FSU Panama City

EEL 4742 Advanced Microprocessor-Based System Design

Performed by:

Jaehyun Lee

Instructor: Dr. Manzak

Dec 5, 2022

Supervised Learning in Quest

Abstract

SLIQ (Supervised Learning in Quest) is a high speed and flexible decision tree classifier developed by IBM Almaden Research Center in 1996 that allows us 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.

Introduction

SLIQ is a decision tree algorithm. This means it splits nodes into two or more sub-nodes based on some criteria. 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 uses a training set and a Gini split to prepare the data for the decision tree algorithm. These equations are what make SLIQ a," supervised learning," algorithms as the data is pre-sorted and pruned. For training set L with n distinct classes the equation is shown blow where the variables are given within the histogram example.

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

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

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

SLIQ is an improvement upon existing decision tree algorithms. SLIQ's advantages are based in its pre-sorting of data, no need for data normalization, scales well with data size, and 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].

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 categorial 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].

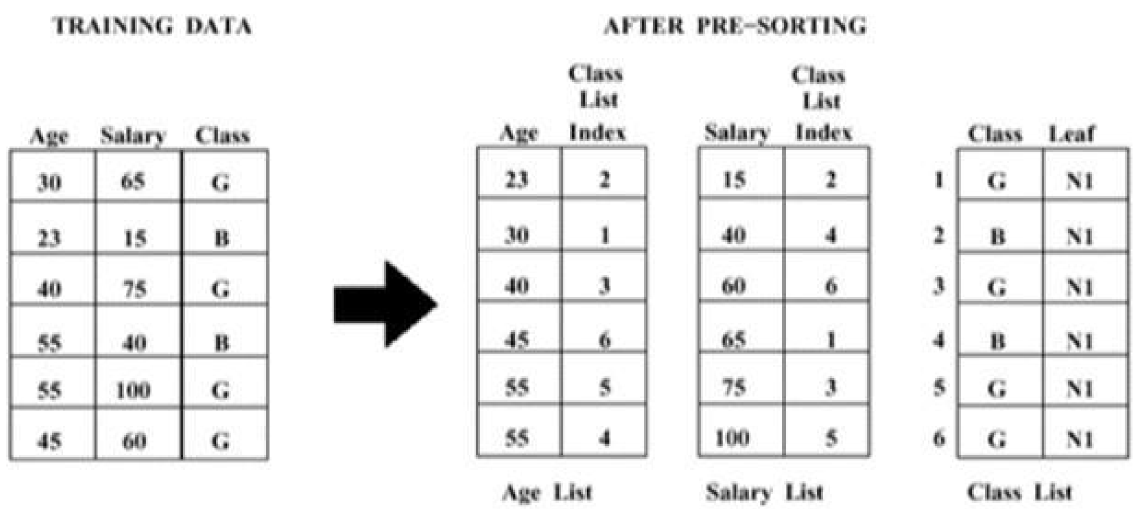
In 2005, through the International Power Engineering Conference, three engineers, named Hongwen Yan, Rui Ma, and Xiaojiao Tong proposed using SLIQ to build a framework for a competitive bidding assessment in a deregulated power market. They suggested that the bidding system using the SLIQ algorithm could 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 examples algorithms of precipitation prediction, bidding, or sorting data. There are also algorithms that identifies 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

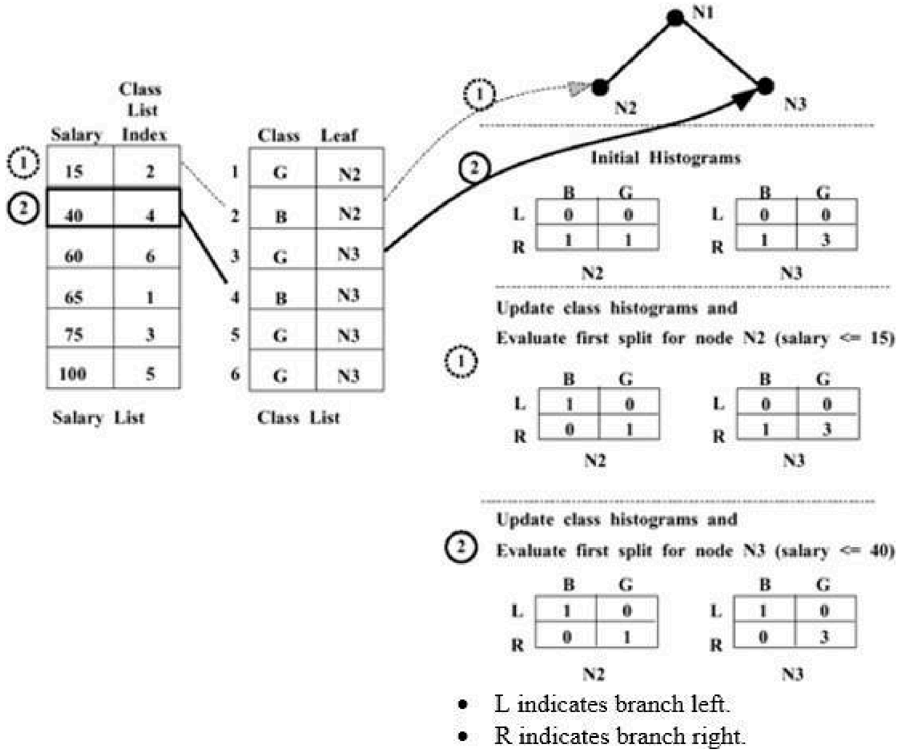
- Algorithm

SLIQ is a decision tree classifier that can improve learning time for the classifier without any loss in accuracy while this technique allows performing on the larger training data. SLIQ uses Gini Index to determine the best split for each node [8]. SLIQ Algorithm can be divided into 3 steps, pre-sorting the sample, processing evaluation on splits, and updating the class list [4].

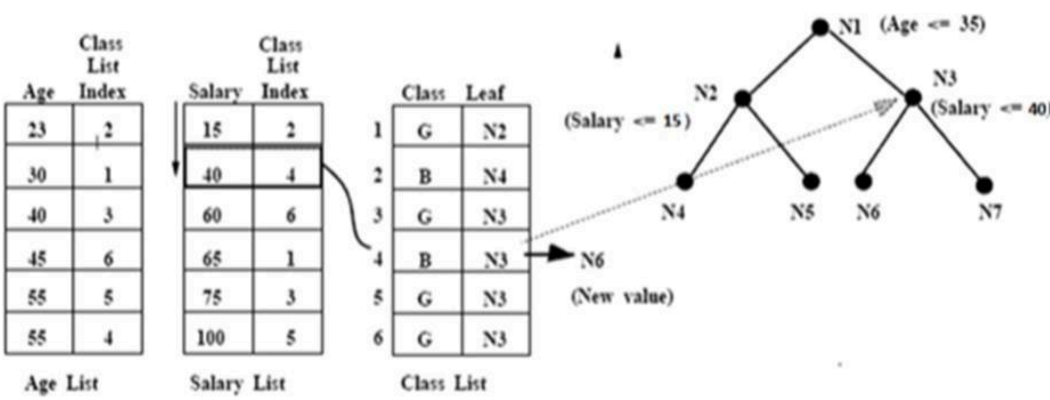


Pre-sorting the sample with given training data, create an attribute list for each attribute.

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. These steps can be repeated until each of the leaf nodes becomes a pure node, meaning that the node only contains one class.



Example

- Question

The question for the example given is whether an individual will be chosen to receive a credit card. The decision tree shows the results using the given data.

Credit Score	Married Status	Debt Existence	House Own	Credit Card
302	Y	Y	N	N
353	Y	Y	N	N
420	N	N	Y	Y
545	N	Y	N	N
610	Y	N	N	N
610	N	N	Y	Y
710	N	N	N	Y
720	N	Y	Y	N
780	Y	Y	Y	Y
850	N	N	Y	Y

- Solution

Credit Score	Married Status	Debt Existence	House Own	Credit Card
302	Y	Y	N	N
353	Y	Y	N	N
420	N	N	Y	Y
545	N	Y	N	N
610	Y	N	N	N
610	N	N	Y	Y
710	N	N	N	Y
720	N	Y	Y	N
780	Y	Y	Y	Y
850	N	N	Y	Y

Before pre-sorting

Credit Score	Index
302	1
353	2
420	3
545	4
610	5
610	6
710	7
720	8
780	9
850	10

Married Status	Index
Y	1
Y	2
Y	5
Y	9
N	3
N	4
N	6
N	7
N	8
N	10

Debt Existence	Index
Y	1
Y	2
Y	4
Y	8
Y	9
N	3
N	5
N	6
N	7
N	10

House Owned	Index
Y	3
Y	6
Y	8
Y	9
Y	10
N	1
N	2
N	4
N	5
N	7

After pre-sorting

Determine the Gini index using the equation and the histogram made based upon the data of the attributes. Where P is the value of the attribute obtained from the sorted data, A is true, B is fails, L is branch left, R is branch right, and a and b is the number of the elements that fall into the criteria of the table.

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

Histogram example

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

Gini index equation

Using the equation, Gini index for each attribute can be obtained in order to find the attribute for the root node.

Gini index for the married status attribute is

$$P\ (M = y)\ \{1 - [P\ (M = y\ \&\&\ C = y)^2 + P(M = y\ \&\ C = n)^2]\ \} +$$
$$P\ (M = n)\ \{1 - [P\ (M = n\ \&\&\ C = y)^2 + P\ (M = n\ \&\ C = n)^2]\ \} = 0.41$$

Gini index for the debt existence attribute is

$$P\ (D = y)\ \{1 - [P\ (D = y\ \&\&\ C = y)^2 + P(D = y\ \&\ C = n)^2]\ \} +$$
$$P\ (D = n)\ \{1 - [P\ (D = n\ \&\&\ C = y)^2 + P\ (D = n\ \&\ C = n)^2]\ \} = 0.34$$

Gini index for the house own attribute is

$$P\ (H = y)\ \{1 - [P\ (H = y\ \&\&\ C = y)^2 + P(H = y\ \&\ C = n)^2]\ \} +$$
$$P\ (H = n)\ \{1 - [P\ (H = n\ \&\&\ C = y)^2 + P\ (H = n\ \&\ C = n)^2]\ \} = 0.31$$

However, the credit score attribute does not predictive power since they can be any number meaning that this attribute can be continuous number. From the Gini indexes found, the attribute with the lowest Gini index gets selected as the root node, the first node, N1. Then same operations can be performed for the root node to find the sub node and so on. In these steps new data tables can be created using the pre-sorted data tables in order to make the process easier. New data tables are shown blow.

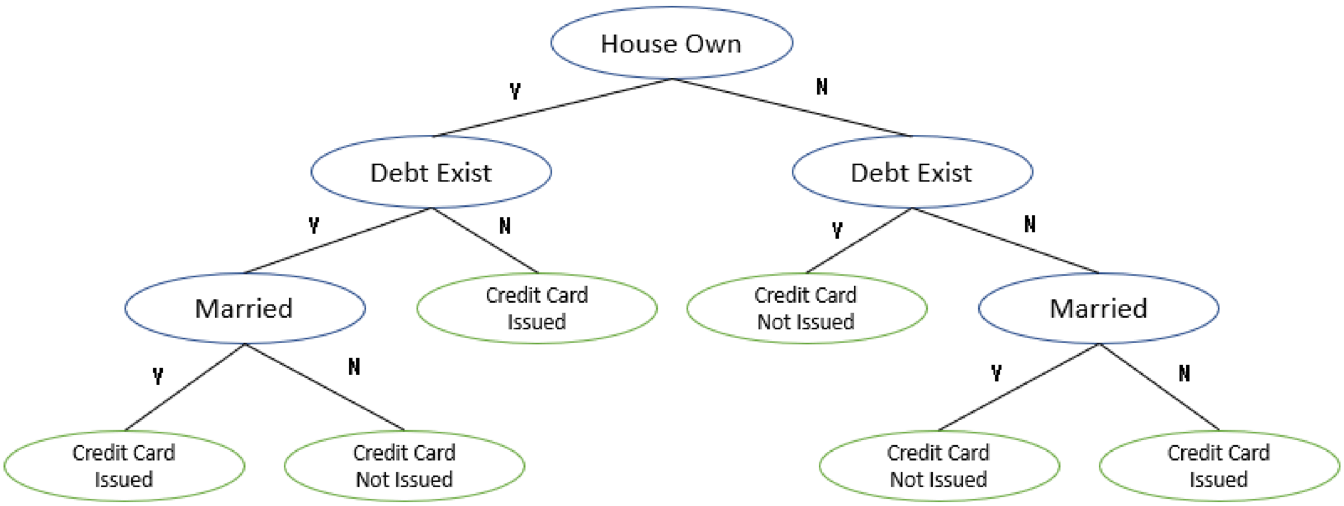
N2

House Own	Married Status	Debt Existence	Credit Card
Y	N	N	Y
Y	N	N	Y
Y	N	Y	N
Y	Y	Y	Y

N3

House Own	Married Status	Debt Existence	Credit Card
N	Y	Y	N
N	Y	Y	N
N	N	Y	N
N	Y	N	N

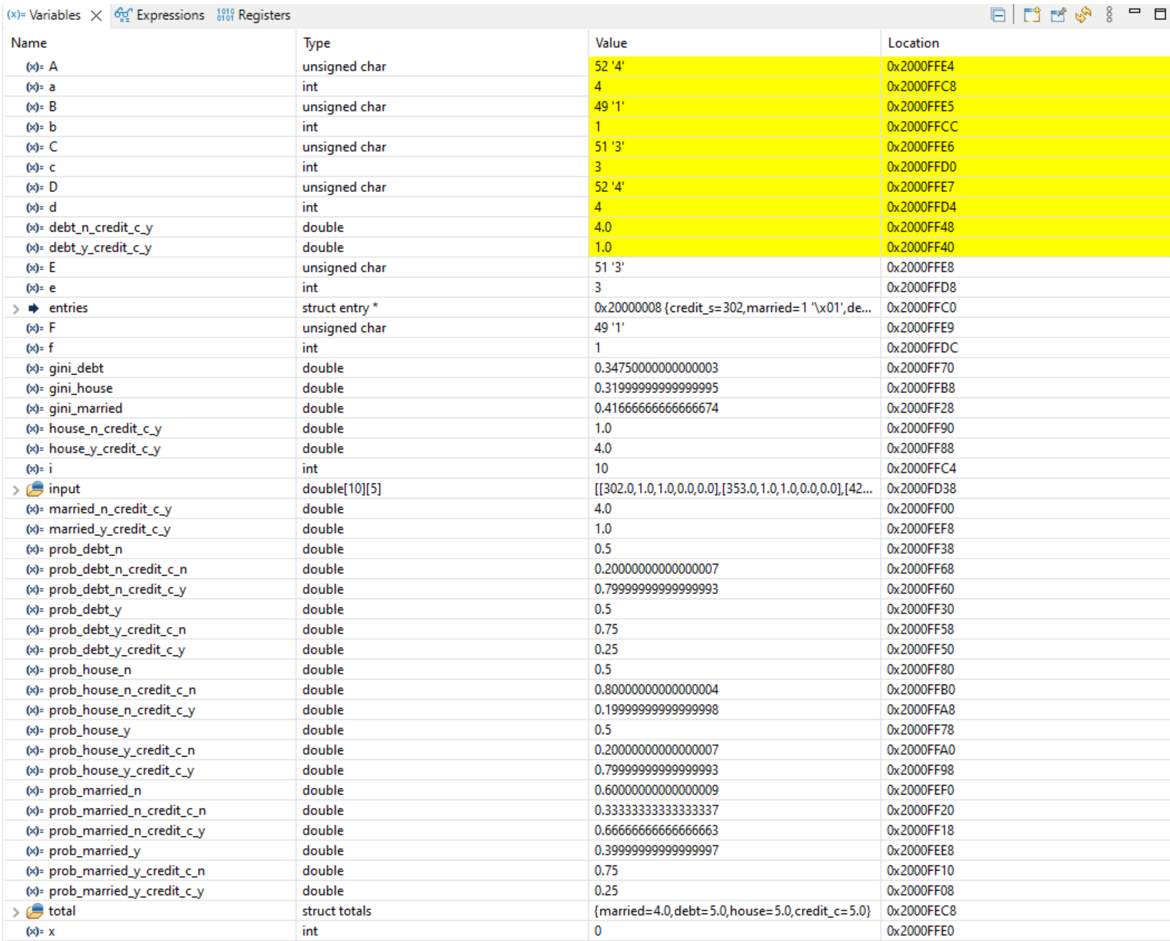
Through the process a decision tree can be created which can be utilized and updated as a based model for various input data in the future.



Methodology

As seen in the previous example, finding the Gini index for each attribute can be created as a program in order to determine the attribute that will be used for each node and whether or not a person is eligible to receive a credit card based on a set of data. The factors include the person's current credit score, married status, debt existence, and house own Below are Tera terminal and Code Composer Studio debugged screen. The Tera terminal is connected with the Texas Instruments MSP432P401 through the USB port and set with 115200 baud rate. It will display the output which is the Gini index for each attribute with given training data. CCS debugged screen will show the values for each variable that were being used within the program. Using the screen debugging can be simpler and efficient.

Tera terminal output



Variables | Expressions | Registers

Name	Type	Value	Location
A	unsigned char	52 '4'	0x2000FFE4
a	int	4	0x2000FFC8
B	unsigned char	49 '1'	0x2000FFE5
b	int	1	0x2000FFCC
C	unsigned char	51 '3'	0x2000FFE6
c	int	3	0x2000FFD0
D	unsigned char	52 '4'	0x2000FFE7
d	int	4	0x2000FFD4
debt_n_credit_c_y	double	4.0	0x2000FF48
debt_y_credit_c_y	double	1.0	0x2000FF40
E	unsigned char	51 '3'	0x2000FFE8
e	int	3	0x2000FFD8
entries	struct entry *	0x20000008 {credit_s=302,married=1 '\x01',de...	0x2000FFC0
F	unsigned char	49 '1'	0x2000FFE9
f	int	1	0x2000FFDC
gini_debt	double	0.34750000000000003	0x2000FF70
gini_house	double	0.31999999999999995	0x2000FFB8
gini_married	double	0.41666666666666674	0x2000FF28
house_n_credit_c_y	double	1.0	0x2000FF90
house_y_credit_c_y	double	4.0	0x2000FF88
i	int	10	0x2000FFC4
input	double[10][5]	[[302.0,1.0,1.0,0.0,0.0],[353.0,1.0,1.0,0.0,0.0],[42...	0x2000FD38
married_n_credit_c_y	double	4.0	0x2000FF00
married_y_credit_c_y	double	1.0	0x2000FEF8
prob_debt_n	double	0.5	0x2000FF38
prob_debt_n_credit_c_n	double	0.20000000000000007	0x2000FF68
prob_debt_n_credit_c_y	double	0.79999999999999993	0x2000FF60
prob_debt_y	double	0.5	0x2000FF30
prob_debt_y_credit_c_n	double	0.75	0x2000FF58
prob_debt_y_credit_c_y	double	0.25	0x2000FF50
prob_house_n	double	0.5	0x2000FF80
prob_house_n_credit_c_n	double	0.80000000000000004	0x2000FFB0
prob_house_n_credit_c_y	double	0.19999999999999998	0x2000FFA8
prob_house_y	double	0.5	0x2000FF78
prob_house_y_credit_c_n	double	0.20000000000000007	0x2000FFA0
prob_house_y_credit_c_y	double	0.79999999999999993	0x2000FF98
prob_married_n	double	0.60000000000000009	0x2000FEF0
prob_married_n_credit_c_n	double	0.33333333333333337	0x2000FF20
prob_married_n_credit_c_y	double	0.66666666666666663	0x2000FF18
prob_married_y	double	0.39999999999999997	0x2000FEE8
prob_married_y_credit_c_n	double	0.75	0x2000FF10
prob_married_y_credit_c_y	double	0.25	0x2000FF08
total	struct totals	{married=4.0,debt=5.0,house=5.0,credit_c=5.0}	0x2000FEC8
x	int	0	0x2000FFE0

Code Composer Studio Debugged Screen

Though the way the code was written was great, there are some other ways it could have been written an improve. Using more function would have made the code more simpler and less complicated as well as reducing the number of variables within the code significantly. Another improvement can could have been made will be making the program to compare the Gini index for each attribute and create a new two dimension array to calculate the Gini index for the sub nodes. This will increase the size of the code file however using the functions to calculate the Gini index , the size will decrease by a lot since calculations are going to be the same.

Result

Using the C language code of the SLIQ algorithm, it was easier to create a decision tree using the given training data for deciding whether or not the credit card will be issued. Gini index played as an important role with in the SLIQ algorithm and finding the Gini index with the code clearly outputted the information that was required to create the decision tree. CCS debugging feature allowed to debug the code easier making it possible to output the information to the Tera terminal.

Conclusion

In this project, a student successfully used the supervised learning in quest (SLIQ) algorithm to determine whether or not a person from the training data should receive a credit card. The student used a data set and Gini index to determine which attribute information was most important and which was not. The student was then able to create a decision tree with multiple nodes and branches. Afterwards, the student used the decision tree to serve as a guide to the coding process. Like how a decision tree makes many comparisons, the program includes many different comparison functions. From this project, the student learned the history behind SLIQ and decision tree algorithms, they also used previous work done by other class and improved the troubleshooting skills when things went wrong in the coding process. For future work, the student would alter the code so that it could handle larger sets of data like SLIQ is designed for and also like to implement their design onto hardware where it could receive inputs and give back outputs as a decision tree diagrams.

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

CCS Code

```
#include "msp.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

void UART0_init(void);

// structure for the attribute
typedef struct entry
{
    int credit_s;
    bool married;
    bool debt;
    bool house;
    bool credit_c;
} entry;

// structure for counting each attribute
typedef struct totals
{
    double married;
    double debt;
    double house;
    double credit_c;
} totals;

// function to analyze the input data (return type: entry structure)
entry* parse_input(double input[10][5])
{
    // allocate the structure 'result' with memory size 100
    entry* result = malloc(100);
    int i;
    // summarize each attribute depending on each attributes' value
    for (i = 0; i < 10; i++)
    {
        result[i].credit_s = input[i][0];

        if (input[i][1] == 1)
        {
            result[i].married = true;
        }
        else
        {
            result[i].married = false;
        }

        if (input[i][2] == 1)
        {
            result[i].debt = true;
        }
        else
        {
            result[i].debt = false;
        }
```

```
        if (input[i][3] == 1)
        {
            result[i].house = true;
        }
        else
        {
            result[i].house = false;
        }

        if (input[i][4] == 1)
        {
            result[i].credit_c = true;
        }
        else
        {
            result[i].credit_c = false;
        }
    }

    return result;
}

// function to count the total number of 1s (yes) for each attribute
// of the data (return type: totals structure)
totals get_totals(entry* entries)
{
    totals result = {0};
    int i;
    for (i = 0; i < 10; ++i)
    {
        if (entries[i].married == true)
        {
            result.married += 1;
        }
        if (entries[i].debt == true)
        {
            result.debt += 1;
        }
        if (entries[i].house == true)
        {
            result.house += 1;
        }
        if (entries[i].credit_c == true)
        {
            result.credit_c += 1;
        }
    }

    return result;
}


void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;// stop watchdog timer
```

```
// training data table as a 2d array
// Credit Score     Married Status  Debt Status     House Own   Credit Card
//                         Yes = 1     True = 1       Yes = 1       Yes = 1
//                          No = 0       No = 0        No = 0        No = 0
double input[10][5] =
{
 {302,1,1,0,0},
 {353,1,1,0,0},
 {420,0,0,1,1},
 {545,0,1,0,0},
 {610,1,0,0,0},
 {610,0,0,1,1},
 {710,0,0,0,1},
 {720,0,1,1,0},
 {780,1,1,1,1},
 {850,0,0,1,1}
};

entry* entries = parse_input(input);

totals total = get_totals(entries);

//calculate gini for married
double prob_married_y = total.married / 10;
double prob_married_n = 1 - prob_married_y;

double married_y_credit_c_y = 0;
double married_n_credit_c_y = 0;

int i;
for (i = 0; i < 10; ++i)
{
    if (entries[i].married == true && entries[i].credit_c == true)
    {
        married_y_credit_c_y += 1;
    }

    if (entries[i].married == false && entries[i].credit_c == true)
    {
        married_n_credit_c_y += 1;
    }
}

double prob_married_y_credit_c_y = married_y_credit_c_y / total.married;
double prob_married_y_credit_c_n = 1 - prob_married_y_credit_c_y;


double prob_married_n_credit_c_y = married_n_credit_c_y / (10 - total.married);
double prob_married_n_credit_c_n = 1 - prob_married_n_credit_c_y;

double gini_married = ((prob_married_y) * (1 - (pow(prob_married_y_credit_c_y, 2) +
                 pow(prob_married_y_credit_c_n, 2)))) + ((prob_married_n) *
         (1 - (pow(prob_married_n_credit_c_y, 2) + pow(prob_married_n_credit_c_n, 2))));
```

```
// calculate gini for debt
double prob_debt_y = total.debt / 10;
double prob_debt_n = 1 - prob_debt_y;

double debt_y_credit_c_y = 0;
double debt_n_credit_c_y = 0;

for (i = 0; i < 10; ++i)
{
    if (entries[i].debt == true && entries[i].credit_c == true)
    {
        debt_y_credit_c_y += 1;
    }

    if (entries[i].debt == false && entries[i].credit_c == true)
    {
        debt_n_credit_c_y += 1;
    }
}

double prob_debt_y_credit_c_y = debt_y_credit_c_y / total.married;
double prob_debt_y_credit_c_n = 1 - prob_debt_y_credit_c_y;

double prob_debt_n_credit_c_y = debt_n_credit_c_y / (10 - total.debt);
double prob_debt_n_credit_c_n = 1 - prob_debt_n_credit_c_y;

double gini_debt= ((prob_debt_y) * (1 - (pow(prob_debt_y_credit_c_y, 2)
       + pow(prob_debt_y_credit_c_n, 2)))) + ((prob_debt_n) *
       (1 - (pow(prob_debt_n_credit_c_y, 2) + pow(prob_debt_n_credit_c_n, 2))));


// calculate gini for house
double prob_house_y = total.house / 10;
double prob_house_n = 1 - prob_house_y;

double house_y_credit_c_y = 0;
double house_n_credit_c_y = 0;

for (i = 0; i < 10; i++)
{
    if (entries[i].house == true && entries[i].credit_c == true)
    {
        house_y_credit_c_y += 1;
    }

    if (entries[i].house == false && entries[i].credit_c == true)
    {
        house_n_credit_c_y += 1;
    }
}

double prob_house_y_credit_c_y = house_y_credit_c_y / total.house;
double prob_house_y_credit_c_n = 1 - prob_house_y_credit_c_y;

double prob_house_n_credit_c_y = house_n_credit_c_y / (10 - total.house);
double prob_house_n_credit_c_n = 1 - prob_house_n_credit_c_y;
```

```
double gini_house = ((prob_house_y) * (1 - (pow(prob_house_y_credit_c_y, 2)
        + pow(prob_house_y_credit_c_n, 2))))  + ((prob_house_n) *
            (1 - (pow(prob_house_n_credit_c_y, 2) + pow(prob_house_n_credit_c_n, 2))));

// integers and char to represents the decimal points for each gini index
int a, b, c, d, e, f;
char A, B, C, D, E, F;

a = gini_married * 10;
A = (char)a + '0';
b = (gini_married * 100) - (a * 10);
B = (char)b + '0';

c = gini_debt *10;
C = (char)c +'0';
d = (gini_debt * 100) - (c * 10);
D = (char)d +'0';

e = gini_house * 10;
E = (char)e +'0';
f = (gini_house * 100) - (e * 10);
F = (char)f +'0';

int x = 0;

UART0_init();

    // print out each gini index to the terminal to decide which attribute
    // to be used for splitting the decision tree

    // first attribute
    while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */
    EUSCI_A0->TXBUF = '1';              /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = ':';              /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = ' ';              /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = '0';              /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = '.';              /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = A;          /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = B;
    while(!(EUSCI_A0->IFG & 0x02)) { }
    EUSCI_A0->TXBUF = '\t';



    // second attribute
    while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */
    EUSCI_A0->TXBUF = '2';              /* send a char */
    while(!(EUSCI_A0->IFG & 0x02)) { }
```

```
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = ':';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = ' ';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = '0';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = '.';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = C;
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = D;
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = '\t';


        // third attribute
        while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */
        EUSCI_A0->TXBUF = '3';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = ':';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = ' ';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = '0';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = '.';
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = E;
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = F;
        while(!(EUSCI_A0->IFG & 0x02)) { }
        EUSCI_A0->TXBUF = '\t';

        while(1){}
}

void UART0_init(void)
{
    EUSCI_A0->CTLW0 |= 1;         /* put in reset mode for config */
    EUSCI_A0->MCTLW = 0;          /* disable oversampling */
    EUSCI_A0->CTLW0 = 0x0081;     /* 1 stop bit, no parity, SMCLK, 8-bit data */
    EUSCI_A0->BRW = 26;           /* 3,000,000 / 115200 = 26 */
    P1->SEL0 |= 0x0C;             /* P1.3, P1.2 for UART */
    P1->SEL1 &= ~0x0C;
    EUSCI_A0->CTLW0 &= ~1;        /* take UART out of reset mode */
}
```