Supervised Learning In Quest SLIQ

Jaehyun Lee

FALL2022

EEL4742

Introduction

What is SLIQ?

- SLIQ (Supervised Learning in Quest) is a high speed and flexible decision tree classifier that allows to sort and interpret data.
- 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.
- SLIQ Algorithm can be divided into 3 steps, pre-sorting the sample, processing evaluation on splits, and updating the class list.

Gini Index

- 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:

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

Histogram example

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

- pj is the relative frequency of j.

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

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

with data classes a and b :

Example

Training Data Table

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

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

- Credit Score attribute does not have predictive power since they can be any number (continuous).

Gini Index

-Finding Gini index for each attribute, in order to find the root node.

-Gini index for Married status attribute =

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

-Gini index for Debt existence attribute = 0.34

-Gini index for House own attribute = 0.31

-House own attribute is used for the root node.

Processing Evaluation on Splits

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

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

Credit Card	Index
N	3
N	6
Y	7
N	9
N	10
Y	1
Y	2
N	4
Y	5
Y	8

- Using the root node create and determine the sub node.

Updating the Class List.

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
Y	N	N	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
N	N	N	Y

- Previous steps repeat for each sub nodes.

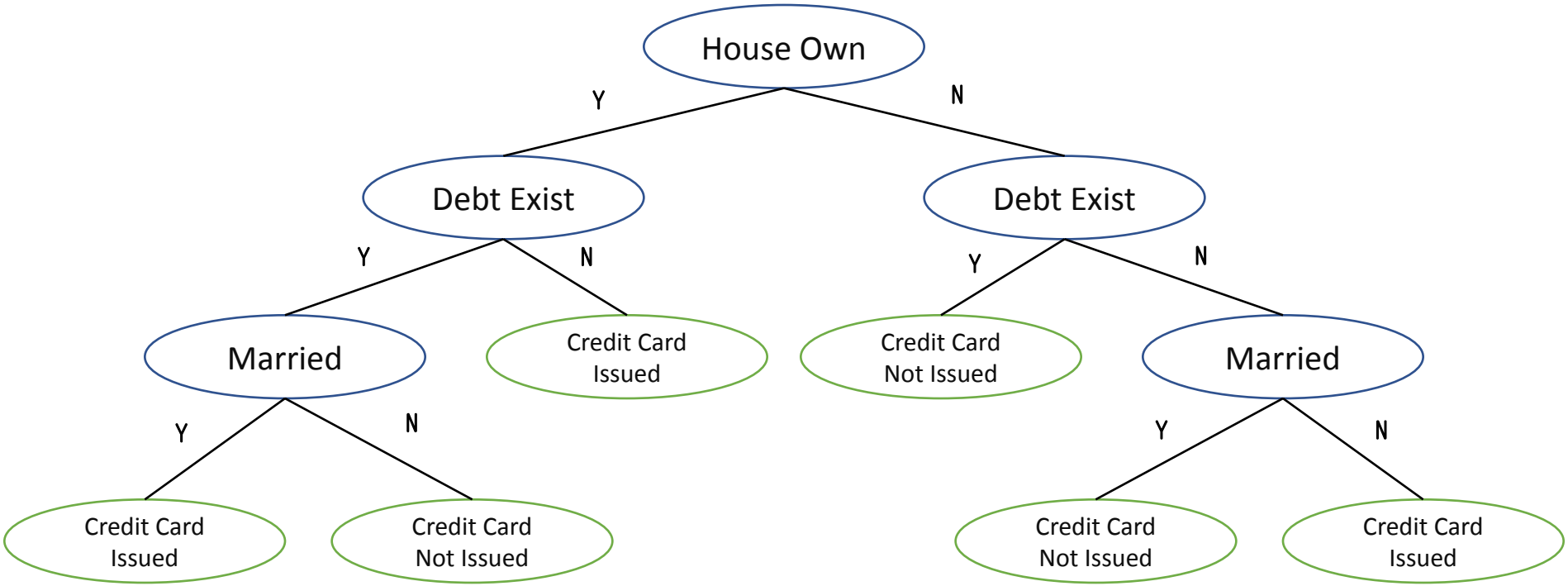
Gini Index for Sub Nodes N2 and N3

- N2: Gini index for Married status attribute = 0.3
  Gini index for Debt existence attribute = 0.2
- Debt existence attribute is used on sub node N2.

- N3: Gini index for Married status attribute = 0.2
  Gini index for Debt existence attribute = 0.2
- Either Debt existence or Married status attribute can be used on sub node N3.

Decision Tree

- Decision tree made based on the training data:



Code

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
```

Code

```
114    // training data table as a 2d array
115    // Credit Score     Married Status  Debt Status     House Own   Credit Card
116    //                        Yes = 1     True = 1       Yes = 1       Yes = 1
117    //                         No = 0       No = 0        No = 0        No = 0
118    double input[10][5] =
119    {
120     {302,1,1,0,0},
121     {353,1,1,0,0},
122     {420,0,0,1,1},
123     {545,0,1,0,0},
124     {610,1,0,0,0},
125     {610,0,0,1,1},
126     {710,0,0,0,1},
127     {720,0,1,1,0},
128     {780,1,1,1,1},
129     {850,0,0,1,1}
130    };
131
132    entry* entries = parse_input(input);
133
134    totals total = get_totals(entries);
135
136    //calculate gini for married
137    double prob_married_y = total.married / 10;
138    double prob_married_n = 1 - prob_married_y;
139
140    double married_y_credit_c_y = 0;
141    double married_n_credit_c_y = 0;
142
143    int i;
144    for (i = 0; i < 10; ++i)
145    {
146        if (entries[i].married == true && entries[i].credit_c == true)
147        {
148            married_y_credit_c_y += 1;
149        }
150
151        if (entries[i].married == false && entries[i].credit_c == true)
152        {
153            married_n_credit_c_y += 1;
154        }
155    }
156
157    double prob_married_y_credit_c_y = married_y_credit_c_y / total.married;
158    double prob_married_y_credit_c_n = 1 - prob_married_y_credit_c_y;
159
160
161    double prob_married_n_credit_c_y = married_n_credit_c_y / (10 - total.married);
162    double prob_married_n_credit_c_n = 1 - prob_married_n_credit_c_y;
163
164    double gini_married = ((prob_married_y) * (1 - (pow(prob_married_y_credit_c_y, 2) +
165                    pow(prob_married_y_credit_c_n, 2)))) + ((prob_married_n) *
166            (1 - (pow(prob_married_n_credit_c_y, 2) + pow(prob_married_n_credit_c_n, 2))));
167
168
```

Code

```
169     // calculate gini for debt
170     double prob_debt_y = total.debt / 10;
171     double prob_debt_n = 1 - prob_debt_y;
172
173     double debt_y_credit_c_y = 0;
174     double debt_n_credit_c_y = 0;
175
176     for (i = 0; i < 10; ++i)
177     {
178         if (entries[i].debt == true && entries[i].credit_c == true)
179         {
180             debt_y_credit_c_y += 1;
181         }
182
183         if (entries[i].debt == false && entries[i].credit_c == true)
184         {
185             debt_n_credit_c_y += 1;
186         }
187     }
188
189     double prob_debt_y_credit_c_y = debt_y_credit_c_y / total.married;
190     double prob_debt_y_credit_c_n = 1 - prob_debt_y_credit_c_y;
191
192     double prob_debt_n_credit_c_y = debt_n_credit_c_y / (10 - total.debt);
193     double prob_debt_n_credit_c_n = 1 - prob_debt_n_credit_c_y;
194
195     double gini_debt= ((prob_debt_y) * (1 - (pow(prob_debt_y_credit_c_y, 2)
196            + pow(prob_debt_y_credit_c_n, 2)))) + ((prob_debt_n) *
197            (1 - (pow(prob_debt_n_credit_c_y, 2) + pow(prob_debt_n_credit_c_n, 2))));
198
199
200     // calculate gini for house
201     double prob_house_y = total.house / 10;
202     double prob_house_n = 1 - prob_house_y;
203
204     double house_y_credit_c_y = 0;
205     double house_n_credit_c_y = 0;
206
207     for (i = 0; i < 10; i++)
208     {
209         if (entries[i].house == true && entries[i].credit_c == true)
210         {
211             house_y_credit_c_y += 1;
212         }
213
214         if (entries[i].house == false && entries[i].credit_c == true)
215         {
216             house_n_credit_c_y += 1;
217         }
218     }
219
220     double prob_house_y_credit_c_y = house_y_credit_c_y / total.house;
221     double prob_house_y_credit_c_n = 1 - prob_house_y_credit_c_y;
222
223     double prob_house_n_credit_c_y = house_n_credit_c_y / (10 - total.house);
224     double prob_house_n_credit_c_n = 1 - prob_house_n_credit_c_y;
```

Code

```
225
226     double gini_house = ((prob_house_y) * (1 - (pow(prob_house_y_credit_c_y, 2)
227             + pow(prob_house_y_credit_c_n, 2)))  + ((prob_house_n) *
228                 (1 - (pow(prob_house_n_credit_c_y, 2) + pow(prob_house_n_credit_c_n, 2))));
229
230     // integers and char to represents the decimal points for each gini index
231     int a, b, c, d, e, f;
232     char A, B, C, D, E, F;
233
234     a = gini_married * 10;
235     A = (char)a + '0';
236     b = (gini_married * 100) - (a * 10);
237     B = (char)b + '0';
238
239     c = gini_debt *10;
240     C = (char)c +'0';
241     d = (gini_debt * 100) - (c * 10);
242     D = (char)d +'0';
243
244     e = gini_house * 10;
245     E = (char)e +'0';
246     f = (gini_house * 100) - (e * 10);
247     F = (char)f +'0';
248
249     int x = 0;
250
251     UART0_init();
252
253         // print out each gini index to the terminal to decide which attribute
254         // to be used for splitting the decision tree
255
256         // first attribute
257         while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */
258         EUSCI_A0->TXBUF = '1';              /* send a char */
259         while(!(EUSCI_A0->IFG & 0x02)) { }
260         EUSCI_A0->TXBUF = ':';              /* send a char */
261         while(!(EUSCI_A0->IFG & 0x02)) { }
262         EUSCI_A0->TXBUF = ' ';              /* send a char */
263         while(!(EUSCI_A0->IFG & 0x02)) { }
264         EUSCI_A0->TXBUF = '0';              /* send a char */
265         while(!(EUSCI_A0->IFG & 0x02)) { }
266         EUSCI_A0->TXBUF = '.';              /* send a char */
267         while(!(EUSCI_A0->IFG & 0x02)) { }
268         EUSCI_A0->TXBUF = A;          /* send a char */
269         while(!(EUSCI_A0->IFG & 0x02)) { }
270         EUSCI_A0->TXBUF = B;
271         while(!(EUSCI_A0->IFG & 0x02)) { }
272         EUSCI_A0->TXBUF = '\t';
273
274
275
276         // second attribute
277         while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */
278         EUSCI_A0->TXBUF = '2';              /* send a char */
279         while(!(EUSCI_A0->IFG & 0x02)) { }
```

Code

```
279         while(!(EUSCI_A0->IFG & 0x02)) { }
280         EUSCI_A0->TXBUF = ':';
281         while(!(EUSCI_A0->IFG & 0x02)) { }
282         EUSCI_A0->TXBUF = ' ';
283         while(!(EUSCI_A0->IFG & 0x02)) { }
284         EUSCI_A0->TXBUF = '0';
285         while(!(EUSCI_A0->IFG & 0x02)) { }
286         EUSCI_A0->TXBUF = '.';
287         while(!(EUSCI_A0->IFG & 0x02)) { }
288         EUSCI_A0->TXBUF = C;
289         while(!(EUSCI_A0->IFG & 0x02)) { }
290         EUSCI_A0->TXBUF = D;
291         while(!(EUSCI_A0->IFG & 0x02)) { }
292         EUSCI_A0->TXBUF = '\t';
293
294
295         // third attribute
296         while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */
297         EUSCI_A0->TXBUF = '3';
298         while(!(EUSCI_A0->IFG & 0x02)) { }
299         EUSCI_A0->TXBUF = ':';
300         while(!(EUSCI_A0->IFG & 0x02)) { }
301         EUSCI_A0->TXBUF = ' ';
302         while(!(EUSCI_A0->IFG & 0x02)) { }
303         EUSCI_A0->TXBUF = '0';
304         while(!(EUSCI_A0->IFG & 0x02)) { }
305         EUSCI_A0->TXBUF = '.';
306         while(!(EUSCI_A0->IFG & 0x02)) { }
307         EUSCI_A0->TXBUF = E;
308         while(!(EUSCI_A0->IFG & 0x02)) { }
309         EUSCI_A0->TXBUF = F;
310         while(!(EUSCI_A0->IFG & 0x02)) { }
311         EUSCI_A0->TXBUF = '\t';
312
313         while(1){}
314 }
315
316 void UART0_init(void)
317 {
318     EUSCI_A0->CTLW0 |= 1;       /* put in reset mode for config */
319     EUSCI_A0->MCTLW = 0;        /* disable oversampling */
320     EUSCI_A0->CTLW0 = 0x0081;   /* 1 stop bit, no parity, SMCLK, 8-bit data */
321     EUSCI_A0->BRW = 26;         /* 3,000,000 / 115200 = 26 */
322     P1->SEL0 |= 0x0C;           /* P1.3, P1.2 for UART */
323     P1->SEL1 &= ~0x0C;
324     EUSCI_A0->CTLW0 &= ~1;      /* take UART out of reset mode */
325 }
326
```

Code

(x)= Variables | Expressions | Registers

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

Demo