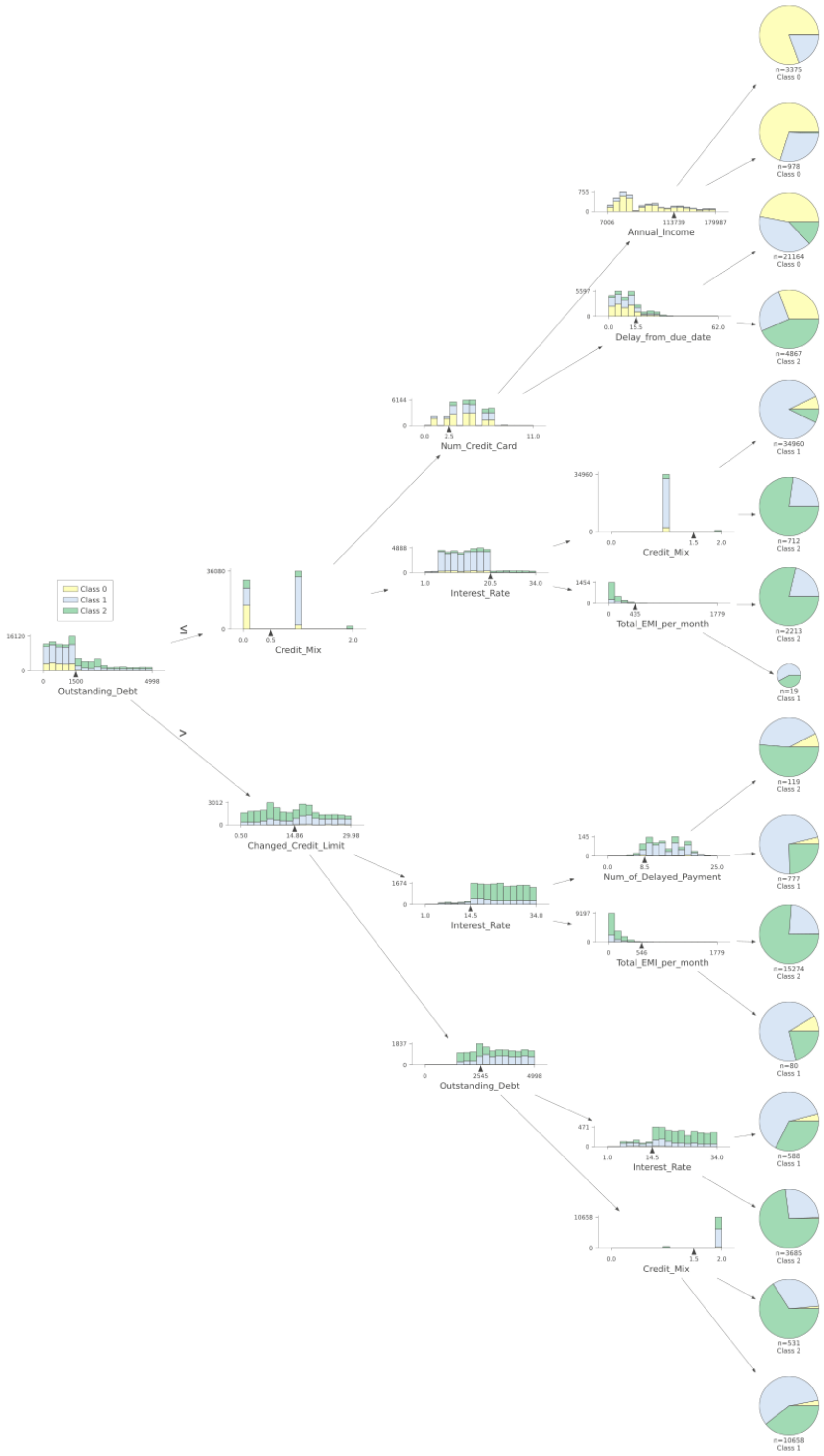
**KSHITIZ POUDEL** is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. His relentless pursuit of knowledge and dedication to uplifting and empowering others make him an exceptional contributor and an invaluable asset to the academic community.

## VII. APPENDIX

## VIII. DECISION TREE USING ENTROPY



## IX. DECISION TREE USING GINI INDEX



## CODE

### A. CREDIT SCORE CLASSIFICATION CODE

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

path = "cleaned/train.csv"
df = pd.read_csv(path)

df.describe()

# Remove Columns that will not be used for classification
d_col = ["ID", "Customer_ID", "Month", "Name", "SSN", "Monthly_Inhand_Salary"]

for _ in d_col:
    if _ in df.columns:
        df = df.drop(_, axis=1)

df.info()


# See Nominal values
for col in df:
    if df[col].dtypes == object:
        print(col)
        print("**" * 20)
        print(df[col].value_counts(dropna=False))
        print("**" * 20)


df["Credit_Score"]


# Conversion of Nominal data into Numeric
y_, label = pd.factorize(df["Credit_Score"])
df[df.select_dtypes(["object"]).columns] = df[
    df.select_dtypes(["object"]).columns
].apply(lambda x: pd.factorize(x)[0])

df.describe()

# finding Columns with Outliers using IQR method
def find_outliers(df, threshold=1.5):
    cols = []

    for _ in df.columns:
        q1 = np.percentile(df[_], 25)
        q3 = np.percentile(df[_], 75)
        iqr = q3 - q1
        lower_limit = q1 - threshold * iqr
        upper_limit = q3 + threshold * iqr

        if any((df[_] < lower_limit) | (df[_] > upper_limit)):
            cols.append(_)
    return cols


outlier_columns = find_outliers(df)
print(outlier_columns)

import matplotlib.pyplot as plt
import seaborn as sns

# Generate a color palette with a unique color for each box plot
num_plots = len(outlier_columns)
palette = sns.color_palette("PiYG", num_plots)

fig, axes = plt.subplots(nrows=num_plots, ncols=1, figsize=(10, 2 * num_plots))

for i, column in enumerate(outlier_columns):
    ax = axes[i]
    sns.boxplot(x=df[column], ax=ax, color=palette[i])
    ax.set_title(f"Box plot of {column}", fontsize=12)
    ax.set_ylabel("")
    ax.grid(True, axis="y")
plt.text(
    0.9,
    0.1,
    "Roll: 18, 25",
    ha="right",
    va="bottom",
    transform=plt.gca().transAxes,
    color="red",
    fontsize=24,
)
plt.tight_layout()
plt.savefig("outlier_box.png", dpi=300)
plt.show()


# Limit the Outliers to Upper limit and Lower Limit
threshold = 1.5
df2 = df.copy()
for col in outlier_columns:
    q1 = np.percentile(df[col], 25)
    q3 = np.percentile(df[col], 75)
    iqr = q3 - q1
    lower_limit = q1 - threshold * iqr
    upper_limit = q3 + threshold * iqr

    df2[col] = np.where(
        df[col] > upper_limit,
        upper_limit,
        np.where(df[col] < lower_limit, lower_limit, df[col]),
    )

"""for _ in outlier_columns:
    Q1 = df[_].quantile(0.25)
    Q3 = df[_].quantile(0.75)
    IQR = Q3 - Q1
    df = df.drop(df.loc[df[_] > (Q3 + 1.5 * IQR)].index)
    df = df.drop(df.loc[df[_] < (Q1 - 1.5 * IQR)].index)
df.info()"""

df["Annual_Income"]

# Box plot after handeling outliers
fig, axes = plt.subplots(
    nrows=len(outlier_columns), ncols=1, figsize=(10, 2.5 * len(outlier_columns))
)

for i, column in enumerate(outlier_columns):
    ax = axes[i]
    sns.boxplot(x=df2[column], ax=ax)
    ax.set_xlabel("Index", fontsize=12)
    ax.set_ylabel(column, fontsize=12)
    ax.set_title(f"Box plot of {column}", fontsize=14)

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns
```

```python

# Set the color palette
palette = sns.color_palette("PiYG", 14)

fig, axes = plt.subplots(2, 1, figsize=(6, 10))

# Plot "Before" distribution
sns.histplot(df["Annual_Income"], kde=True, ax=axes[0], color=palette[0], alpha=0.5)
axes[0].set_title("Before")

# Plot "After" distribution
sns.histplot(df2["Annual_Income"], kde=True, ax=axes[1], color=palette[0], alpha=0.5)
axes[1].set_title("After")

# Adjust alpha value for plot elements
for ax in axes:
    ax.set_facecolor((1, 1, 1, 1))  # Set background alpha value
    ax.grid(alpha=0.2)  # Adjust gridlines alpha value
plt.text(
    0.9,
    0.1,
    "Roll: 18, 25",
    ha="right",
    va="bottom",
    transform=plt.gca().transAxes,
    color="red",
    fontsize=14,
)
plt.tight_layout()
plt.savefig("limit.png", dpi=300)
plt.show()

corr = df.corr()

plt.figure(figsize=(20, 20))
matrix = np.triu(corr)
sns.heatmap(corr, cmap="PiYG", annot=True, mask=matrix)
plt.tight_layout()
plt.text(
    0.9,
    0.9,
    "Roll: 18, 25",
    ha="right",
    va="top",
    transform=plt.gca().transAxes,
    color="red",
    fontsize=34,
)
plt.savefig("matrix.png", dpi=300)
plt.show()


# Training Data
y = df["Credit_Score"]
X = df.drop("Credit_Score", axis=1)

from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion="entropy")


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100
)


clf = clf.fit(X_train, y_train)

predicted = clf.predict(X_test)
pred_label = label[predicted]
y_label = label[y_test]

print(pred_label)


print(y_label)

from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
    classification_report,
)

conf_mat = confusion_matrix(y_label, pred_label)
C = conf_mat / conf_mat.astype(np.float).sum(axis=1)
disp = ConfusionMatrixDisplay(confusion_matrix=C, display_labels=label)
fig, ax = plt.subplots(figsize=(8, 6))

# Use only the green color from the "PiYG" palette
cmap = plt.cm.get_cmap("PiYG")
cmap = cmap(np.linspace(0.5, 1, cmap.N))
cmap = cmap[:, 1:2]
cmap = plt.cm.colors.ListedColormap(cmap)

disp.plot(ax=ax, cmap="Greens", xticks_rotation="vertical")

plt.title("Confusion Matrix")
plt.tight_layout()
plt.text(
    0.9,
    0.1,
    "Roll: 18, 25",
    ha="right",
    va="bottom",
    transform=plt.gca().transAxes,
    color="red",
    fontsize=18,
)
plt.savefig("entropy.png", dpi=300)


print(classification_report(y_test, predicted))


clf_gini = tree.DecisionTreeClassifier(criterion="gini", random_state=0)


# fit the model
clf_gini.fit(X_train, y_train)


y_pred_gini = clf_gini.predict(X_test)



conf_mat = confusion_matrix(y_test, y_pred_gini)
C = conf_mat / conf_mat.astype(np.float).sum(axis=1)
disp = ConfusionMatrixDisplay(confusion_matrix=C, display_labels=label)
fig, ax = plt.subplots(figsize=(8, 6))
disp.plot(ax=ax, cmap="Greens", xticks_rotation="
    vertical")

plt.title("Confusion Matrix")
plt.tight_layout()
plt.text(
    0.9,
    0.1,
    "Roll: 18, 25",
    ha="right",
    va="bottom",
    transform=plt.gca().transAxes,
    color="red",
    fontsize=18,
)
plt.savefig("entropy2.png", dpi=300)


print(classification_report(y_test, y_pred_gini))
```

### B. TIME COMPLEXITY CODE

```

import matplotlib.pyplot as plt
import numpy as np


# O(1) - Constant Time Complexity
def constant_time(n):
    return np.ones_like(n)  # Return an array of ones with the same shape as n


# O(n) - Linear Time Complexity
def linear_time(n):
    return n


# O(log n) - Logarithmic Time Complexity
def logarithmic_time(n):
    return np.log2(n)


# O(n log n) - Linearithmic Time Complexity
def linearithmic_time(n):
    return n * np.log2(n)


# O(nm log n) - Decision Tree Time Complexity
def nmlogn_time(n, m):
    return n * m * np.log2(n)


# O(n^2) - Quadratic Time Complexity
def quadratic_time(n):
    return n**2


n = np.arange(1, 200)
m1 = 2
m2 = 5

# Calculate the number of operations for each complexity
operations_constant = constant_time(n)
operations_linear = linear_time(n)
operations_logarithmic = logarithmic_time(n)
operations_linearithmic = linearithmic_time(n)
operations_nmlogn1 = nmlogn_time(n, m1)
operations_nmlogn2 = nmlogn_time(n, m2)
operations_quadratic = quadratic_time(n)

plt.plot(n, operations_constant, label="O(1)")
plt.plot(n, operations_linear, label="O(n)")
plt.plot(n, operations_logarithmic, label="O(log n)")
plt.plot(n, operations_linearithmic, label="O(n log n)")
plt.plot(n, operations_nmlogn1, label="Decision tree O(nm log n), m=2")
plt.plot(n, operations_nmlogn2, label="Decision tree O(nm log n), m=5")
plt.plot(n, operations_quadratic, label="O(n^2)")

plt.xlabel("Input Size (n)")
plt.ylabel("Number of Operations")
plt.title("Time complexity")
plt.legend()
plt.text(
    0.9,
    0.1,
    "Roll: 18, 25",
    ha="right",
    va="bottom",
    transform=plt.gca().transAxes,
    color="red",
    fontsize=12,
)
plt.savefig("complexity.png", dpi=300)
plt.show()
```