FSU Panama City

EEL 4887 - CpE Language: Introduction to Python

Performed by:

Jaehyun Lee

Instructor: Dr. Manzak

Mar 30, 2023

Supervised Learning in Quest

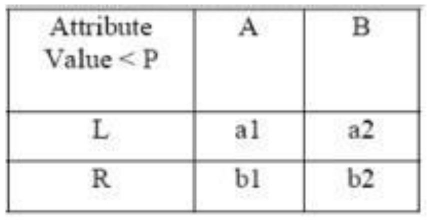
Abstract

This paper aims to explore the Supervised Learning In Quest (SLIQ) algorithm and its implementation for creating a decision tree classifier. The paper uses a dataset related to loan approval and implements the algorithm using Micropython and Python3. The primary objective of this study is to create a decision tree classifier using SLIQ for loan approval prediction. The methodology involves implementing the SLIQ algorithm using both Python3 and Micropython and creating a decision tree classifier for loan approval. The paper analyzes the performance of the classifiers in terms of accuracy and efficiency. The results show that the SLIQ algorithm can be used to create a decision tree classifier with high accuracy rates for loan approval prediction. However, the implementation of the algorithm in Micropython is limited, and only the Gini index of the first node of the classifier could be calculated. However, using Python3, the entire decision tree classifier could be created. The SLIQ algorithm is an effective machine learning technique for creating decision tree classifiers. The study shows that while Micropython has certain limitations, Python3 provides a more efficient and effective platform for implementing the SLIQ algorithm. The research contributes to the understanding of the SLIQ algorithm and its implementation in different computer programming language for creating decision tree classifiers.

Table of Contents

Introduction

SLIQ (Supervised Learning in Quest) is a high speed and flexible decision tree classifier developed by IBM Almaden Research Center in 1996 that allows to sort and interpret data. Data classification is a bottleneck in data mining due to other methods' inability to scale with large data sets spread across different classifications. SLIQ can reduce costs using efficient and pre-sorting decision trees to sort through larger data sets while accounting for differences in data types. This maintains competitive accuracy with the ability to scale and interpret larger data sets with multiple classes and attributes. The primary objective of this course project was to develop a decision tree classification model using a significant training dataset and apply it to another extensive dataset for predicting results. Utilization of the computer programming language, Python, made it possible to carry out the larger datasets, as well as allowing calculations and generation of the decision tree classification more efficiently. As more sub-nodes are created the bucketed data's homogeneity increases, the data becomes more similar, and the purity of the node increases with respect to the target variable. SLIQ improves on this framework by aiming to reduce the diversity of the tree at each split. This allows SLIQ to sort through data more efficiently and cost-effectively over large data sets with different data types[1]. SLIQ Algorithm can be divided into 3 steps, pre-sorting the sample, processing evaluation on splits, and updating the class list [4]. Presorting involves preprocessing the input data by sorting it in ascending order for each feature. By doing so, the algorithm can quickly calculate the impurity of each possible split point at a node by comparing the target variable values of consecutive instances in the sorted feature. Without presorting, the algorithm would need to scan the entire dataset and calculate the impurity of each possible split point, which can be computationally expensive for large datasets. Once the data is sorted, it can process evaluation on splits by assuming the split of the first node and evaluate each histogram. (Gini index will be used during this step but for this example since attribute, age and salary do not have predictive power since they can be any number. For training set L with n distinct classes the equation is shown above where the variables are given within the histogram example. SLIQ is an improvement upon existing decision tree algorithms. Then the class list can be updated. Traverse the training data through the decision tree and replace the node with the new node. These steps can be repeated until each of the leaf nodes becomes a pure node, meaning that the node only contains one class. SLIQ's advantages are based in its pre-sorting of data, no need for data normalization, scales well with data size, and



Attribute Value < P	A	B
L	a1	a2
R	b1	b2

Histogram example

$$Gini(L) = 1 - \sum j = 1 \ldots n\, P^2 j \qquad (1)$$

- pj is the relative frequency of j.

After the binary split of the set L into sets L1 and L2 the index becomes,

$$Ginisplit(L) = \frac{|L1|}{|L|} Gini(L1) + \frac{|L2|}{|L|} Gini(L2). \qquad (2)$$

Lastly, the Gini Index equation is with data classes a and b is

$$Gini\ Index = \frac{a1+a2}{n}\left[1-\left(\frac{a1}{a1+a2}\right)^2-\left(\frac{a2}{a1+a2}\right)^2\right]+ \frac{b1+b2}{n}\left[1-\left(\frac{b1}{b1+b2}\right)^2-\left(\frac{b2}{b1+b2}\right)^2\right] \ (5)$$

(3). These mathematical formulations are what sets SLIQ algorithms apart from traditional decision tree algorithms.

it can handle a variety of data types across many features and classifications. The downsides to SLIQ lie in its complexity, time, and cost. SLIQ can very quickly become very complex for large data sets, requires more time to train the model, is more expensive and complex than a normal decision tree algorithm, and cannot be applied to regressions or predictive modeling [2]. The application of SLIQ can be used in any field where data mining is prevalent. Specifically, SLIQ is being used today in the deregulated power market. The SLIQ algorithm allows us to mine data in terms of cost of energy for purchase and sale to meet load demands and decrease the cost of energy usage across any industry [4]. One real life application is Bidding Decision System of Electricity Market. The SLIQ algorithm is applied to the bidding decision system of the electricity market, where the knowledge of the bidding unit's ability is gained by considering the market's demand, bidding price, and capacity [3].

Existing Work

Throughout the research of SLIQ, researchers found that there is a lack of studies about the algorithm but were unable to find it's first inventors. From the IEEE Xplore library, researchers were able to find a paper from 1996 written by three engineers: Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. These engineers proposed that SLIQ could solve an important problem at the time, data mining. The algorithm for SLIQ creates a decision tree that can handle both numeric and categorical attributes, which uses presorting techniques and optimizations to create the ideal results[4]. These engineers suggested that one could use the algorithm to create inexpensive, compact, and accurate trees.

Within the first research paper reviewed, the researchers reviewed previous studies conducted on classification, but found that for large data sets, they don't scale well. As a solution, they proposed a decision- tree classifier, SLIQ, designed specifically for scalability.

Within the second research paper reviewed, researchers found that the prediction with the greatest separating power correlates to a split in a decision tree. The optimal split creates nodes where a single class dominates the most[5]. The predictor's power to separate data may be calculated in a variety of ways. The Gini coefficient of inequality is one of the most well-known methodologies.

Within the second research paper reviewed, the researchers used data mining for an easy tool to analyze historical rainfall data, it allowed them to measure valuable patterns within a short period of time. With an average accuracy of 74.92 percent, the SLIQ decision tree algorithm was able to estimate an accurate precipitation forecast[6].

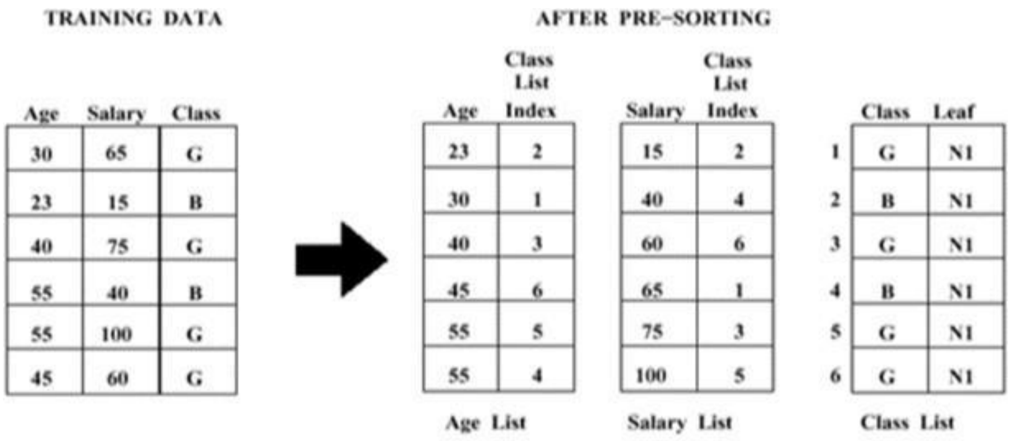
In 2005, through the International Power Engineering Conference, three engineers, named Hongwen Yan, Rui Ma, and Xiaojiao Tong proposed using SLIQ to build a framework for a competitive bidding assessment in a deregulated power market. They suggested that the bidding system using the SLIQ algorithm could be consistent with the features of the electric energy production and consummation, this would be more convenient for operating the power markets[7].

Throughout the research of this algorithm, researchers only found software implementations. Where SLIQ is used to make decisions based on a set of data for a specific use. Of these software implementations, there are example algorithms of precipitation prediction, bidding, or sorting data. There are also algorithms that identify households that are most likely to respond to a promotion of a product, such as a new banking service[5]. Researchers have found that the SLIQ decision tree, superior to other algorithms, can be built fast and scalable for larger data sets.

Example

- First Example

The question of first example involved with two continuous attributes, the age and salary, to create the decision tree classifier. In this example, the SLIQ Algorithm is divided into 3 steps, pre-sorting the sample, processing evaluation on splits, and updating the class list [4].



Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

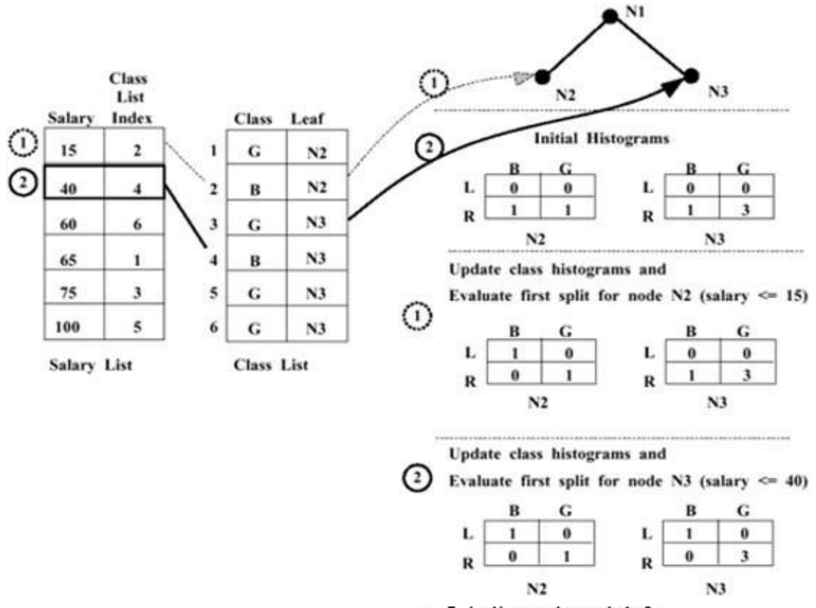
Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

	Class	Leaf
1	G	N1
2	B	N1
3	G	N1
4	B	N1
5	G	N1
6	G	N1

Pre-sorting the sample with given training data, create an attribute list for each attribute.

Once the data is sorted, it can process evaluation on splits by assuming the split of the first node and evaluate each histogram. (Gini index will be used during this step but for this example since attribute, age and salary do not have predictive power since they can be any number. Finally, the class list can be updated. Traverse the training data through the decision tree and replace the node with the new node.


N1
N2
N3
1
2
Salary List
Class List
Initial Histograms
Update class histograms and
Evaluate first split for node N2 (salary <= 15)
Update class histograms and
Evaluate first split for node N3 (salary <= 40)

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

	Class	Leaf
1	G	N2
2	B	N2
3	G	N3
4	B	N3
5	G	N3
6	G	N3

Initial Histograms

N2	B	G	N3	B	G
L	0	0	L	0	0
R	1	1	R	1	3

Evaluate first split for node N2 (salary <= 15)

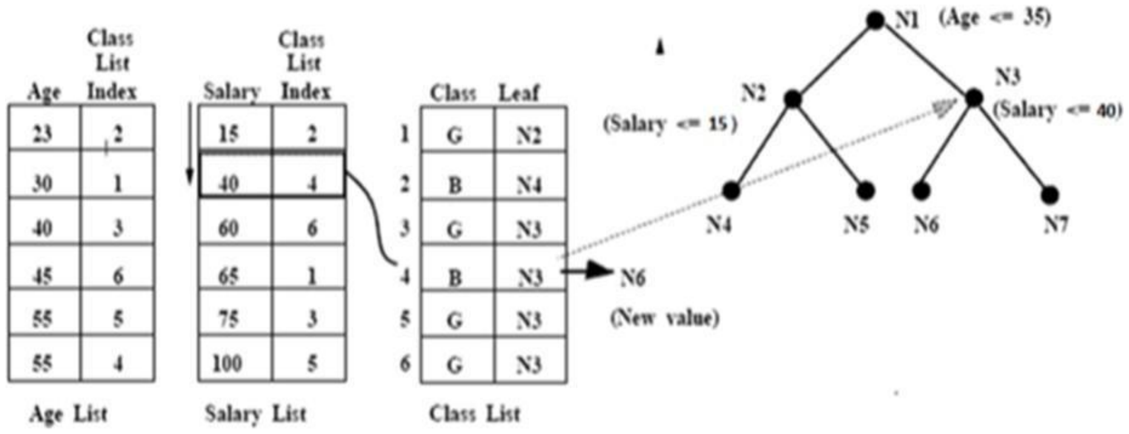
N2	B	G	N3	B	G
L	1	0	L	0	0
R	0	1	R	1	3

Evaluate first split for node N3 (salary <= 40)

N2	B	G	N3	B	G
L	1	0	L	1	0
R	0	1	R	0	3

- L indicates branch left.
- R indicates branch right.

Then the final decision tree classifier gets created with the final classification according to the previous steps and calculations.



Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

Age List

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

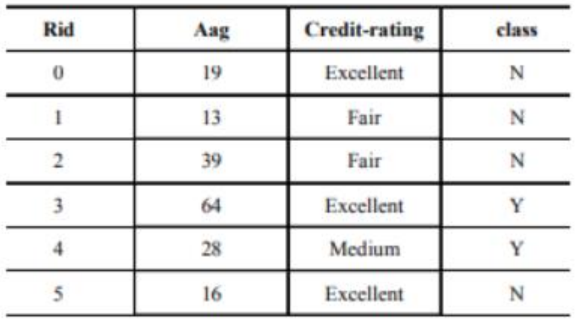
Salary List

	Class	Leaf
1	G	N2
2	B	N4
3	G	N3
4	B	N3
5	G	N3
6	G	N3

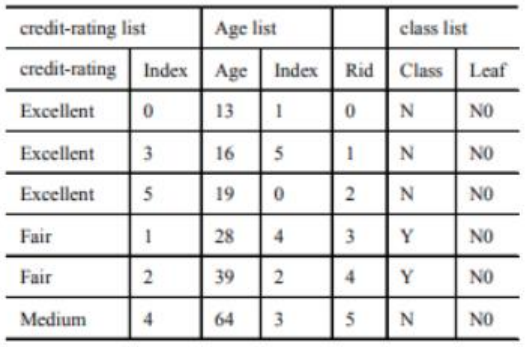
Class List

- Second Example

The question of second example involved with a continuous attribute and a categorical attribute, the age and credit rating, to create the decision tree classifier. Similar to the first example, the SLIQ Algorithm is divided into 3 steps, pre-sorting the sample, processing evaluation on splits, and updating the class list [4].

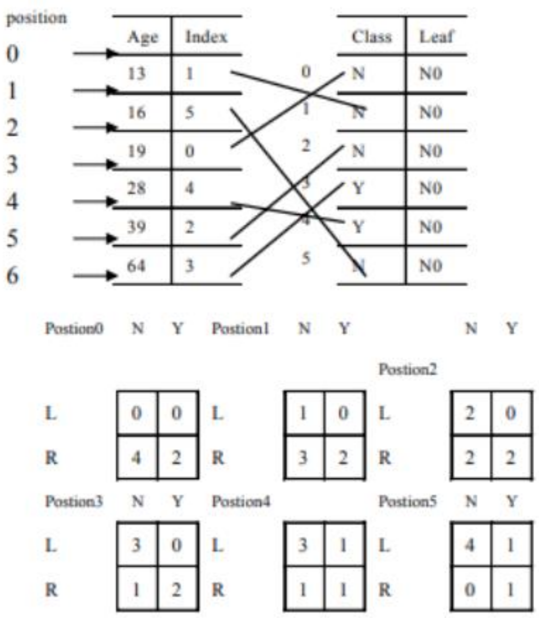


Rid	Aag	Credit-rating	class
0	19	Excellent	N
1	13	Fair	N
2	39	Fair	N
3	64	Excellent	Y
4	28	Medium	Y
5	16	Excellent	N



credit-rating list		Age list			class list	
credit-rating	Index	Age	Index	Rid	Class	Leaf
Excellent	0	13	1	0	N	N0
Excellent	3	16	5	1	N	N0
Excellent	5	19	0	2	N	N0
Fair	1	28	4	3	Y	N0
Fair	2	39	2	4	Y	N0
Medium	4	64	3	5	N	N0

First, the given training data set gets presorted then using the Gini index for the best splitting value is determined. For the categorical attribute acts similar to the discrete attribute, however, it has more than two values. In this example they are either excellent, fair, or medium. Using those information the Gini index can be calculate similar to how the Gini index for splitting value is calculated for continuous attribute. Finally, each nodes can be updated and create the decision tree classifier.



position	Age	Index
0	13	1
1	16	5
2	19	0
3	28	4
4	39	2
5	64	3
6		

	Class	Leaf
0	N	N0
1	N	N0
2	N	N0
3	Y	N0
4	Y	N0
5	N	N0

Postion0	N	Y
L	0	0
R	4	2

Postion1	N	Y
L	1	0
R	3	2

Postion2	N	Y
L	2	0
R	2	2

Postion3	N	Y
L	3	0
R	1	2

Postion4		
L	3	1
R	1	1

Postion5	N	Y
L	4	1
R	0	1

S1: $Gini(s_1 < 14.5) = 1 - (\frac{1}{1})^2 = 0$

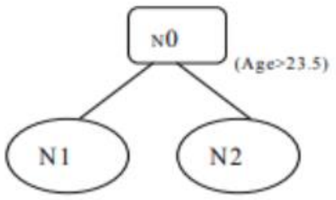
S2: $Gini(s_2 > 14.5) = 1 - ((\frac{3}{5})^2 + (\frac{2}{5})^2) = 0.48$

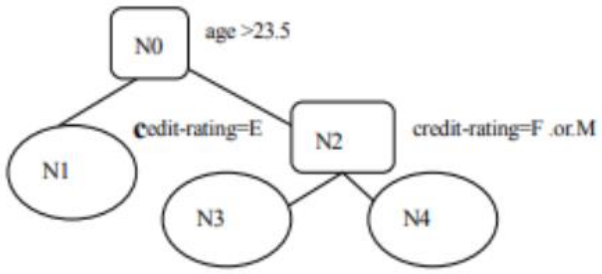
$Gini\_split(14.5) = \frac{1}{6} \times 0 + \frac{5}{6} \times 0.48 = 0.4$

In the same way, $Gini\_split(17.5) = 0.33$

$Gini\_split(23.5) = 0.22$; $Gini\_split(33.5) = 0.42$

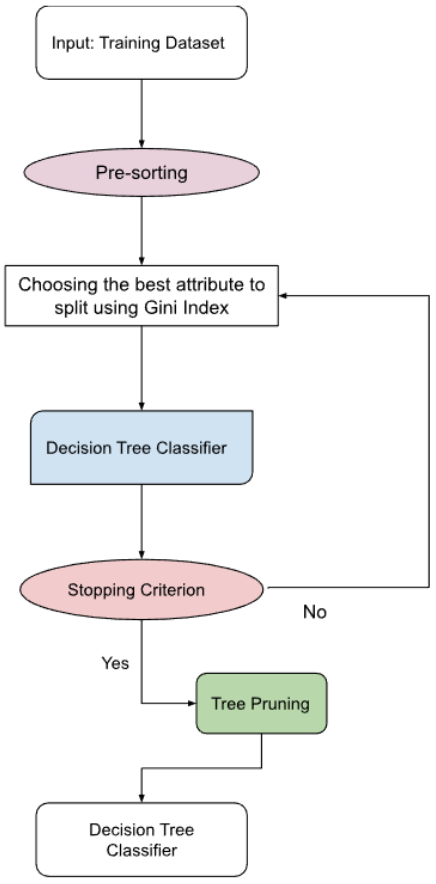
	Class	Leaf
0	N	N1
1	N	N1
2	N	N2
3	Y	N2
4	Y	N2
5	N	N1





Methodology – Python3

The primary goal of this example is to determine the loan approval status by utilizing the given attributes in the training dataset. A loan approval can be influenced by various factors, and the project's training data has seven key attributes: Married Status, Dependents Existence, Education Level, Self Employment Status, Applicant Income, Co-applicant Income, and Loan Amount. These attributes are carefully analyzed to establish the loan approval status. The training dataset consists of 614 examples, each with a loan approval status.

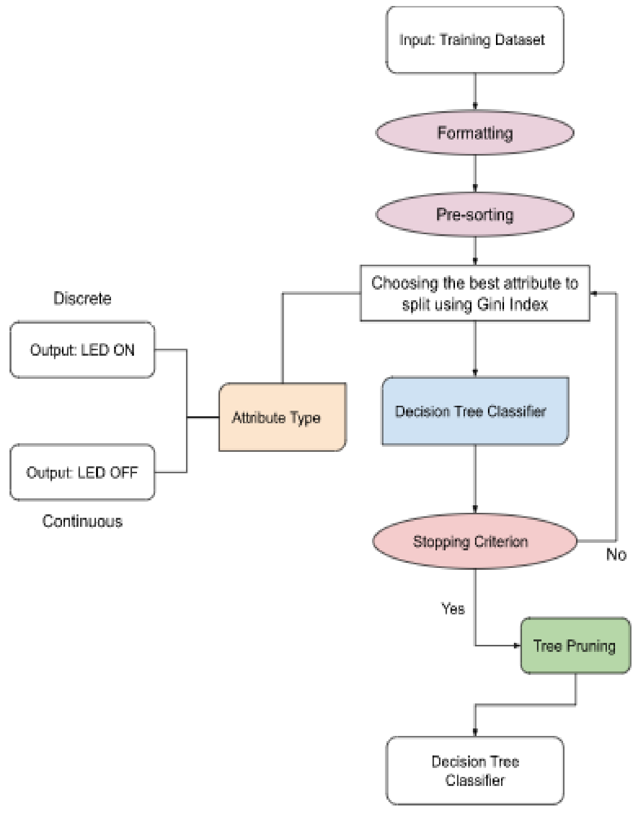


As shown in the Data Flow Graph, many different calculations are required to form the decision tree classifier, and the main repeated calculation is the Gini index. By using the Gini index equation from the introduction section of the paper, the best attribute for decision tree classifier splitting can be determined and updated. These steps continue until the stopping criterion is met. The stopping criterion is a condition that determines when the splitting of the dataset should stop, and when a leaf node of the decision tree should be created. Common stopping criteria include when no more attributes are available for splitting, when all instances in the node belong to the same class, when the number of instances in the node falls below a certain threshold, or when the depth of the decision tree reaches a certain limit. The choice of stopping criterion is important to balance the accuracy and complexity of the decision tree. If the stopping criterion is too strict, the decision tree may be too simple and inaccurate, while if it is too relaxed, the decision tree may be too complex and overfit to the training data. Then the dataset can be applied to find the loan approval status.

One limitation that can be pointed out is when continuous attributes are present within the dataset. In order to handle this, SLIQ uses presorting for the continuous attributes. The dataset being used for the project has several continuous attributes, but the Python code implemented for the project did not use presorting. This can be done either in Excel or in the code to see the difference between the presorted decision tree and the non-presorted decision tree. By presorting the data based on the attribute values, the algorithm can quickly identify the best split points and build an accurate decision tree. Which can significantly speed up the runtime because it reduces the number of comparisons needed to find the best split points. In addition, presorting can also improve the accuracy of the decision tree.

Methodology – Microcontroller Implementation using Micropython

For implementation on the microcontroller, Raspberry Pi Pico, Micropython was necessary to be used. However, due to the lack of memory space of the microcontroller, the training data needed to be reduced to 50 examples from 614 examples. However, it was possible to keep all seven key attributes: Married Status, Dependents Existence, Education Level, Self Employment Status, Applicant Income, Co-applicant Income, and Loan Amount. Micropython had limited library that can be used, therefore, the full implementation of creating the decision tree classifier was challenging. However, without the library, importing, formatting, and pre-sorting the training data set as well as calculating the Gini index were possible within the microcontroller. Using the Gini index that were calculated, the best splitting attribute or the best splitting value was able to be found. Farther more using the Raspberry Pi Pico, it was possible to determine the types of the attributes and display the LED within the microcontroller.



Data Flow Graph

Procedure

For this project, some Python libraries were required to be imported into the Jupyter Notebook to bring in the datasets and create the decision tree classifier. The two main libraries imported in the project were Pandas and Scikit-learn (sklearn).

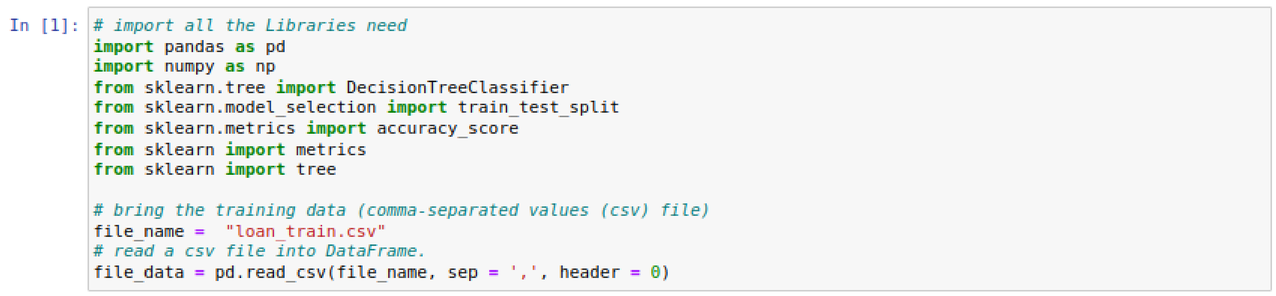
Pandas is a Python library designed for data manipulation and analysis. It provides easy-to-use data structures for working with structured data like CSV, Excel, SQL databases, and more. One main data structure in Pandas that this project mainly used is DataFrame. DataFrame is a two-dimensional labeled data structure that can hold data of different types in columns.

Scikit-learn, or sklearn, is a Python library for machine learning that provides a wide range of functions and tools for data preprocessing, model selection, and model evaluation. In the project, several functions from the sklearn library were utilized, including the DecisionTreeClassifier class, which implements the decision tree algorithm for classification, the train_test_split function, which splits a dataset into training and testing subsets, the accuracy_score function, which computes the accuracy of a classification model, the metrics module, which provides various metrics for evaluating machine learning models, and the tree module, which provides tools for working with decision trees.

Result – Python3

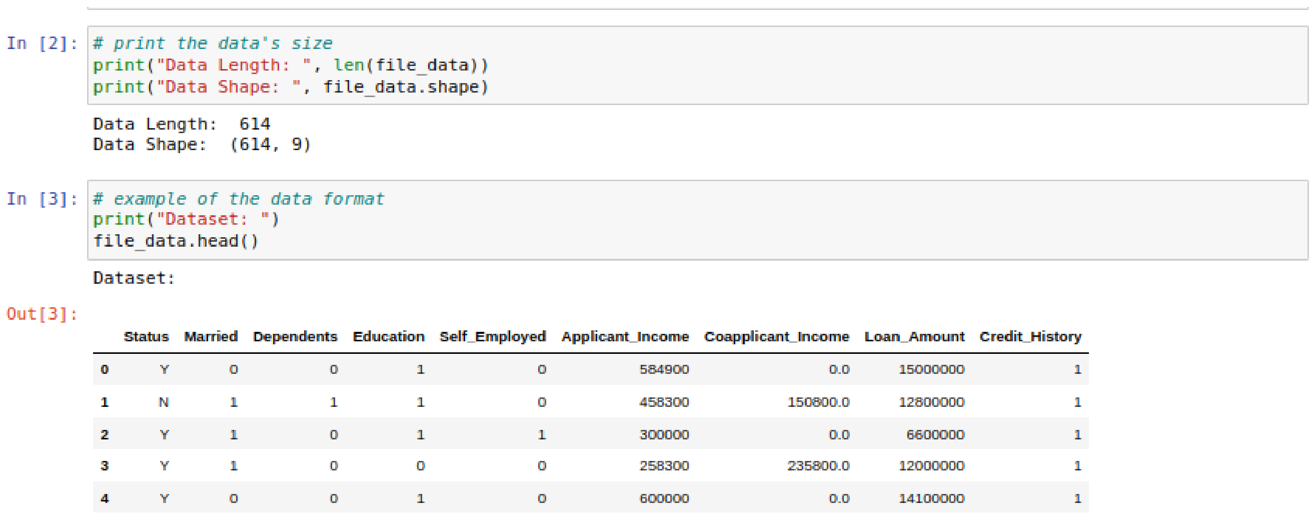
Using all the knowledge gathered, the Supervised Learning in Quest, or SLIQ, machine learning algorithm was able to be implement into Python on the Jupyter Notebook.

Importing all the libraries mentioned in the procedure section of the paper as well as bring the dataset to train the decision tree classifier.



```
In [1]: # import all the Libraries need
        import pandas as pd
        import numpy as np
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        from sklearn import metrics
        from sklearn import tree

        # bring the training data (comma-separated values (csv) file)
        file_name =  "loan_train.csv"
        # read a csv file into DataFrame.
        file_data = pd.read_csv(file_name, sep = ',', header = 0)
```

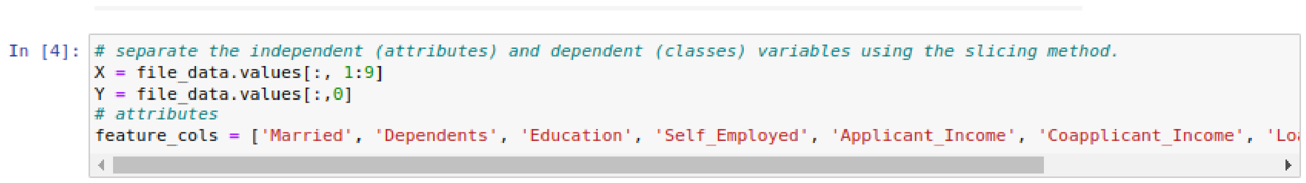
Outputing the size of the traing dataset and example of the dataset.



```
In [2]: # print the data's size
        print("Data Length: ", len(file_data))
        print("Data Shape: ", file_data.shape)

        Data Length:  614
        Data Shape:  (614, 9)

In [3]: # example of the data format
        print("Dataset: ")
        file_data.head()

        Dataset:
Out[3]:
```

	Status	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_Amount	Credit_History
0	Y	0	0	1	0	584900	0.0	15000000	1
1	N	1	1	1	0	458300	150800.0	12800000	1
2	Y	1	0	1	1	300000	0.0	6600000	1
3	Y	1	0	0	0	258300	235800.0	12000000	1
4	Y	0	0	1	0	600000	0.0	14100000	1

Separating the training dataset into the attributes and classes.



```
In [4]: # separate the independent (attributes) and dependent (classes) variables using the slicing method.
        X = file_data.values[:, 1:9]
        Y = file_data.values[:,0]
        # attributes
        feature_cols = ['Married', 'Dependents', 'Education', 'Self_Employed', 'Applicant_Income', 'Coapplicant_Income', 'Lo
```

Splitting the training dataset in order to test the decision tree created.

```
In [5]: # 70% training and 30% test, for accuracy test for the classifier created from the trained data
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

        # Create Decision Tree classifer object using gini criterion
        clf = DecisionTreeClassifier(criterion = "gini")

        # Train Decision Tree Classifer using the 70% of the training data
        clf = clf.fit(X_train,y_train)

        #Predict the response for the test dataset, 30% of the training data
        y_pred = clf.predict(X_test)
```

```
In [6]: # prediction of the test
        print(y_pred)
```

```
['N' 'Y' 'N' 'N' 'Y' 'N' 'N' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'N'
 'N' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'N' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y'
 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'N' 'N'
 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'N' 'N' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y'
 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N'
 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'N']
```

```
In [7]: # find the accuracy of training data set
        print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.654054054054054
```

Bring the dataset that needs to be tested for the loan approval status.

```
In [8]: # bring the file for Loan Approval prediction
        app_file_name =  "loan_app.csv"
        # read a csv file into DataFrame.
        app_file_data = pd.read_csv(app_file_name, sep = ',', header = 0)
        # print the data information
        print("Data Length: ", len(app_file_data))
        print("Data Shape: ", app_file_data.shape)
        print("Dataset: ")
        app_file_data.head()
```

```
Data Length:  367
Data Shape:  (367, 8)
Dataset:
```

Out[8]:

	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_Amount	Credit_History
0	1	0	1	0	572000	0	11000000	1
1	1	1	1	0	307600	150000	12600000	1
2	1	1	1	0	500000	180000	20800000	1
3	1	1	1	0	234000	254600	10000000	0
4	0	0	0	0	327600	0	7800000	1

Testing and outputting the results of the dataset.

```
In [9]: app_X = app_file_data.values[:, 0:8]
```

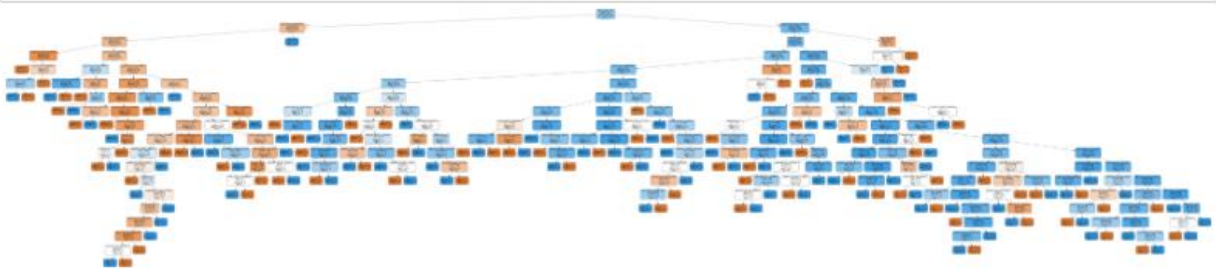
```
In [10]: # Create Decision Tree classifer object using gini criterion
         clf_app = DecisionTreeClassifier(criterion = "gini")
         # Train Decision Tree Classifer using all the training data
         clf_app = clf_app.fit(X,Y)

         #Predict the response for Loan Approval file
         y_pred_app = clf_app.predict(app_X)
         print(y_pred_app)
```

```
['Y' 'Y' 'Y' 'N' 'N' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'N' 'N' 'Y' 'Y' 'Y' 'Y'
 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'N'
 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'N'
 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'N'
 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'Y'
 'N' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'N' 'Y' 'Y' 'Y'
 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'N'
 'Y' 'N' 'N' 'N' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'N' 'N' 'N'
 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'N' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'N' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'N'
 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'N' 'N' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'N' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'N'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'Y' 'N'
 'N' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'N' 'Y' 'Y' 'Y' 'N' 'N'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y']
```

Visualizing the decision tree that is created.

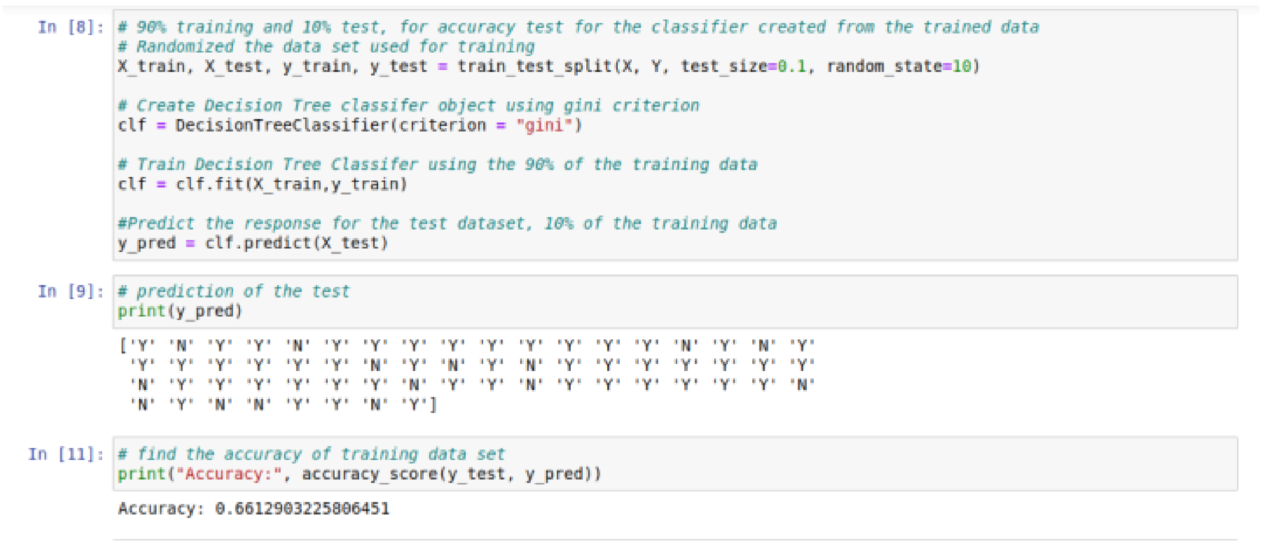
```
In [11]: # save and print the clf created for the loan approval

         from six import StringIO
         from sklearn.tree import export_graphviz
         from IPython.display import Image
         import pydotplus
         import graphviz
         import os

         dot_data = StringIO()
         export_graphviz(clf_app, out_file = dot_data,
                         filled = True, rounded = True,
                         special_characters = True,
                         feature_names = feature_cols,
                         class_names=['0','1'])
         graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         graph.write_png('clf_app.png')
         Image(graph.create_png())
```

Out[11]:



Result – Python3 Alternative Solution

Alternatively, when training the decision tree classifier with the training data, it could have used more percentage of the training data and randomizing the selection to get different accuracy for the decision tree classifier.



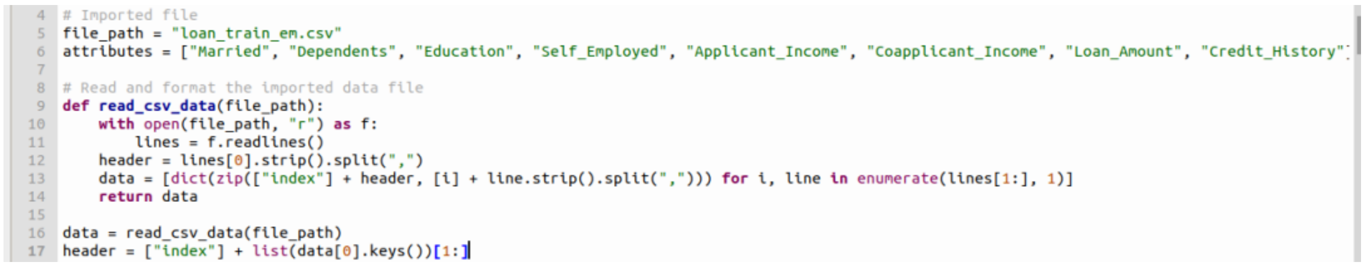
```
In [8]: # 90% training and 10% test, for accuracy test for the classifier created from the trained data
        # Randomized the data set used for training
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=10)

        # Create Decision Tree classifer object using gini criterion
        clf = DecisionTreeClassifier(criterion = "gini")

        # Train Decision Tree Classifer using the 90% of the training data
        clf = clf.fit(X_train,y_train)

        #Predict the response for the test dataset, 10% of the training data
        y_pred = clf.predict(X_test)

In [9]: # prediction of the test
        print(y_pred)

        ['Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y'
         'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y'
         'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'N'
         'N' 'Y' 'N' 'N' 'Y' 'Y' 'N' 'Y']

In [11]: # find the accuracy of training data set
         print("Accuracy:", accuracy_score(y_test, y_pred))

         Accuracy: 0.6612903225806451
```

The accuracy of the decision tree classifier does not change significantly, but if there were more training data examples then this accuracy can be improved significantly, resulting in the more accurate decision making when the data set without the classes are inputted.

Result – Microcontroller Implementation using Micropython

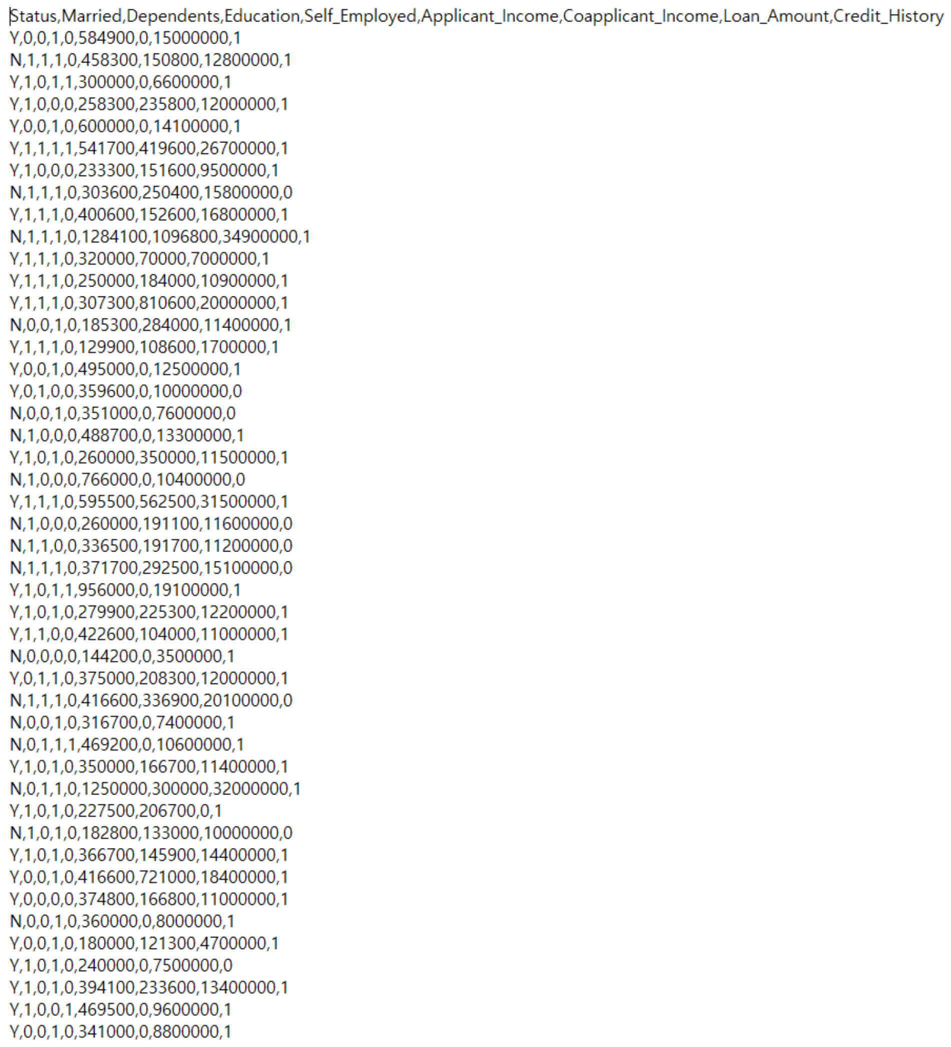
Micropython does not support much libraries like Python3 does so in order for the imported data to be pre-sorted, it needs to be formatted correctly then the attribute values need to be iterated through and compared each with each.

Importing and formatting the data set.



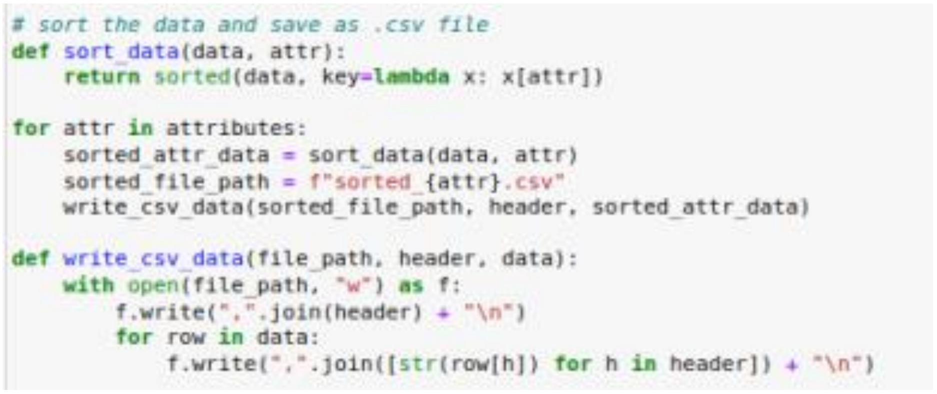
```
# Imported file
file_path = "loan_train_em.csv"
attributes = ["Married", "Dependents", "Education", "Self_Employed", "Applicant_Income", "Coapplicant_Income", "Loan_Amount", "Credit_History"

# Read and format the imported data file
def read_csv_data(file_path):
    with open(file_path, "r") as f:
        lines = f.readlines()
    header = lines[0].strip().split(",")
    data = [dict(zip(["index"] + header, [i] + line.strip().split(","))) for i, line in enumerate(lines[1:], 1)]
    return data

data = read_csv_data(file_path)
header = ["index"] + list(data[0].keys())[1:]
```

Formatted data set.



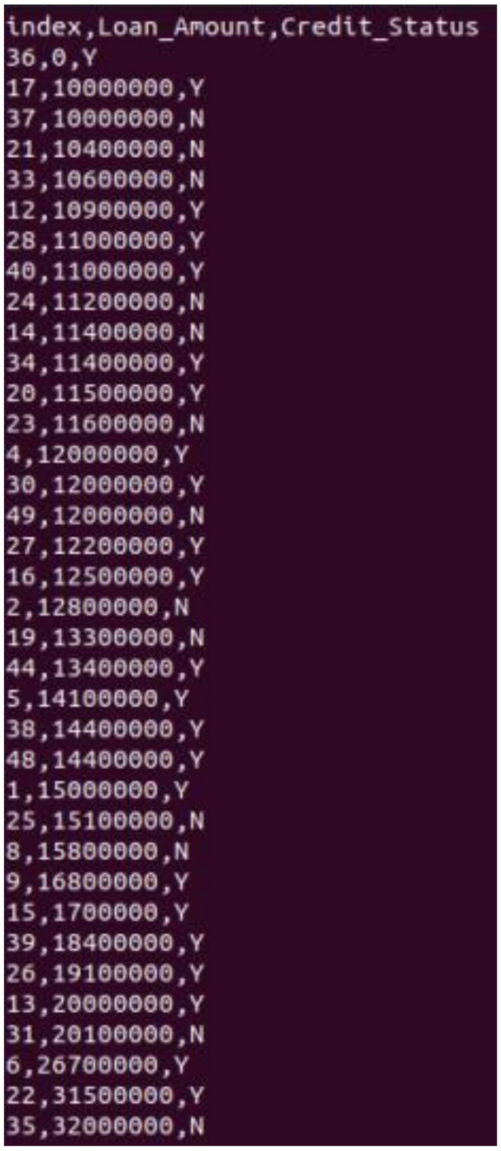
```
Status,Married,Dependents,Education,Self_Employed,Applicant_Income,Coapplicant_Income,Loan_Amount,Credit_History
Y,0,0,1,0,584900,0,15000000,1
N,1,1,1,0,458300,150800,12800000,1
Y,1,0,1,1,300000,0,6600000,1
Y,1,0,0,0,258300,235800,12000000,1
Y,0,0,1,0,600000,0,14100000,1
Y,1,1,1,1,541700,419600,26700000,1
Y,1,0,0,0,233300,151600,9500000,1
N,1,1,1,0,303600,250400,15800000,0
Y,1,1,1,0,400600,152600,16800000,1
N,1,1,1,0,1284100,1096800,34900000,1
Y,1,1,1,0,320000,70000,7000000,1
Y,1,1,1,0,250000,184000,10900000,1
Y,1,1,1,0,307300,810600,20000000,1
N,0,0,1,0,185300,284000,11400000,1
Y,1,1,1,0,129900,108600,1700000,1
Y,0,0,1,0,495000,0,12500000,1
Y,0,1,0,0,359600,0,10000000,0
N,0,0,1,0,351000,0,7600000,0
N,1,0,0,0,488700,0,13300000,1
Y,1,0,1,0,260000,350000,11500000,1
N,1,0,0,0,766000,0,10400000,0
Y,1,1,1,0,595500,562500,31500000,1
N,1,0,0,0,260000,191100,11600000,0
N,1,1,0,0,336500,191700,11200000,0
N,1,1,1,0,371700,292500,15100000,0
Y,1,0,1,1,956000,0,19100000,1
Y,1,0,1,0,279900,225300,12200000,1
Y,1,1,0,0,422600,104000,11000000,1
N,0,0,0,0,144200,0,3500000,1
Y,0,1,1,0,375000,208300,12000000,1
N,1,1,1,0,416600,336900,20100000,0
N,0,0,1,0,316700,0,7400000,1
N,0,1,1,1,469200,0,10600000,1
Y,1,0,1,0,350000,166700,11400000,1
N,0,1,1,0,1250000,300000,32000000,1
Y,1,0,1,0,227500,206700,0,1
N,1,0,1,0,182800,133000,10000000,0
Y,1,0,1,0,366700,145900,14400000,1
Y,0,0,1,0,416600,721000,18400000,1
Y,0,0,0,0,374800,166800,11000000,1
N,0,0,1,0,360000,0,8000000,1
Y,0,0,1,0,180000,121300,4700000,1
Y,1,0,1,0,240000,0,7500000,0
Y,1,0,1,0,394100,233600,13400000,1
Y,1,0,0,1,469500,0,9600000,1
Y,0,0,1,0,341000,0,8800000,1
```

Pre-sorting the data set.



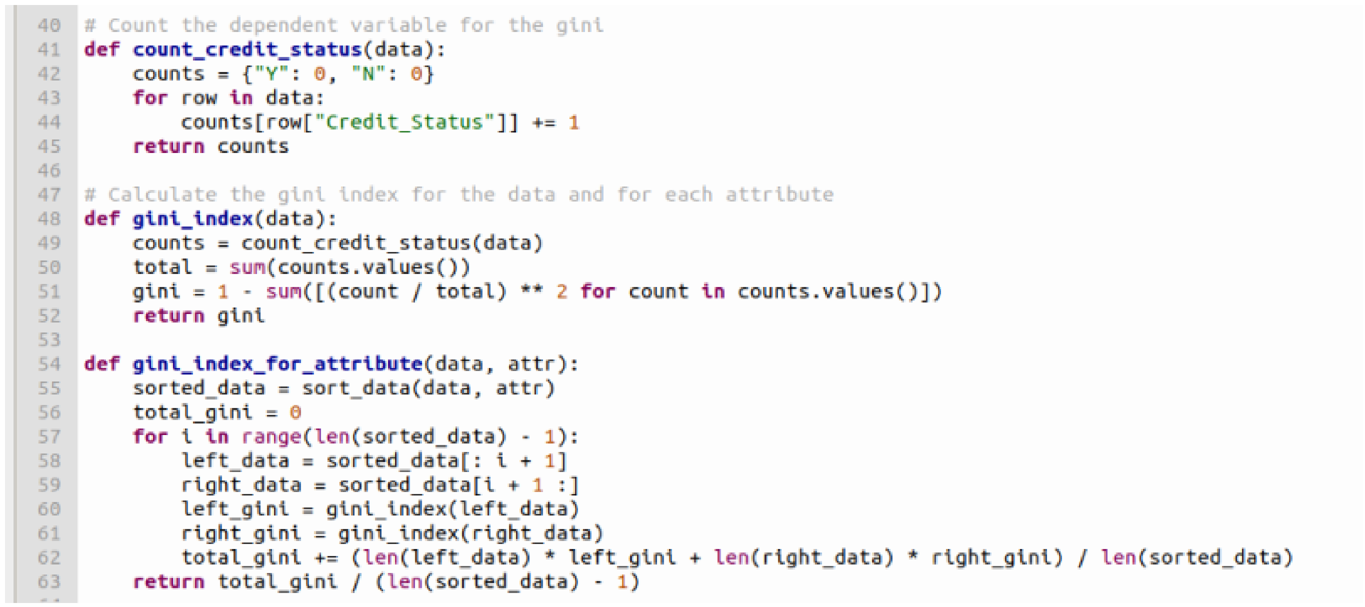
```python
# sort the data and save as .csv file
def sort_data(data, attr):
    return sorted(data, key=lambda x: x[attr])

for attr in attributes:
    sorted_attr_data = sort_data(data, attr)
    sorted_file_path = f"sorted_{attr}.csv"
    write_csv_data(sorted_file_path, header, sorted_attr_data)

def write_csv_data(file_path, header, data):
    with open(file_path, "w") as f:
        f.write(",".join(header) + "\n")
        for row in data:
            f.write(",".join([str(row[h]) for h in header]) + "\n")
```

Pre-sorted data set



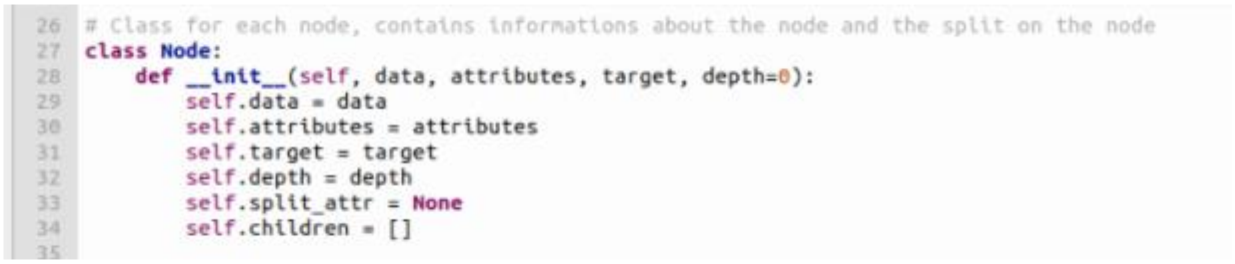
```
index,Loan_Amount,Credit_Status
36,0,Y
17,10000000,Y
37,10000000,N
21,10400000,N
33,10600000,N
12,10900000,Y
28,11000000,Y
40,11000000,Y
24,11200000,N
14,11400000,N
34,11400000,Y
20,11500000,Y
23,11600000,N
4,12000000,Y
30,12000000,Y
49,12000000,N
27,12200000,Y
16,12500000,Y
2,12800000,N
19,13300000,N
44,13400000,Y
5,14100000,Y
38,14400000,Y
48,14400000,Y
1,15000000,Y
25,15100000,N
8,15800000,N
9,16800000,Y
15,1700000,Y
39,18400000,Y
26,19100000,Y
13,20000000,Y
31,20100000,N
6,26700000,Y
22,31500000,Y
35,32000000,N
```

Calculating the Gini index.

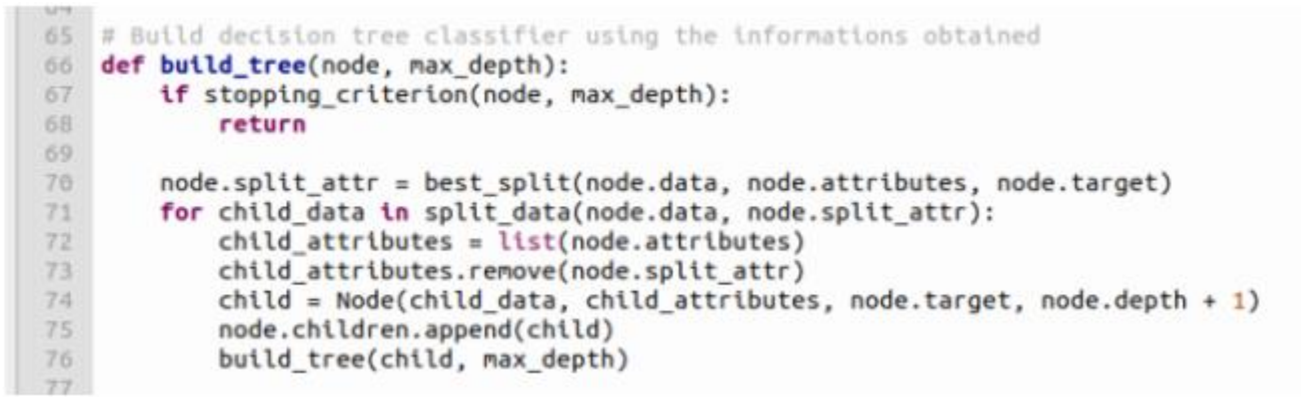


```python
# Count the dependent variable for the gini
def count_credit_status(data):
    counts = {"Y": 0, "N": 0}
    for row in data:
        counts[row["Credit_Status"]] += 1
    return counts

# Calculate the gini index for the data and for each attribute
def gini_index(data):
    counts = count_credit_status(data)
    total = sum(counts.values())
    gini = 1 - sum([(count / total) ** 2 for count in counts.values()])
    return gini

def gini_index_for_attribute(data, attr):
    sorted_data = sort_data(data, attr)
    total_gini = 0
    for i in range(len(sorted_data) - 1):
        left_data = sorted_data[: i + 1]
        right_data = sorted_data[i + 1 :]
        left_gini = gini_index(left_data)
        right_gini = gini_index(right_data)
        total_gini += (len(left_data) * left_gini + len(right_data) * right_gini) / len(sorted_data)
    return total_gini / (len(sorted_data) - 1)
```

Attempt to build the decision tree classifier.

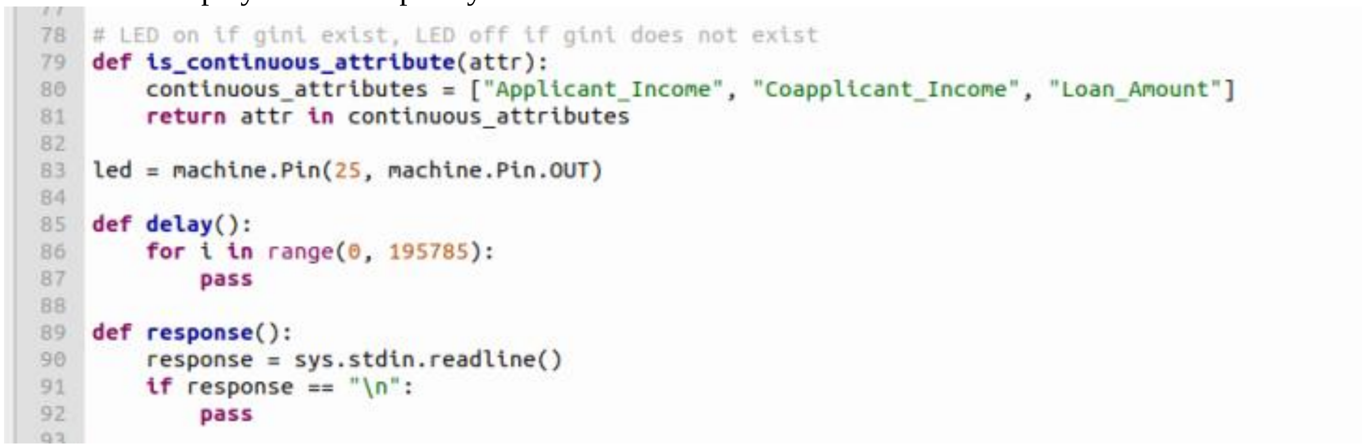


```python
# Class for each node, contains informations about the node and the split on the node
class Node:
    def __init__(self, data, attributes, target, depth=0):
        self.data = data
        self.attributes = attributes
        self.target = target
        self.depth = depth
        self.split_attr = None
        self.children = []
```

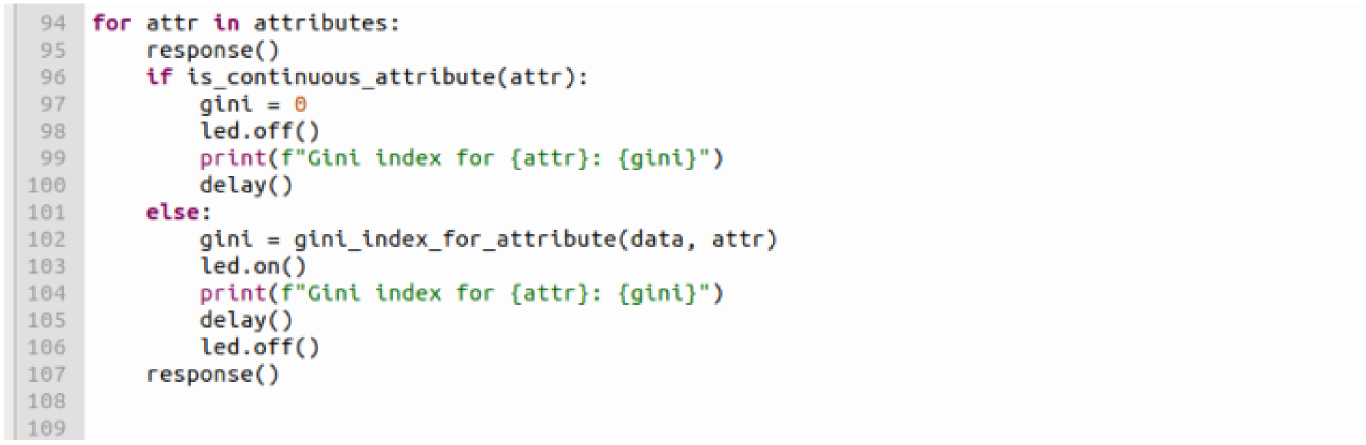


```python
# Build decision tree classifier using the informations obtained
def build_tree(node, max_depth):
    if stopping_criterion(node, max_depth):
        return

    node.split_attr = best_split(node.data, node.attributes, node.target)
    for child_data in split_data(node.data, node.split_attr):
        child_attributes = list(node.attributes)
        child_attributes.remove(node.split_attr)
        child = Node(child_data, child_attributes, node.target, node.depth + 1)
        node.children.append(child)
        build_tree(child, max_depth)
```

LED display on the Raspberry Pi Pico Microcontroller.



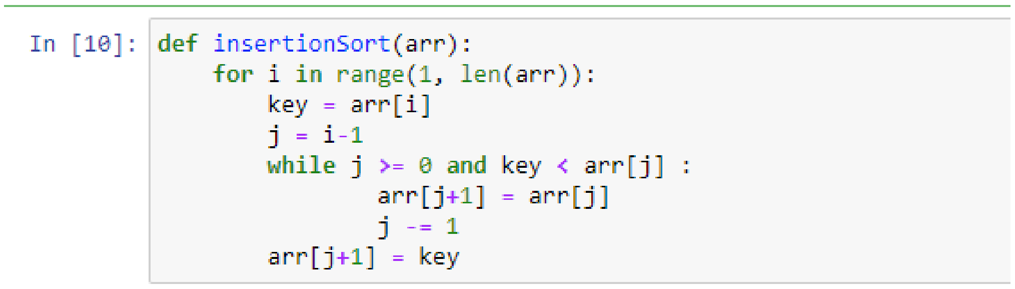
```
# LED on if gini exist, LED off if gini does not exist
def is_continuous_attribute(attr):
    continuous_attributes = ["Applicant_Income", "Coapplicant_Income", "Loan_Amount"]
    return attr in continuous_attributes

led = machine.Pin(25, machine.Pin.OUT)

def delay():
    for i in range(0, 195785):
        pass

def response():
    response = sys.stdin.readline()
    if response == "\n":
        pass
```



```
for attr in attributes:
    response()
    if is_continuous_attribute(attr):
        gini = 0
        led.off()
        print(f"Gini index for {attr}: {gini}")
        delay()
    else:
        gini = gini_index_for_attribute(data, attr)
        led.on()
        print(f"Gini index for {attr}: {gini}")
        delay()
        led.off()
    response()
```

Result – Micropython Alternative Solution

When sorting the data set, there are many different ways of sorting the data. This is one of the different ways of sorting. In the alternative solution, each of the attribute values are getting compared. This is similar to the code that was used on the project. However, the code is more extended. Which will result in the longer running time for this code to be execute.



```
In [10]: def insertionSort(arr):
             for i in range(1, len(arr)):
                 key = arr[i]
                 j = i-1
                 while j >= 0 and key < arr[j] :
                         arr[j+1] = arr[j]
                         j -= 1
                 arr[j+1] = key
```

Conclusion

The paper explored the potential of the Supervised Learning In Quest (SLIQ) algorithm for creating decision tree classifiers, specifically for loan approval prediction. The study aimed to implement the algorithm using Micropython and Python3, analyze its performance, and compare the effectiveness and efficiency of both programming languages for implementing the algorithm. The results showed that the SLIQ algorithm can be effectively used for creating decision tree classifiers with high accuracy rates for loan approval prediction. However, the implementation of the algorithm in Micropython was limited compared to Python3. Specifically, in Micropython, only the Gini index of the first node of the classifier could be calculated, whereas using Python3, the entire decision tree classifier could be created. The paper also highlighted the advantages and disadvantages of the SLIQ algorithm. One of the most significant advantages of SLIQ is its ability to handle different data types and scale well with large datasets, making it an ideal machine learning technique for data mining. Additionally, SLIQ eliminates the need for data normalization, which is a typical preprocessing step in many other machine learning methods. However, the algorithm's complexity and time constraints remain a significant challenge. SLIQ can quickly become complex for large datasets and requires more time and resources to train the model compared to ordinary decision tree algorithms. The paper contributes to a better understanding of the SLIQ algorithm and its implementation in different programming languages for creating decision tree classifiers. The results of this study can be applied in various fields where data mining is prevalent, such as the deregulated power market. Furthermore, SLIQ can also be applied to other applications, such as the bidding decision system of electricity markets, where it allows the knowledge gained from considering a market's demand, bidding price, and capacity to mine data in terms of the cost of energy purchase and sale to meet load demands and decrease energy usage costs across any industry.

Overall, the findings of this study demonstrate that the SLIQ algorithm is a highly effective machine learning technique for creating decision tree classifiers. Python3 is a more efficient and effective platform than Micropython for implementing the algorithm. The research also highlights the potential advantages and challenges of using the SLIQ algorithm for data mining and the various fields where it can be applied.

References

[1] Huacheng Zhang, Wu Xie, "Improvement of SLIQ Algorithm and its Application in Evaluation", Genetic and Evolutionary Computing 2009. WGEC '09. 3rd International Conference on, pp. 77-80, 2009.

[2] Xie Wu, Huacheng Zhang, Huimin Zhang, "Study of comprehensive evaluation method of undergraduates based on data mining", Intelligent Computing and Integrated Systems (ICISS) 2010 International Conference on, pp. 541-543, 2010.

[3] Hongwen Yan, Rui Ma and Xiaojiao Tong, "SLIQ in data mining and application in the generation unit's bidding decision system of electricity market," 2005 International Power Engineering

[4] Mehta, M., Agrawal, R., Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In: Apers, P., Bouzeghoub, M., Gardarin, G. (eds) Advances in Database Technology — EDBT '96. EDBT 1996. Lecture Notes in Computer Science, vol 1057. Springer, Berlin, Heidelberg.

[5] S. Sivagama Sundhari, "A knowledge discovery using decision tree by Gini coefficient," 2011 International Conference on Business, Engineering and Industrial Applications, 2011, pp. 232-235.

[6] N. Prasad, P. K. Reddy and M. M. Naidu, "A Novel Decision Tree Approach for the Prediction of Precipitation Using Entropy in SLIQ," 2013 UKSim 15th International Conference on Computer Modelling and Simulation, 2013, pp. 209-217.

[7] Hongwen Yan, Rui Ma and Xiaojiao Tong, "SLIQ in data mining and application in the generation unit's bidding decision system of electricity market," 2005 International Power Engineering Conference, 2005, pp. 1-137.

[8] B. Chandra and V. P. Paul, "A Robust Algorithm for Classification Using Decision Trees," 2006 IEEE Conference on Cybernetics and Intelligent Systems, 2006, pp. 1-5