Universidad Autónoma de Querétaro
DIVISIÓN DE INVESTIGACIÓN Y POSGRADO

Faculty of Engineering
MCIA

Activity 4

Decision Tree

Student:
Juan Manuel
Aviña Muñoz

Professor:
Dr. Marco
Aceves Fernandez

November 3rd, 2023

Contents

1 Objectives

The primary objective of this project is to implement a Decision Tree classifier to predict the likelihood of diabetes diagnosis using a well-known diabetes dataset. The dataset includes various features such as age, gender, BMI (Body Mass Index), blood pressure, and other health-related parameters. Each instance in the dataset is labeled with a binary indicator, representing whether a patient has been diagnosed with diabetes (1) or not (0).

Our goal is to build a Decision Tree model that can accurately predict the probability of a patient being diagnosed with diabetes based on these health-related features. Decision Tree models are powerful tools for classification tasks, and by utilizing this dataset, we aim to develop a predictive model that can assist in the early detection of diabetes. This model will enable healthcare professionals to make informed decisions and offer appropriate medical guidance, ultimately contributing to better patient care and improved public health outcomes.

2 Introduction

In this decision tree analysis, we will delve into the dataset related to diabetes predictions. Our goal is to create a data-driven model to predict whether individuals are at risk of diabetes based on several key features. This machine learning approach leverages a decision tree algorithm to make these predictions.

The decision tree algorithm, known for its simplicity and effectiveness, works by recursively splitting the dataset into subsets based on certain features. It aims to find the most informative features and determine the optimal splits that will result in accurate predictions. We want to predict outcomes, in this case, the likelihood of diabetes, by examining the attributes and their impact.

Our decision tree will consider various factors such as age, glucose levels, and other relevant parameters to classify individuals as either at risk for diabetes or not. This approach is guided by the idea that specific combinations of these attributes may be indicative of an individual's diabetic status.

Similar to the K-Nearest Neighbors (KNN) algorithm, which looks at the proximity of data points, the decision tree explores these factors and their influence on diabetes risk. By creating a decision tree model, we aim to provide insights into the factors that contribute to diabetes risk and make predictions about an individual's diabetic status based on their attributes.

3 Theoretical Foundation

3.1 Diabetes

According to the *Centers for Disease Control and Prevention* (*CDC*), diabetes is a chronic health condition that affects how our bodies turns food into energy; our body breaks down most of the food eaten into sugar and releases it into the bloodstream. When the blood sugar goes up, it signals the pancreas to release insulin. Insulin acts like a key to let the blood sugar into our body's cells for use as energy.

With diabetes, the body doesn't make enough insulin or can't use it as well as it should. When there isn't enough insulin or cells stop responding to insulin, too much blood sugar stays in the bloodstream. Over time, that can cause serious health problems, such as heart disease, vision loss, and kidney disease [1].

3.2 Decision Trees

Decision trees are a popular machine learning algorithm that is used for both classification and regression tasks. They are a type of supervised learning algorithm that is widely used for decision support and predictive [2].

A decision tree is a tree-like structure where an internal node represents a feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. It is a simple yet powerful algorithm for both classification and regression tasks. Decision trees are particularly useful for their interpretability, making them a valuable tool for explaining the reasoning behind a particular decision or prediction [3].

In programming terms, decision trees can be thought of as a vast structure of nested if-else conditions.

From a mathematical perspective, decision trees employ hyperplanes that align with a single axis, dividing the coordinate system into hypercuboids.

3.2.1 How do Decision Trees work

- **Feature Selection:** The algorithm begins by selecting the most significant feature from the dataset to create a node at the top of the tree. This is the feature that provides the best split for the data based on a certain criterion, such as Gini impurity or information gain in the case of classification.
- **Splitting the Data:** The data is then split into subsets based on the chosen feature. For each subset, the algorithm recursively applies the same process to create child nodes.
- **Stopping Criteria:** The algorithm continues to create branches and nodes until a stopping criterion is met, which might include reaching a certain depth in the tree, a minimum number of samples in a node, or no further improvement in the chosen criterion.

- **Leaf Nodes:** Once the tree is fully constructed, the leaf nodes represent the final decision or prediction.

3.2.2 Information Gain

Information gain guides the selection of the feature to split on at each tree-building step. It relies on the concept of entropy to assess the node's purity. Lower the value of entropy, higher is the purity of the node.

$$Entropy = -\sum_{i=1}^{n} p_i log_2 p_i \tag{1}$$

where n is the number of classes in target variable.

$$InformationGain = entropy(parent) - \sum_{i=1}^{n} weightedavg_i * entropy(child) \tag{2}$$

where n is the number of classes in selected feature.

3.2.3 Advantages

- Decision trees provide a clear, human-readable explanation of how decisions or predictions are made.
- They can predict circumstances previously unforeseen in the instances.
- They can model non-linear relationships between features and target variables.
- They can work with both categorical and numerical data with minimal data preprocessing.
- Decision trees can handle outliers without significant impact on the model's performance.

3.2.4 Disadvantages

- Decision trees are prone to overfitting the training data, which can lead to poor generalization to unseen data. Techniques like pruning are used to mitigate this.
- Small changes in the data can lead to different tree structures.
- In classification, they can create biased models when one class dominates the dataset.
- They are prone to error when when dealing with scenarios where there are few instances and many attributes, each with multiple observations.
- They are not well-suited for handling continuous attributes.

3.2.5 Iterative Dichotomiser 3

ID3 (*Iterative Dichotomiser 3*) is one of the earliest and foundational algorithms for decision tree construction in machine learning. It was developed by Ross Quinlan in 1986. ID3 is primarily used for classification tasks, where the goal is to categorize data into predefined classes or categories [4].

Advantages

- ID3 is a straightforward algorithm that is easy to understand and implement.
- The resulting tree structure is highly interpretable, making it useful for explaining the decision-making process.

Disadvantages

- ID3 is prone to overfitting because it doesn't include a tree-pruning mechanism.
- It may favor attributes with many distinct values.
- Designed for categorical data and cannot handle numerical features directly.

4 Methods and Materials

4.1 Dataset

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether a patient has diabetes, based on certain diagnostic measurements included in the dataset.

The dataset consists of 768 instances, each instance represents a patient.

Here are the key features and details of the dataset:

- **Pregnancies:** To express the Number of pregnancies.
- **Glucose:** To express the Glucose level in blood
- **BloodPressure:** To express the Blood pressure measurement
- **SkinThickness:** To express the thickness of the skin
- **Insulin:** To express the Insulin level in blood
- **BMI:** To express the Body mass index
- **DiabetesPedigreeFunction:** To express the Diabetes percentage
- **Age:** To express the age
- **Outcome:** To express the final result 1 is Yes and 0 is No, our target feature.

4.2 Resources

Equipment:

- Intel i7-9750H @ 2.60 GHz processor.
- 16 GB DDR4 @ 2666 MHz.
- 1 Tb HDD + 1 Tb SSD.
- GeForce GTX 1660Ti 6 GB VRAM GDDR6.

4.3 Libraries

The following libraries were utilized for this activity:

- **NumPy** is a fundamental package for scientific computing in Python as it provides support for large, multi-dimensional arrays and matrices, along with a variety of mathematical functions to operate on these arrays. NumPy is a core library in the data science ecosystem and is used for tasks like numerical computations, linear algebra operations, random number generation, and more.

- **NumPy log2** is a function used for computing logarithms with base 2.
- **Pandas** is a powerful library for data manipulation and analysis in Python. It provides data structures (primarily Series and DataFrame) to efficiently handle and manipulate structured data. Pandas enables tasks such as loading data from various file formats (CSV, Excel, SQL databases), cleaning and preprocessing data, filtering, merging, transforming, and summarizing data. It's widely used for data wrangling and exploration.
- **Seaborn** is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations like scatter plots, bar plots, histograms, box plots, and more. It also supports color palettes, themes, and additional statistical plotting capabilities.
- **Matplotlib** is a versatile 2D plotting library in Python. It provides a wide range of functions to create static, interactive, and animated visualizations. While it can be used to create basic plots, it can also be customized to create complex visualizations. Matplotlib is the foundation for many other visualization libraries and tools.
- **Scikit Learn** is a machine learning library. It's *preprocessing* module offers various functions for data preprocessing tasks, such as scaling, data imputation, encoding categorical values and much more.
- **Scikit Learn plot_tree** is a module for visualizing decision trees.
- **Scikit Learn train_test_split** is a module which provides a function for splitting a dataset into training and testing subsets.
- **Scikit Learn confusion_matrix** is a module that provides functions to compute confusion matrices, used to assess the performance of classification modules.
- **Pprint** is a module that provides a pretty-printing function for a more readable output, the decision tree in this case.
- **eps = np.finfo(float).eps** calculates the smallest expressible positive number such that $1.0 + eps! = 1.0$.

These libraries are essential for various data analysis, data preprocessing, and data visualization tasks in Python.

5 Results

The first step is to import the libraries for this project, as seen on Figure 1.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as py
import plotly.graph_objs as go
eps = np.finfo(float).eps
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from numpy import log2 as log
from pprint import pprint
import seaborn as sns
```

Figure 1: *Project libraries.*

Now, we import the dataset; Figure 2 shows that we have 768 instances and 9 features, the "*Outcome*" is our target feature.

```
Total instances: 768
Total features: 9
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Figure 2: *Dataset visualization.*

Figure 3 shows the types of data in our dataset, as well as the amount of null values and the memory usage.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Figure 3: *Dataset description.*

To confirm that our dataset has no null values, we show this information on Figure 4.

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Figure 4: *Null values.*

Figure 5 displays the quantity of healthy patients, whereas Figure 6 illustrates the distribution in terms of percentages.

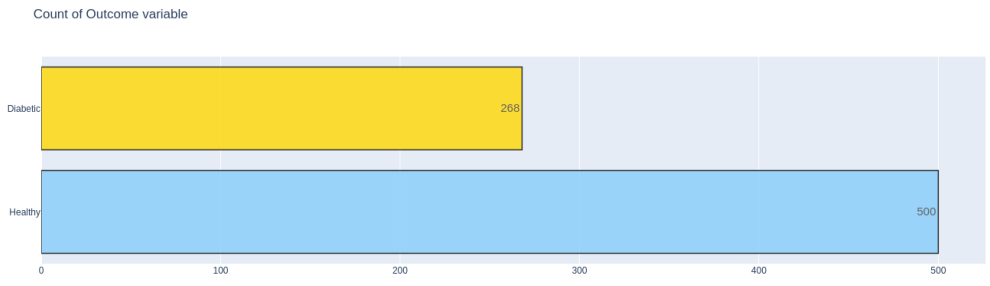

Figure 5: *Quantitative distribution.*

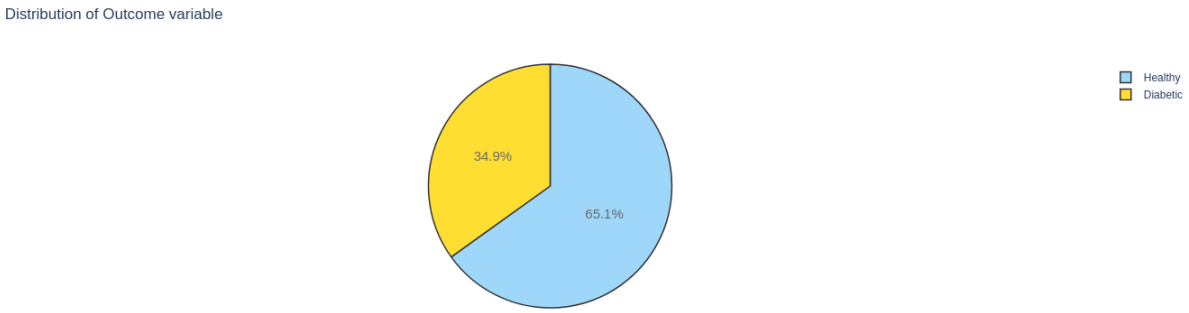

Figure 6: *Percentage distribution.*

On Figure 7 we can see a description of our data.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Figure 7: *Data description.*

Figure 8 depicts the correlation among the features, while Figure 9 presents this correlation in the form of a heatmap.

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

Figure 8: *Feature correlation.*

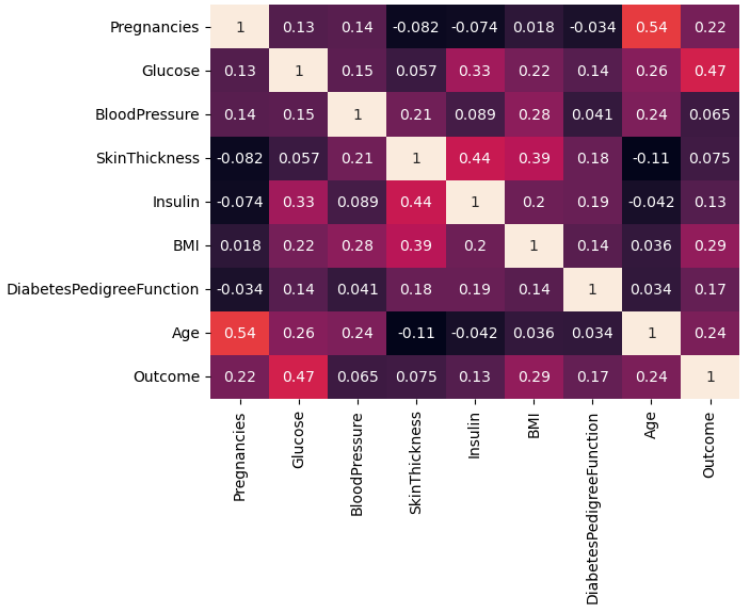

Figure 9: *Percentage distribution.*

Figure 10 shows a histogram for the distribution of the patients W.R.T. the ”*Outcome*” feature.

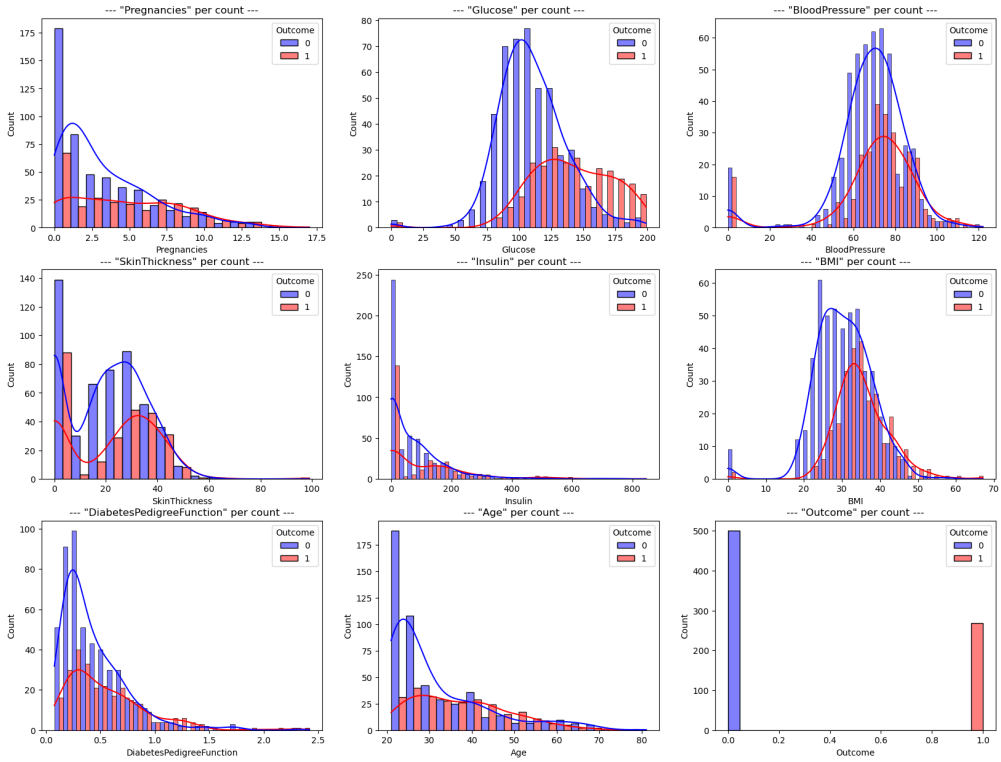

Figure 10: *Feature distribution w.r.t. "Outcome".*

Figure 11 shows a boxplot for the distribution of the patients W.R.T. the "*Outcome*" feature.

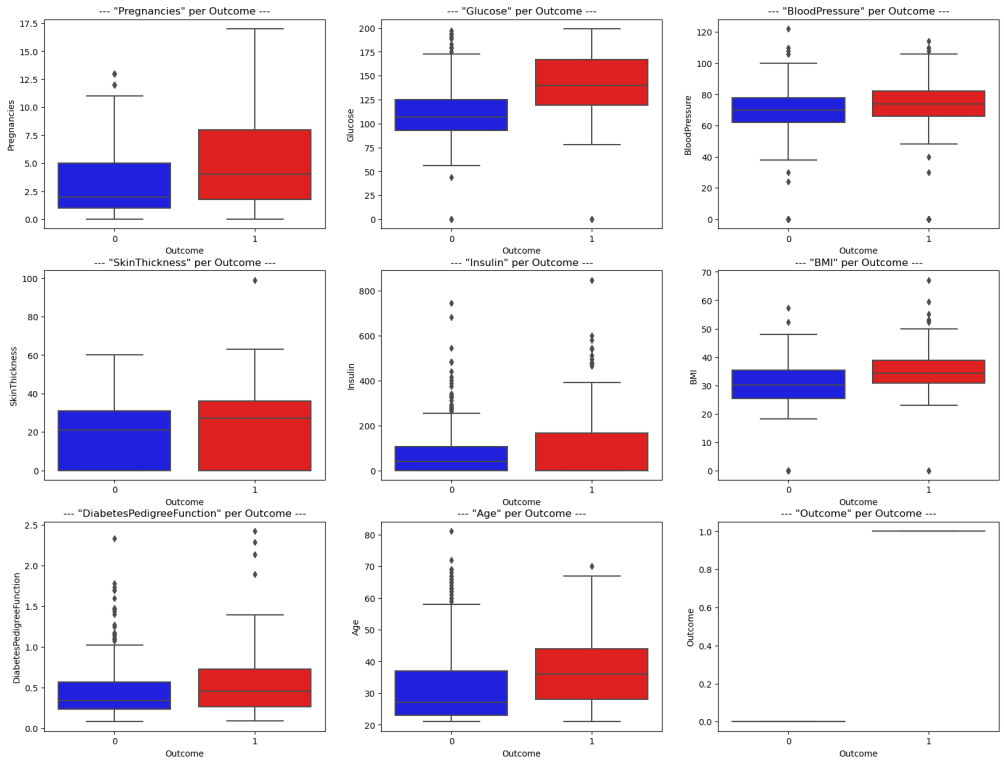

Figure 11: *Boxplot distribution w.r.t. "Outcome".*

By doing our EDA, we notice "*BloodPressure*" and "*SkinThickness*" features are not completely necessary, so we proceed to remove them.

After this, we proceed to transform the data in some of our features to discretize the continuous data in the specified columns into categories or bins, making it easier to work with the data for our modeling purposes.

- "*Pregnancies*" feature will be divided into four quantile ranges, and they will be labeled as '0', '1', '2', and '3'.
- "*Insulin*" feature will be divided into two quantile ranges, and they will be labeled as 'low' and 'high'. This suggests a binary categorization of insulin levels.
- "*BMI*" feature will be divided into two quantile ranges and labeled as '0' and '1'. This likely represents a binary categorization of BMI values.
- "*DiabetesPedigreeFunction*" feature will be divided into four quantile ranges, and they will be labeled as '0', '1', '2', and '3'.
- "*Age*" feature will be divided into three quantile ranges, and they will be labeled with the numerical values 1, 2, and 3.

The next step is to calculate the entropy for each of our features, shown in Figure 12

```
{'Pregnancies': 0.8906174365362114,
 'Glucose': 0.6289331894874066,
 'Insulin': 0.9323589424690207,
 'BMI': 0.8875090613833397,
 'DiabetesPedigreeFunction': 0.9111364000967174,
 'Age': 0.8682334538933789}
```

Figure 12: *Entropy w.r.t. features.*

Figure 13 and 14 show the code created to build a decision tree using the ID3 algorithm.

```
class Make_tree(object):
    def __init__(self):
        pass

    # Method to build a decision tree using the ID3 algorithm
    def buildTree(self, df, tree=None):
        # Get the name of the target variable (class)
        Class = df.keys()[-1]

        # Find the best attribute to split the data on
        node = self.find_winner(df)

        # Get unique values of the selected attribute
        attValue = np.unique(df[node])

        # Create an empty dictionary to represent the decision tree
        if tree is None:
            tree = {}
            tree[node] = {}

        # Recursively construct the decision tree
        for value in attValue:
            subtable = self.get_subtable(df, node, value)
            clValue, counts = np.unique(subtable[df.columns[-1]], return_counts=True)

            if len(counts) == 1:
                # If the subset is pure (all samples belong to one class), assign that class
                tree[node][value] = clValue[0]
            else:
                # Continue building the tree recursively
                tree[node][value] = self.buildTree(subtable)

        return tree

    # Calculate the entropy of the target variable in the dataset
    def find_entropy(self, df):
        Class = df.keys()[-1]
        entropy = 0
        values = df[Class].unique()
        for value in values:
            fraction = df[Class].value_counts()[value] / len(df[Class])
            entropy += -fraction * np.log2(fraction)
        return entropy

    # Calculate the entropy of a specific attribute in the dataset
    def find_entropy_attribute(self, df, attribute):
        Class = df.keys()[-1]
        target_variables = df[Class].unique()
        variables = df[attribute].unique()
        entropy2 = 0
        for variable in variables:
            entropy = 0
            for target_variable in target_variables:
                num = len(df[attribute][df[attribute] == variable][df[Class] == target_variable])
                den = len(df[attribute][df[attribute] == variable])
                fraction = num / (den + eps)  # Note: 'eps' is a small positive value
                entropy += -fraction * log(fraction + eps)
            fraction2 = den / len(df)
            entropy2 += -fraction2 * entropy

        return abs(entropy2)

    # Find the attribute with the highest information gain
    def find_winner(self, df):
        Entropy_att = []
        IG = []
        for key in df.keys()[:-1]:
            IG.append(self.find_entropy(df) - self.find_entropy_attribute(df, key))
        return df.keys()[:-1][np.argmax(IG)]

    # Get a subtable of the dataset where the attribute equals a specific value
    def get_subtable(self, df, node, value):
        return df[df[node] == value].reset_index(drop=True)
```

Figure 13: *Code snippet.*

```python
class DecisionTreeID3:
    def __init__(self):
        self.tree = None

    def fit(self, df):
        # Create an instance of the Make_tree class
        maketree = Make_tree()
        # Build the decision tree using the Make_tree instance
        self.tree = maketree.buildTree(df)

    def predict(self, df):
        # Initialize an empty list for storing predictions
        predictions = []
        for _, sample in df.iterrows():
            # Start the tree traversal from the root
            predicted_class = self.pred(self.tree, sample)
            predictions.append(predicted_class)
        return np.array(predictions)

    def pred(self, tree, sample):
        if not isinstance(tree, dict):
            return tree  # Reached a leaf node, return the predicted class
        # Get the attribute to split on from the current tree node
        root_node = next(iter(tree))
        feature_value = sample[root_node]
        if feature_value in tree[root_node]:
            # Continue traversing the tree with the chosen branch
            return self.pred(tree[root_node][feature_value], sample)
        else:
            # If the value is not in the tree, take the first branch (for simplicity)
            feature_value = list(tree[root_node].keys())[0]
            return self.pred(tree[root_node][feature_value], sample)
```

Figure 14: *Code snippet.*

To train and evaluate our algorithm, we partitioned our data into an $80/20$ split.

Now, we proceed to print our decision tree using the *pprint* function, Figure 15 shows part of the resulting decision tree.

```
{'Glucose': {0: {'Pregnancies': {'0': 0, '2': 1}},
             44: 0,
             56: 0,
             57: 0,
             61: 0,
             65: 0,
             67: 0,
             68: 0,
             71: 0,
             72: 0,
             73: 0,
             74: 0,
             75: 0,
             76: 0,
             77: 0,
             78: {'Age': {1: 0, 2: 1, 3: 0}},
             79: 0,
             80: 0,
             81: 0,
             82: 0,
             83: 0,
             84: {'Pregnancies': {'0': 0, '1': 0, '2': 0, '3': 1}},
             85: 0,
             86: 0,
             87: 0,
...
             196: 1,
             197: {'Pregnancies': {'1': 1, '2': 0, '3': 1}},
             198: 1,
             199: 1}}
```

Figure 15: *Decision tree for our ID3.*

This approach yields an accuracy of 61.04%.

Finally, Figure 16 shows a confusion matrix to observe our model behavior.

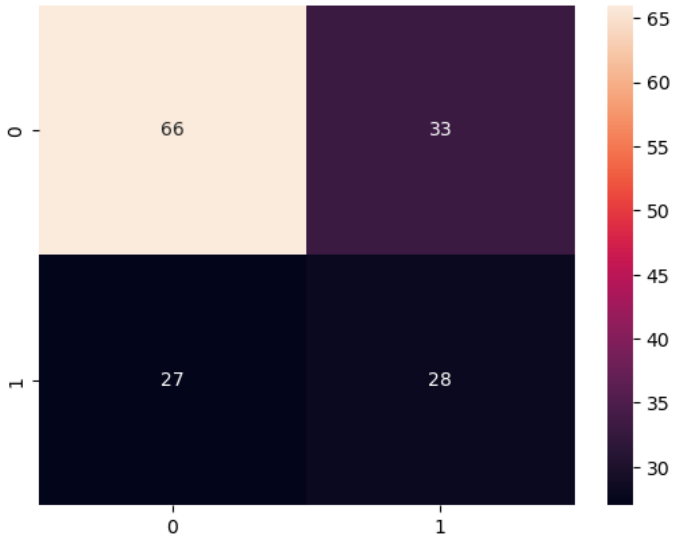

Figure 16: *Confusion matrix for our ID3.*

For the sake of comparison, I employed the ID3 algorithm from the "*Scikit Learn*" library, once more dividing our data into an $80/20$ split.

To evaluate our data, we employed the "*entropy*" criterion to assess the purity of our nodes.

This approach yields an accuracy of 73.38%.

Finally, Figure 17 shows a confusion matrix to observe our model behavior.

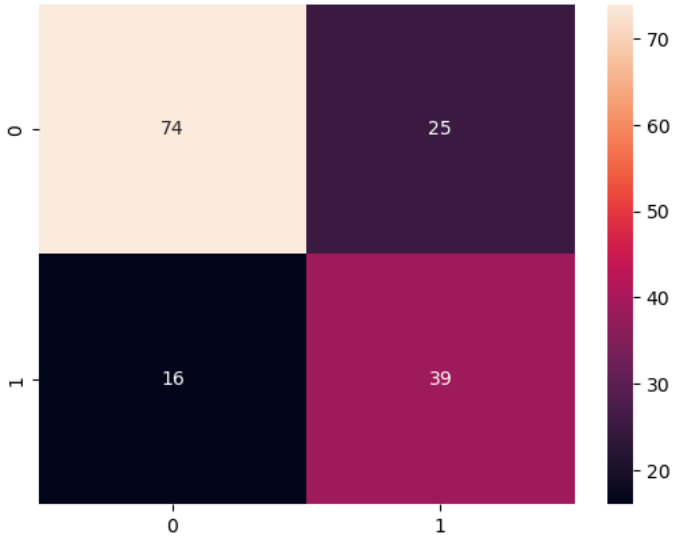

Figure 17: *Confusion matrix for "Scikit Learn" ID3.*

Figure 18 shows the decision tree created.

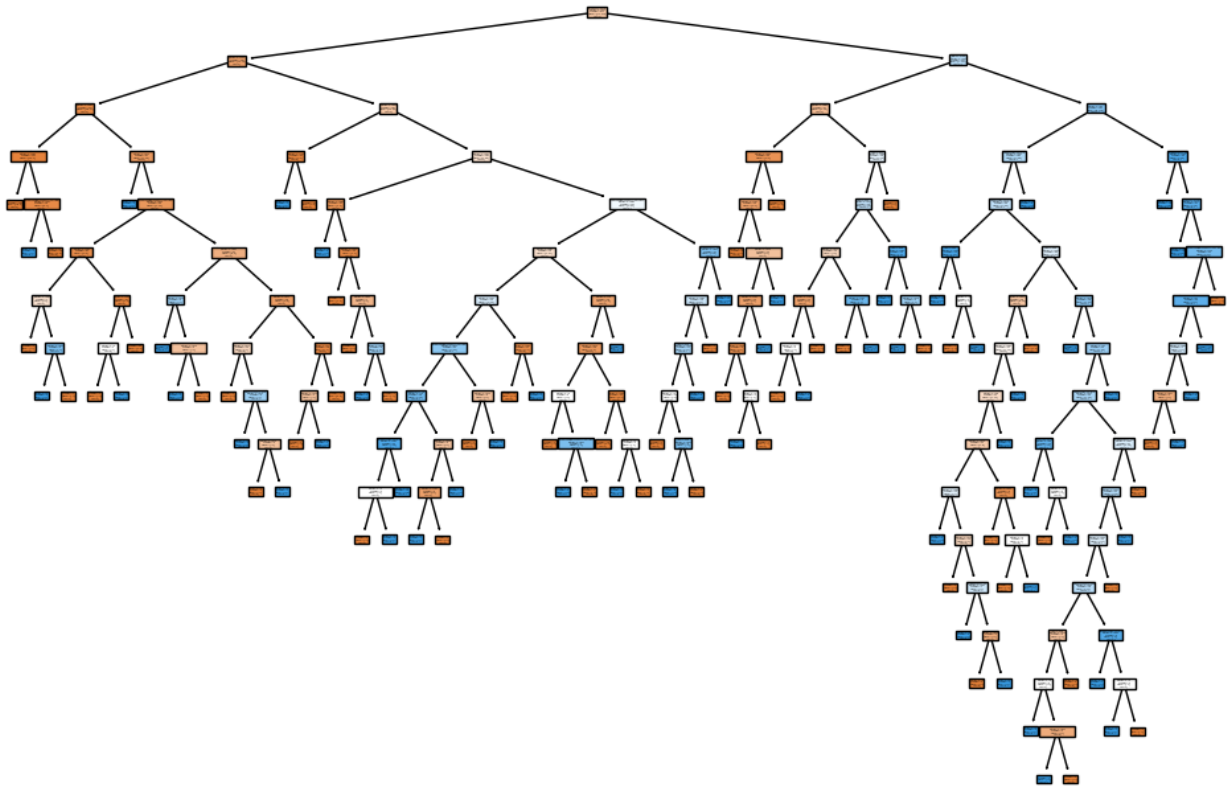

Figure 18: *Decision tree for "Scikit Learn" ID3.*

6 Discussion

We observe that our self-created algorithm produces a lower accuracy, mainly due to the fact that *Scikit Learn's* ID3 algorithm is highly optimized and efficient. It excels at handling extensive datasets and intricate tree structures, outperforming manual construction in terms of speed.

Our manual creation, on the other hand, provides transparency, as the tree is meticulously constructed from scratch. In contrast, *Scikit Learn's* algorithm remains somewhat of a black box, making it less transparent in terms of its decision-making process.

To summarize, manual ID3 decision tree construction is valuable for educational purposes and scenarios where customized control and transparency are paramount. However, it may not be suitable for large or time-sensitive projects; using *Scikit Learn's* ID3 algorithm offers efficiency, reduced error, and enhanced scalability, albeit at the potential cost of reduced transparency and customization.

7 Conclusions

In conclusion, our exploration of the diabetes dataset using decision trees has been a meaningful journey into the realm of data analysis and machine learning. We have observed the versatility of decision trees, ranging from a custom-crafted approach to utilizing *Scikit Learn's* library, and recognized their significance as a foundational model for diabetes prediction.

However, it is essential to acknowledge the inherent limitations of decision trees, including their vulnerability to overfitting and sensitivity to splitting criteria. These limitations remind us of the need for careful model selection and parameter tuning in the pursuit of accurate predictions.

Throughout this activity, decision trees played the role of our reference model, serving as a basis for comparison with more complex machine learning algorithms. They played a pivotal role in establishing a performance benchmark for the evaluation of these advanced models.

In summary, this activity has provided us with valuable insights into diabetes prediction through a data-driven approach. It highlights the pivotal role of data science in addressing significant healthcare challenges and, ultimately, enhancing patient outcomes.

References

[1] Centers for Disease Control and Prevention (CDC). (2023) Diabetes. [Online]. Available: https://www.cdc.gov/diabetes/basics/diabetes.html

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Decision Trees: A decision science perspective on decision trees*. Wadsworth, 1984.

[3] J. R. Quinlan, “Simplifying Decision Trees,” *International Journal of Man-Machine Studies*, vol. 27, pp. 221–234, 1987.

[4] ——, “Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[5] L. Pierson, *Data Science for Dummies*. Wiley, 2015.

[6] M. Aceves, *Inteligencia Artificial Para Programadores Con Prisa*. Universo de Letras, 2021.

[7] P. E. Utgoff, “Incremental induction of decision trees,” *Machine Learning*, vol. 4, no. 2, pp. 161–186, 1989.

[8] Pima Indians, “Diabetes Dataset,” 2022, Obtained from Kaggle on October 30, 2023. [Online]. Available: https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset

[9] P. H. Winston, *Artificial Intelligence, Third Edition*. Addison-Wesley, 1992.