HOW DUPLICATES AFFECT THE ERROR RATE OF DATASETS DURING VALIDATION

By

Nagabhushana Gargeshwari Mahadevaswamy

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Chairperson Dr. Jerzy W. Grzymala-Busse

Dr. Prasad Kulkarni

Dr. Bo Luo

Date Defended:

The Project Committee for Nagabhushana Gargeshwari Mahadevaswamy
certifies that this is the approved version of the following project:

HOW DUPLICATES AFFECT THE ERROR RATE OF DATASETS DURING VALIDATION

Chairperson, Dr. Jerzy W. Grzymala Busse

Date Approved:

ABSTRACT

In data mining, duplicate data plays a huge role in deciding the set of rules. In this project, an analysis has been made on finding the impact of duplicates in the input data set on the rule set. The effect of duplicates is being analyzed using the error rate factor. Error rate is calculated by comparing the obtained rule set against the testing part of input data. The results of experiments have shown decrement of error rate with the increase of percentage of duplicates in the input data set, which demonstrates that the duplicate data plays a crucial role in validation process of machine learning. LEM2 algorithm and rule checker application have been implemented as a part of project. LEM2 algorithm is used to induce the rule set for the given input data set and rule checker application is used to calculate the error rate.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my academic advisor, Dr. Jerzy W. Grzymala-Busse, EECS Department for his constant supervision and support throughout my Master's program. He has been kind enough to spend ample time and share his knowledge with me. As an advisor he always guided me in the right direction for both my career and project.

Secondly, I would like to take this opportunity to thank Dr. Prasad Kulkarni and Dr. Bo Luo for imparting their knowledge to me through their courses, and being a part of my graduation committee.

Also, I would like to thank my wife Vyshnavi, parents and friends who supported me all through my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Nowadays due to increase in dependency on the technology, the proportion of data that needs to be handled is very large in volume. The obtained data are being used in many fields like data mining, machine learning etc in order to derive some meaningful conclusion and make prediction. Usually data will be collected from wide variety of resources, where there might be relation between the data resource or there might not be. Since the data is coming from wide variety of resources, the rate of redundancy in the resulting dataset is high. Increase in redundancy of data is causing a lot of inverse effect on the final decision and prediction.

1.1 Aim of the project

The goal of the project is to study the effect of duplicate data on the error rate during validation by using LEM2 algorithm. As a part of project, I implemented LEM2 algorithm and rule checker application using c# and java respectively. LEM2 algorithm is used to obtain a rule set by giving data set as input and rule checker application is used to obtain the error rate by giving rule set and data set as input. Finally, the obtained error rate was plotted against the percentage of duplicates to learn the effect of duplicates.

1.2 Overview

Figure 1 depicts an overview of rule checker. Here the input data is randomly divided into two parts. First 50% of randomly selected cases from the data set will be given as input to the LEM2 algorithm program and generated set of rules will be stored in file.

In the next step, another 50% of data from the data set and the resulting rule set from LEM2 will be given as an input to the rule checker application. The rule checker produces average error rate based on hits and misses considering all the rules. Error rate is basically the ratio of failed cases over the total number of relevant cases.

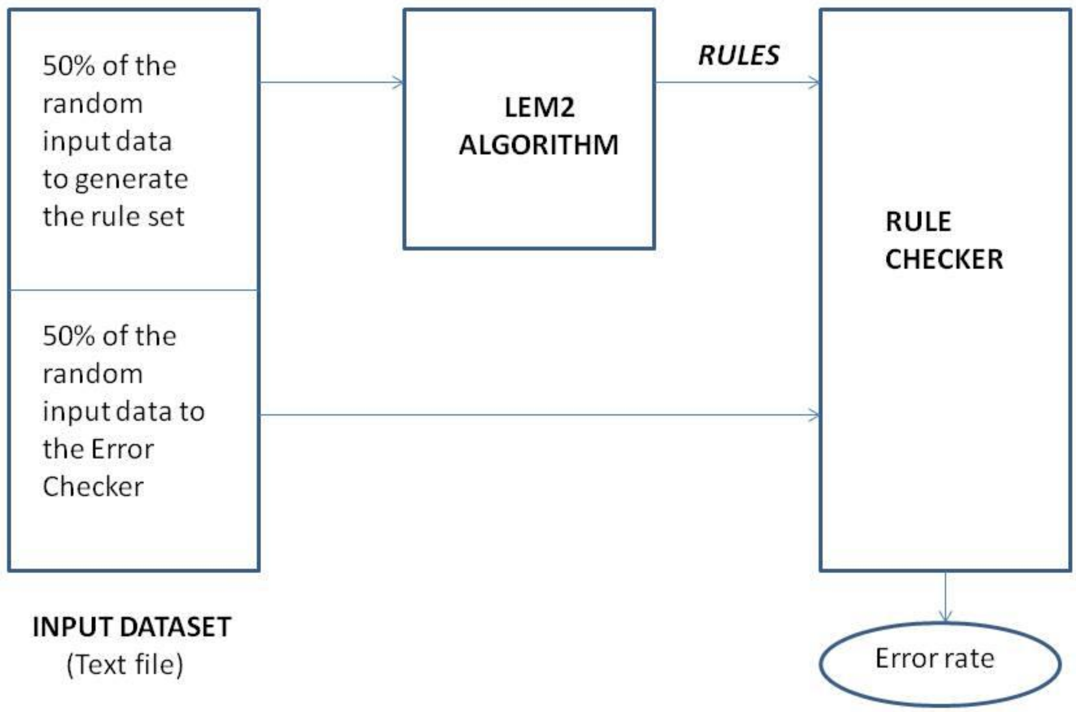

Figure 1: Overview of the rule checker

In the following sections, after brief explanation about LEM2 algorithm, we will discuss about the experiments conducted with different dataset with duplicates ranging from 0 to 100% followed by results, analysis, observation and conclusion.

CHAPTER 2

BACKGROUND

Before moving onto the actual experiment, we will briefly discuss about the LEM2 algorithm and its implementation. LEM2 algorithm has been used in this project to derive the rule set for the given input data set [2].

2.1 LEM 2

An idea of blocks of attribute-value pairs is used in the rule induction algorithm LEM2 (Learning from Examples Module, version 2), another component of LERS. The option LEM2 of LERS is most frequently used since in most cases it gives better results. LEM2 explores the search space of attribute-value pairs. Its input data le is a lower or upper approximation of a concept (for definitions of lower and upper approximations of a concept see, e.g., [4]), so its input data le is always consistent. In general, LEM2 computes a local covering and then converts it into a rule set. We will quote a few definitions to describe the LEM2 algorithm [1], [3].

For an attribute-value pair $(a, v) = t$, a block of t, denoted by [t], is a set of all cases from U such that for attribute a have value v. Let B be a nonempty lower or upper approximation of a concept represented by a decision-value pair (d,w). Set B depends on a set T of attribute-value pairs $t=(a,v)$ if and only if:

$$\emptyset \neq [T] = \cap_{t \in T}[t] \subseteq B.$$

Set T is a minimal complex of B if and only if B depends on T and no proper subset T' of T exists such that B depends on T'. Let T be a nonempty collection of nonempty sets of attribute-value pairs. Then T is a *local covering* of B if and only if the following conditions are satisfied:

(1) each member T of T is a minimal complex of B,
(2) $\cup_{t \in T} [T] = B$, and

T is minimal, i.e., T has the smallest possible number of members.

The procedure LEM2 is presented below **Figure 2**.

Procedure LEM2
(**input**: a set B,
output: a single local covering $\mathcal{T}$ of set B);
begin
  $G := B$;
  $\mathcal{T} := \emptyset$;
  while $G \neq \emptyset$
    begin
    $T := \emptyset$;
    $T(G) := \{t \mid [t] \cap G \neq \emptyset\}$;
    while $T = \emptyset$ **or** $[T] \not\subseteq B$
      begin
        select a pair $t \in T(G)$ such that $|[t] \cap G|$ is maximum; if a tie occurs, select a pair $t \in T(G)$ with the smallest cardinality of $[t]$;
        if another tie occurs, select first pair;
        $T := T \cup \{t\}$;
        $G := [t] \cap G$;
        $T(G) := \{t \mid [t] \cap G \neq \emptyset\}$;
        $T(G) := T(G) - T$;
        end {while}
      for each $t \in T$ **do**
        if $[T - \{t\}] \subseteq$ B **then** $T := T - \{t\}$;
      $\mathcal{T} := \mathcal{T} \cup \{T\}$;
      $G := B - \cup_{T \in \mathcal{T}} [T]$;
  end {while};
  for each $T \in \mathcal{T}$ **do**
    if $\bigcup_{S \in \mathcal{T} - \{T\}} [S] = B$ **then** $\mathcal{T} := \mathcal{T} - \{T\}$;
end {procedure}.

Figure 2: LEM2 algorithm implementation

2.2 LEM 2 example problem

Table 1: Sample dataset

Case	Attributes				Decision
	Temperature	Headache	Weakness	Nausea	Flu
1	very_high	yes	yes	no	yes
2	high	yes	no	yes	yes
3	normal	no	no	no	no
4	normal	yes	yes	yes	yes
5	high	no	yes	no	yes
6	high	no	no	no	no
7	normal	no	yes	no	no

For a set X, |X| denotes the cardinality of X.

The first step of the algorithm LEM2 is to compute all attribute-value pair blocks.

For **Table 1** these blocks are:

[(Temperature, very_high)] = {1},

[(Temperature, high)] = {2, 5, 6},

[(Temperature, normal)] = {3, 4, 7},

[(Headache, yes)] = {1, 2, 4},

[(Headache, no)] = {3, 5, 6, 7},

[(Weakness, yes)] = {1, 4, 5, 7}

[(Weakness, no)] = {2, 3, 6},

[(Nausea, no)] = {1, 3, 5, 6, 7},

[(Nausea, yes)] = {2, 4}.

Let us induce rules for the concept {1, 2, 4, 5}. Hence, B = G = {1, 2, 4, 5}. The set T(G) of all relevant attribute-value pairs is

{(Temperature, very_high), (Temperature, high),

(Temperature, normal), (Headache, yes),

(Headache, no), (Weakness, yes),

(Weakness, no), (Nausea, no), (Nausea, yes)}.

The next step is to identify attribute-value pairs (a, v) with the largest $|[(a, v)] \cap G|$. For two attribute-value pairs from T(G), (Headache, yes) and (Weakness, yes), the cardinality of the set $|[(a, v)] \cap G|$ is equal to three. The next criterion is the size of the attribute-value pair block, this size is smaller for (Headache, yes) than for (Weakness, yes), so we select (Headache, yes). Besides, $[(\text{Headache, yes})] \subseteq B$, so (Headache, high) is the first minimal complex of G.

The new set G is equal to $B - [(\text{Headache, yes})] = \{1, 2, 4, 5\} - \{1, 2, 4\} = \{5\}$. A new set T(G) is equal to

{(Temperature, high), (Headache, no), (Weakness, yes), (Nausea, no)}.

This time the first criterion, the largest $|[(a, v)] \cap G|$, identifies all four attribute-value pairs. The second criterion, the size of the attribute-value block, selects (Temperature, high). However,

$$[(\text{Temperature, high})] = \{2, 5, 6\} \not\subseteq B$$

so we have to go through an additional iteration of the internal loop. The next candidates are (Headache, no) and (Weakness, yes), since for both of these attribute-value pairs the sizes of their blocks are equal to four. On the basis of heuristics, we will select (Headache, no). But

$$[(\text{Temperature, high})] \cap [(\text{Headache, no})] = \{5, 6\} \not\subseteq B = \{1, 2, 4, 5\},$$

so we have to add (Weakness, yes)] as well. This time

$$[(\text{Temperature, high})] \cap [(\text{Headache, no})] \cap [(\text{Weakness, yes})] = \{5\} \subseteq B = \{1, 2, 4, 5\},$$

so our candidate for a minimal complex is the set

$$\{(\text{Temperature, high}), (\text{Headache, no}), (\text{Weakness, yes})\}.$$

We have to run the following part of the LEM2 algorithm:

for each $t \in T$ do

if $[T - \{t\}] \subseteq B$ then $T := T - \{t\}$;

As a result, the second minimal complex is identified:

$$\{(\text{Temperature, high}), (\text{Weakness, yes})\}.$$

Eventually, the local covering of $B = \{1, 2, 4, 5\}$ is the set

$$\{\{(\text{Headache, yes})\}, \{(\text{Temperature, high}), (\text{Weakness, yes})\}\}$$

The complete rule set, induced by LEM2, is

(Headache, yes) → (Flu, yes)

(Temperature, high) & (Weakness, yes) → (Flu, yes)

(Temperature, normal) & (Headache, no) → (Flu, no)

(Headache, no) & (Weakness, no) → (Flu, no)

CHAPTER 3

EXPERIMENTS

This section explains about the use of LEM2 program, rule checker application, experimental setup and conducted experiments to obtain error rate.

3.1 Implementation of LEM2

The LEM2 algorithm is implemented using the C# language. The program will take input files in LERS format. Input data files will start from the list of variable declarations. This list starts from "<", then a space, then a sequence of following symbols: "a", "x", or "d" separated by spaces, again another space followed by the last symbol of the list ">". You may ignore this line. The second list of input data file starts from "[", then a space, then comes a list of attribute and decision names separated by spaces followed by another space and "]" symbol. The decision will be always the last variable and remaining all variables are attributes. The following part contains values of the attributes and decisions.

The input data file may contain comments. A comment starts from "!"; everything that follows "!" until the end of the line is the comment. Obviously, comments should be ignored during reading of data. Attribute values are separated by spaces. Spaces should be understood as not only ordinary spaces but also as a white space character, such as end of line, tab, etc. Any line of the input data file may start from one or more spaces and can end with one or more spaces.

After a valid input is given to LEM2 program, it generates the rule set and stores the rules in text file, which in turn is used as input file for the rule checker program. The algorithm is capable of processing any length of input data, but if the given dataset has more than 5000 lines then the program slows down a bit.

3.2 Implementation of rule checker

Rule checker program has been implemented using Java language. Java collections have been used immensely to get better performance. Rule checker takes 50% of randomly selected dataset and rule set (output from LEM2) as input. And error rate will be calculated for every rule in the rule set and the average will be taken. Average error rate is the output of rule checker program.

Basically error rate will be calculated with respect to each rule in the rule set. For each rule the valid cases will be selected from dataset based on the attribute signature, so that will be our 'Total cases' with respect to that rule. Among the 'Total cases', invalid cases are selected based on the mismatch with decision in the input dataset for the same attribute signature. The error rate is measured as:

$$\text{Error rate} = \frac{\text{Invalid cases}}{\text{Total cases}}$$

Once the error rate is calculated for each of rule in the final rule set, we will take average of it using the below formula:

$$\text{Average Error rate} = \frac{\sum \text{Error rate}}{\text{Number of rules in the Rule set}}$$

The average error rate is the output of rule checker program.

3.3 Experimental setup

This section explains about input file preparation, IDE used and environment setup.

3.3.1 Preparing input files

Eleven different input files were created for experiment purpose. The first file is with no duplicates and ten other files have duplicates ranging from 10% to 100%. All input files are in LERS format.

3.3.2 IDE and programming language

Java and C# was used to build rule checker and LEM2 respectively. Eclipse and visual studio 2013 IDE were used for development platform.

3.4 Experiments

Initially the given dataset with no duplicates is randomly divided into two parts. Hence after division we will have two parts of input dataset. First part of the input data set will be given as input to the LEM2 algorithm and the rule generated in LEM2 will be written into the text file and stored.

As a next step, the second part of data set will be given as an input to the rule checker program and stored in the program using hash map. Then each rule from the generated rule set will be picked and compared against the input dataset in order to calculate the value of error rate. This process is repeated for all obtained rules and the error rate will be recorded for each rule. Then at the end of program, average error rate will be calculated. This average error rate is the output of rule checker. At the end of first experiment the percentage of duplicates i.e. 0% and the average error rate is recorded.

The same experiment is repeated for each of the input dataset files which have duplicates ranging from 10% to 100%, so at the end of experiment we have eleven different values of average error rate for each input dataset.

While creating the duplicates files, entries from original data set were randomly selected and copied to the same data set. The actual size of the data set increased after the addition of duplicate entries. For an instance, to create a file with 40% duplicates, if assume there were 100 entries in the original data set, then 67 entries were randomly selected and copied back into the same data set. Finally, there were 167 entries in the data set out of which 67 entries were duplicate. Same procedure was repeated to create other required duplicate files.

CHAPTER 4

RESULTS & ANALYSIS

4.1 Results from experiments

After repeating the execution for eleven times with duplicates percentage of data in the input dataset ranging from 0 to 100%, we obtained eleven different values of average error rate for each input dataset.

4.1.1 Experiment 1

Table 2 gives the detail about the error rate for each input files with duplicates ranging from 0 to 100% in the input dataset.

Table 2: Error rate for different percentage of duplicates

Duplicates % in the input data set	Error rate
0	0.451
10	0.457
20	0.442
30	0.33
40	0.272
50	0.223
60	0.245
70	0.249
80	0.141
90	0.137
100	0.117

Figure 3 shows the graphical representation of average error rate plotted against the percentage of duplicates:

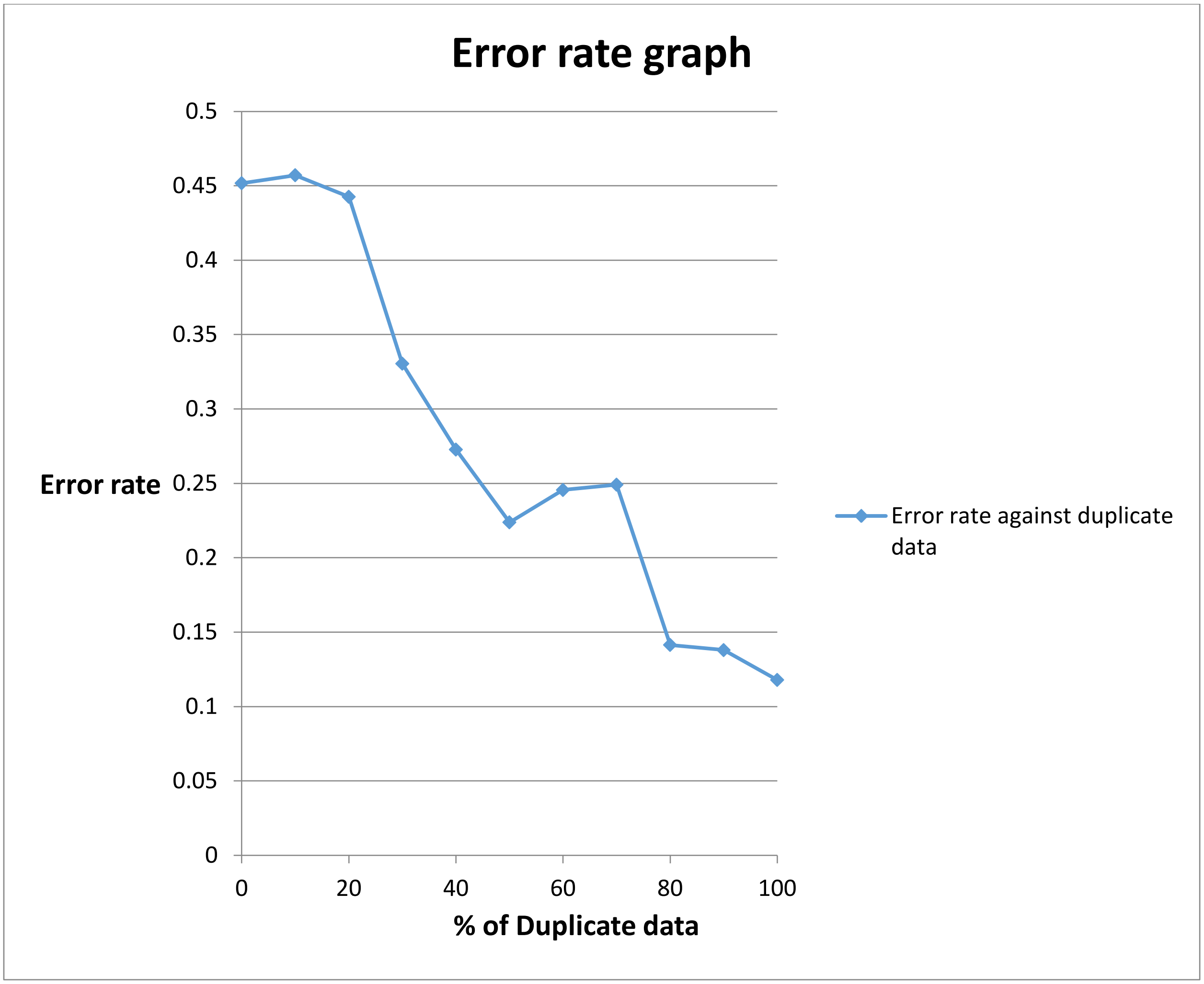

Figure 3: Error rate against the percentage of duplicates

4.1.2 Experiment 2

Table 3 gives the detail about the error rate for each input files with duplicates ranging from 0 to 100% in the input dataset.

Table 3: Error rate for different percentage of duplicates

Duplicates % in the input data set	Error rate
0	0.297
10	0.38
20	0.382
30	0.27
40	0.257
50	0.31
60	0.192
70	0.217
80	0.101
90	0.084
100	0.036

Figure 4 shows the graphical representation of average error rate plotted against the percentage of duplicates:

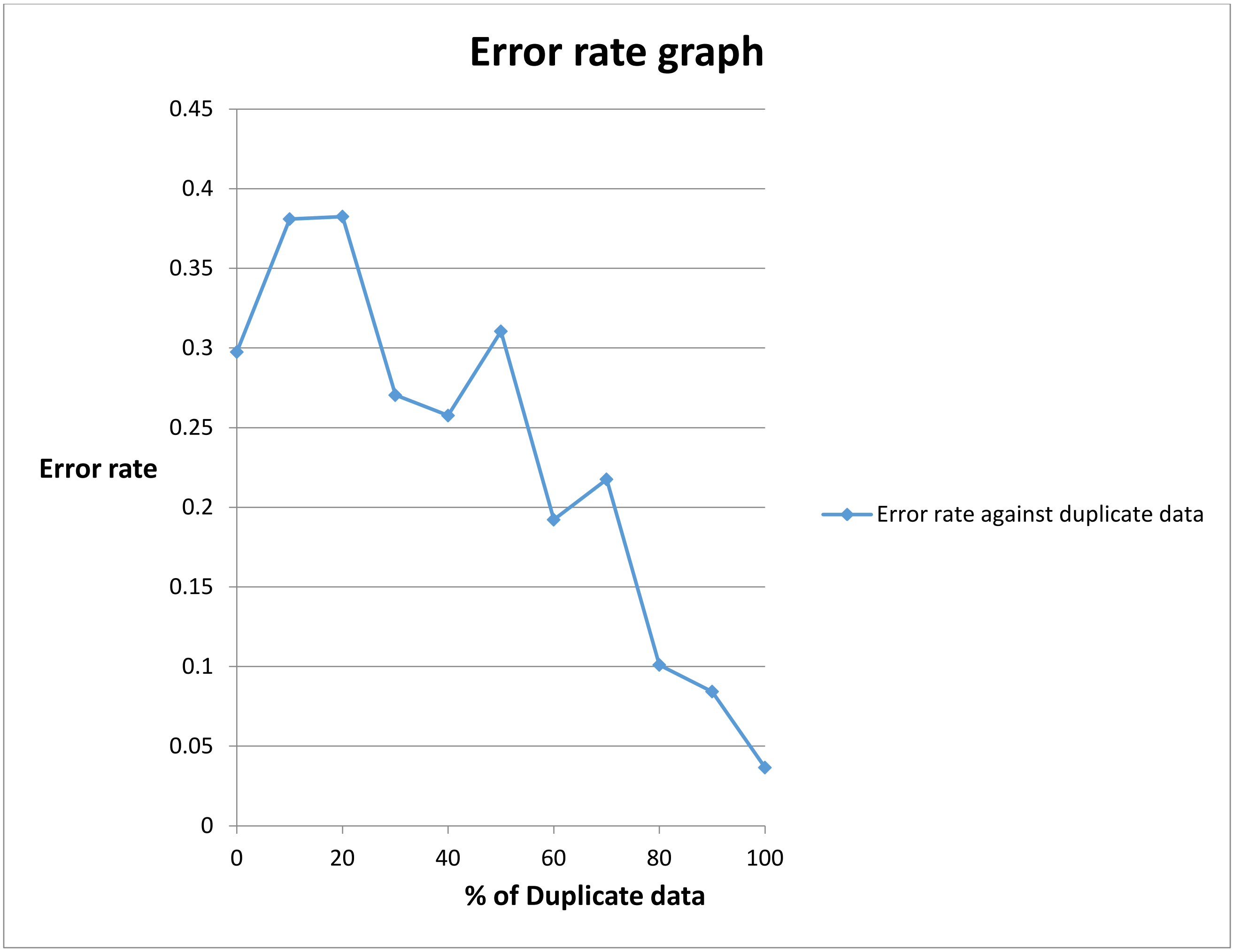

Figure 4: Error rate against the percentage of duplicates

4.1.3 Experiment 3

Table 4 gives the detail about the error rate for each input files with duplicates ranging from 0 to 100% in the input dataset.

Table 4: Error rate for different percentage of duplicates

Duplicates % in the input data set	Error rate
0	0.7151899
10	0.4285714
20	0.6229508
30	0.4206009
40	0.1628788
50	0.1444043
60	0.1743772
70	0.2421053
80	0.1515152
90	0.2030651
100	0.1747967

Figure 5 shows the graphical representation of average error rate plotted against the percentage of duplicates:

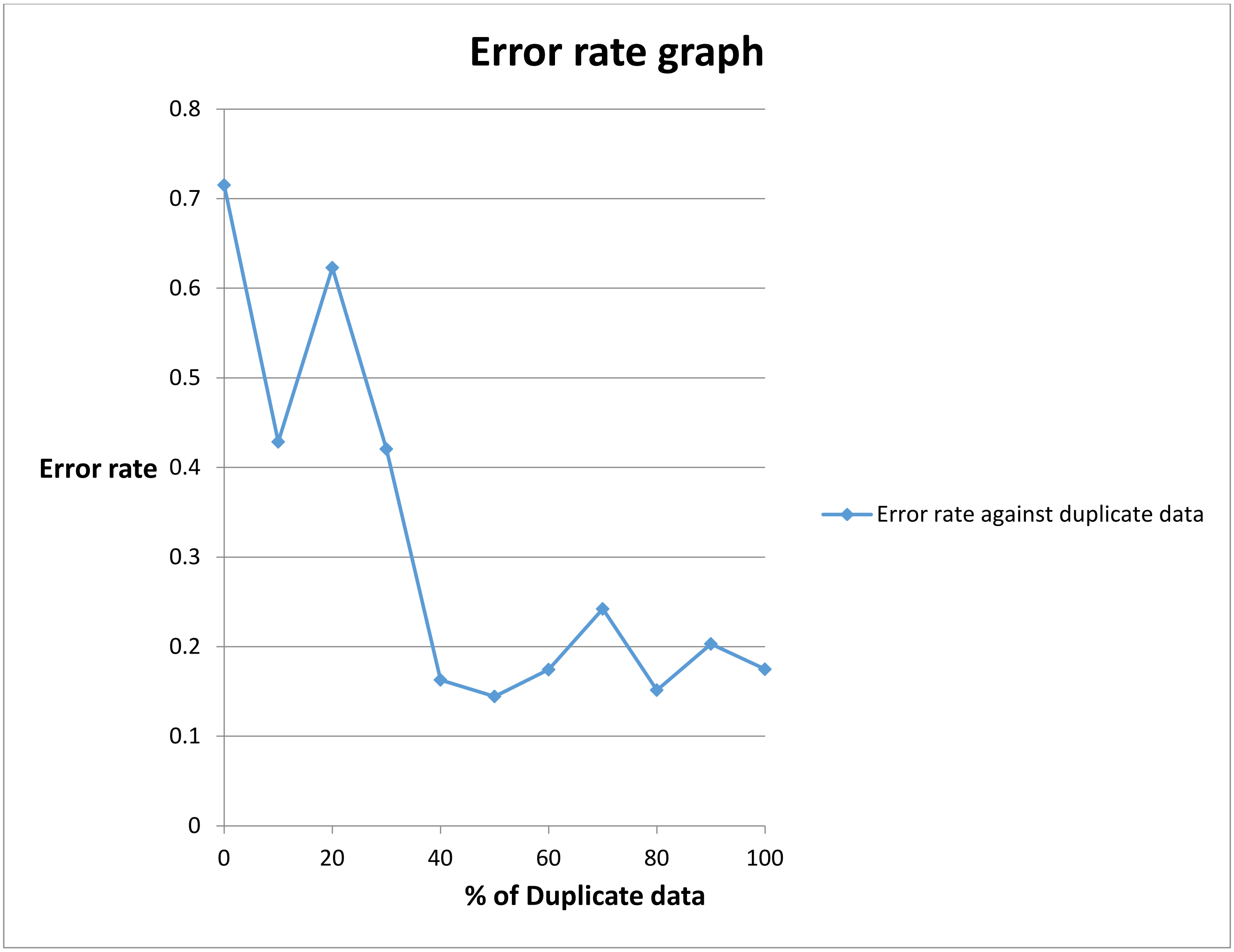

Figure 5: Error rate against the percentage of duplicates

4.2 Analysis

We can see in the graph that, as the percentage of duplicates is increasing the error rate is reducing. There are few spikes in the graph, they are because of the randomness in selection of input dataset, and sometimes there is a probability where maximum percentage of duplicate data will be put into the first part of input dataset (which was given as an input to LEM2 algorithm). So in second part of the dataset, lot many entries will not be in support of the rule generated by LEM2, therefore the irregular pattern in the error rate can be seen.

By ignoring the spikes in the graph caused due to randomness, the graph proves that with increase in percentage of duplicates the error rate reduces and the possibility of getting error rate as 1 or 0 is highly negligible and it is not guaranteed to get same result during all experiments since there is randomness in the selection of input dataset.

CHAPTER 5
CONCLUSION

The goal of this project is to check the effect of duplicates on the error rate. The experiments conducted demonstrate that with the increase in percentage of duplicates, the error rate is reducing. Duplicates play a huge role in deciding the generated rule's precision. The precision of the rule will be high if there are supporting entries for those rules in the dataset. But that's not a good sign, because duplicate data repeatedly states the same content. Though the error rate reduces with the duplicates it does not mean the predictions are perfect, in fact it's the other way round. The high precision is just due to supporting duplicates.

To have a better prediction system and to obtain best results in data mining we have to ignore or remove the duplicates from the dataset. There are a lot of techniques out there to remove the duplicates. Using the database as a source for input dataset is one of the good options where we can actually place a restriction in database for not storing the duplicates. To conclude, error rate reduces with the increase in duplicates but to get a good prediction or to make a best decision it’s better to remove duplicates from the input data.

REFERENCES

[1] [Chan and Grzymala-Busse, 1991] Chan C.C. and Grzymala-BusseJ.W. On the attribute redundancy and the learning programs ID3,PRISM, and LEM2. Department of Computer Science, University of Kansas, TR-91-14, December 1991, 20 pp.

[2] https://pdfs.semanticscholar.org/3cf4/1bb66e99e4155b92f1ea4d10af057dedf88a.pdf

[3] [Grzymala-Busse, 1992] Grzymala-Busse J.W. LERS—A system for learning from examples based on rough sets. In Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory, ed. by R. Slowinski, Kluwer Academic Publishers, Dordrecht, Boston, London, 1992, 3–18.

[4] [Grzymala-Busse, 1997] Grzymala-Busse J.W. A new version of the rule induction system LERS, Fundamenta Informaticae 1997; 31: 27–39.

APPENDIX

LEM2 code in C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Data;

namespace MiningSpecialData
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                bool mining=false, files=false, uop=false;
                string fileName=null;
                string outputPath = null;
                double alpha1=0.0f;
                string alpha=null;


               do
                {
                    Console.WriteLine("Please enter the Full path if the file(along with
                    the extension, EX: c:\\839-test-2.txt) \n ");
                    fileName = Console.ReadLine();
                    if (File.Exists(fileName))
                    {
                        files = true;

                    }
                    else
                    {
                        Console.WriteLine("File does not exists\n");
                    }


                } while (files==false);


                Console.WriteLine("\n***********************************************");
                int i = 0, j = 0, k = 0, col = 0, ro = 0,m=0,h=0;
                string path = Directory.GetCurrentDirectory();
                int norisk;

                int x = 0, y = 0,z=0;
                int bekitha = 0;
                bekitha = 4;
                StreamReader sr = new StreamReader(fileName);
```

```
        //  string data = sr.ReadLine();
        string data2 = sr.ReadToEnd();

        string[] values = data2.Split(' ','\t','\n','\r');
        var l = new List<string>() { };
    //    Console.WriteLine(values.Length);

        for (i = 0,j=0; i < values.Length; i++)
        {
            if (values[i] != "" && values[i] != "!")
            {
                l.Add(values[i]);
                j++;
            }
        }

        string[] realValues = l.ToArray();

        //calculating rows n column
        for (i = 0; realValues[i] != ">";i++ )
        {
            if (realValues[i] != "<")
            col++;
        }

        //4 - bcoz of <>[] and 2*col - for aa.....ad and wind,attr.....decision.
        ro = (realValues.Length - 4 - 2 * col) / col;


        string[] head = new string[col];
        DataTable table = new DataTable("ParentTable");

        i = 0;
        j = 0;

            while (realValues[i] != "[")
                i++;

        while (realValues[i] != "]")
        {

            if (realValues[i] != "[")
                head[j++] = realValues[i++];
            else
                i++;
        }


        for (i = 0; i < head.Length;i++ )
            table.Columns.Add(head[i], typeof(string));

        for (i = 0; i < ro; i++)
            table.Rows.Add();

        for (i = 0,k=(col + 2) * 2; i < ro; i++)
            for (j = 0; j < col; j++)
            {
               // Console.WriteLine(realValues[k]);
```

```
        table.Rows[i][j]=realValues[k++];

    }

//second phase

double dblValue = 0,insideLoop=0;

int[] colNum = new int[col];
double[] frNum = new double[ro];

/* Checking for numerical attribute*/
 for (j = 0; j < col; j++)
   for (i = 0; i < ro; i++)
    {
        if (Double.TryParse(table.Rows[i][j].ToString(), out dblValue))
        {
            colNum[j] = 1;
            break;

        }

        else if(i==ro-1)
        {
            colNum[j] = 0;
        }

    }

 DataTable table2 = new DataTable("SecondTable");
 table2 = table;

 List<double> g = new List<double>() { };
 List<double> avg = new List<double>() { };
 List<double> ele = new List<double>() { };
 List<double>[] cutPoints = new List<double>[col-1];
 List<double>[] distinctValue = new List<double>[col - 1];
 List<int[]> eleReq = new List<int[]>() { };
 List<string> str = new List<string>() { };
 List<string> str1 = new List<string>() { };

 k = 2;

 var UniqDecision = (from DataRow dr in table.Rows
             orderby (string)dr[head[head.Length-1]]
                     select (string)dr[head[head.Length -
                                       1]]).Distinct();

 List<string> uD = new List<string>() { };

 uD = UniqDecision.ToList();


 for (j = 0; j < col-1; j++)
     {
         cutPoints[j] = new List<double>();
         distinctValue[j] = new List<double>();
     //to check whether the column is numerical, if so enter
```

```
            if (colNum[j] == 1)
            {
                for (i = 0; i < ro; i++)
                {
                    //retreiving the value from table
                   if (Double.TryParse(table.Rows[i][j].ToString(), out
                                                    dblValue))
                    {
                        g.Add(dblValue);
                        distinctValue[j].Add(dblValue);
                    }

                }

                //getting distinct value
                distinctValue[j] = distinctValue[j].Distinct().ToList();
                g = g.Distinct().ToList();

                //sorting it
                distinctValue[j].Sort();
                g.Sort();



                //cut points
                for (i = 0; i < g.Count - 1; i++)
                {
                    avg.Add((g[i] + g[i + 1]) / 2);
                    cutPoints[j].Add((g[i] + g[i + 1]) / 2);
                }
                g.Clear();
            }

        }

   //Actual algorithm starts

    bool nConsistent = true;
/*   double[] entropyvalue*/
    List<double> entropy = new List<double>() { };
    List<int> vv = new List<int>() { };
    List<int>[][] mainCount = new List<int>[col - 1][];


    multipleScanning firstClss = new multipleScanning();

    firstClss.MutipleScanning(table,distinctValue,cutPoints,ro,col,head,uD);


   while(nConsistent)
   {
       //calculate entropy
       for(i=0;i<table2.Columns.Count-1;i++)
       {
           for(int tic1=0;tic1<distinctValue[i].Count;tic1++)
           {

               mainCount[tic1] = new List<int>[uD.Count];
```

```
            for(int tic2=0;tic2<UniqDecision.Count();tic2++)

            {

                var query = from DataRow dr in table2.Rows
                            where dr[head[i]] ==
                            distinctValue[i][tic1].ToString() &&
                            dr[head[col]] == uD[tic2].ToString()
                            select (string)dr[head[i]];

            int count1 = 0;
            count1 =
            (from DataRow dr in table2.Rows
             where dr[head[i]] == distinctValue[i][tic1].ToString() &&
              dr[head[col]] == uD[tic2].ToString()
             select (string)dr[head[i]]).Count();

            mainCount[tic1][tic2].Add(count1);

            }

        }
    }

}



 for (j = 0; j < ro; j++)
 {
     //retreiving the value from table

     str.Add(table.Rows[j][col-1].ToString());

 }

 str = str.Distinct().ToList();
 List<int>[] goal = new List<int>[str.Count()];

 for (i = 0; i < str.Count();i++)
 {
     goal[i] = new List<int>();

         for (j = 0; j < ro; j++)
       {
     //retreiving the value from table

          if(str[i].ToString()==table.Rows[j][col-1].ToString())
              {
                  goal[i].Add(j);
              }

       }
 }

 List<int>[] MainTerm = new List<int>[h];
 for (i = 0; i < h;i++ )
```

```
{
    MainTerm[i] = new List<int>();

    for(j=1;j<=ro;j++)
    {
        if(j==140)
        {
            int jjjj = 10;
        }
        try{
             if(Int32.Parse(table2.Rows[i][j].ToString())>=0)
        {
            MainTerm[i].Add(j-1);
        }
        }
            catch
        {
            continue;
            }


    }
}

List<int>[] uGoal = new List<int>[str.Count()];
int Cond1=0;
int sel = 0;

for (int rpt = 0; rpt < str.Count(); rpt++)
{

    uGoal[rpt] = goal[rpt];
    List<int> dVar = new List<int>() { };
    dVar=goal[rpt];
    List<int>[] inTerms = new List<int>[h];

    List<int> fatT = new List<int>() { };
    List<int> fatJ = new List<int>() { };
    List<string> allT = new List<string> { };

    //while goal not equal to null
    while(uGoal[rpt].Count>0)
    {
        //T
        List<int> TVar = new List<int>() { };
        List<int> SVar = new List<int>() { };
        List<int> NVar = new List<int>() { };
        List<string> Tstr = new List<string>() { };
        int high = 0;

        //t n G
        for(i=0;i<h;i++)
        {
            inTerms[i] = uGoal[rpt].Intersect(MainTerm[i]).ToList();
            if (inTerms[i].Count() > 0)
            {
                Cond1 = 1;
                if (high < inTerms[i].Count())
```

```
            {
                high = inTerms[i].Count();
                sel = i;
            }
        }


    }

    var cond3 = dVar;
    bool condition4 = false;

    while (Cond1 == 1 &&  !(TVar.SequenceEqual(cond3)))
    {
        //selecting best pair
        //improvement here
        for (i = 0; i < h; i++)
        {
            if (high == inTerms[i].Count() &&
             MainTerm[sel].Count()>MainTerm[i].Count())
            {
                sel = i;
            }
        }

        Tstr.Add(table2.Rows[sel][0].ToString());

        //G n {t}
        uGoal[rpt] = uGoal[rpt].Intersect(inTerms[sel]).ToList();

        if (TVar.Count() == 0)
        {
            TVar = MainTerm[sel];
        }
        else
        {

              TVar = MainTerm[sel].Intersect(TVar).ToList();
        }

        if(table2.Rows[sel][ro+2].ToString()=="N")
        {

            //improvement here
            NVar.Add(sel);
           /* int something = 0;

        }
        else
        {


            for(i = 0; i < h; i++)
            {
                if(table2.Rows[sel][ro +
                    1].ToString()==table2.Rows[i][ro +
                    1].ToString())
                {
```

```
                    SVar.Add(i);
                }
            }

        }

        for(int iml=0;iml<SVar.Count();iml++)
        {
            inTerms[SVar[iml]].Clear();
        }

        for (int ikl = 0; ikl < NVar.Count(); ikl++)
        {
            inTerms[NVar[ikl]].Clear();
        }


        high = 0;
        Cond1 = 0;
        for (i = 0; i < h; i++)
        {
            inTerms[i] = uGoal[rpt].Intersect(inTerms[i]).ToList();
            if (inTerms[i].Count() > 0)
            {
                //here also cardinality
                Cond1 = 1;
                if (high < inTerms[i].Count())
                {
                    high = inTerms[i].Count();
                    sel = i;
                }
            }

        }


        cond3 = TVar.Intersect(dVar).ToList();
      /*  if(TVar.SequenceEqual(cond3))
        {
            uGoal[rpt] = goal[rpt].Except(cond3).ToList();
            condition4 = false;
        }*/


    }

    double prob = 0.0f;
    double first = 0.0f;
    double second = 0.0f;


    var condf = TVar.Intersect(goal[rpt]).ToList();
    first = System.Convert.ToDouble(condf.Count());
    second = System.Convert.ToDouble(TVar.Count());
    prob = first / second;


    if (prob >= alpha1)
```

```
    {
        dVar = dVar.Union(TVar).ToList();
        fatT = fatT.Union(TVar).ToList();

        for(i=0;i<Tstr.Count();i++)
        {
            allT.Add(Tstr[i]);
        }
        allT.Add("delim");

    }
    else
    {
        fatJ = fatJ.Union(TVar).ToList();
    }

    uGoal[rpt] = dVar.Except(fatJ.Union(fatT).ToList()).ToList();

}

int Waste = 0;
for(i=0;i<allT.Count();i++)
{
    if (allT[i] == "delim")
        Waste++;
}

List<string>[] tnew= new List<string>[Waste];

for (i = 0; i < Waste;i++)
{
    tnew[i] = new List<string>();
}

int counter = 0;
i = 0;

while(i<allT.Count())
{
    while(allT[i] != "delim" && counter<Waste)
    {
        tnew[counter].Add(allT[i].ToString());
        i++;
    }
    counter++;
    i++;
}

//redundancy
int[] waste1 = new int[]{};
int[] waste2 = new int[99999];

List<int>[] Rafterredudncy = new List<int>[9999];
for(int jump=0;jump<9999;jump++)
{
    Rafterredudncy[jump] = new List<int>();
    Rafterredudncy[jump] = (waste1.ToArray()).ToList();
}
```

```
bool yako = false;
int overAll = 0;

for(i=0;i<tnew.Count();i++)
{


    yako = false;
    List<string> DupI = new List<string> { };

    DupI=tnew[i].ToList();


    //corresponding main table values

    List<int>[] ValN = new List<int> [DupI.Count()];
    for (j = 0; j < tnew[i].Count(); j++)
    {
        ValN[j] = new List<int>();
        for (k = 0; k < h; k++)
        {
            if(DupI[j].ToString()==table2.Rows[k][0].ToString())
            {
                ValN[j] = MainTerm[k];
            }
        }

    }

    List<int>[] DupVal = ValN;
    List<int>[] CurrentSet = new List<int>[DupVal.Count()];
    List<int> currentValue = new List<int> { };
    List<int> subSet = new List<int> { };
    List<int> index = new List<int> { };

    int[] skiprow = new int[DupVal.Count()];
    for(int ii=0;ii<DupVal.Count();ii++)
    {
        skiprow[ii] = 1;
    }
    int need = 0;
    List<int>[] JustCopy = new List<int>[DupVal.Count() - 1];



    //else maribeda
    if (DupVal.Count() != 1)
    {


        for (j = 0; j < DupVal.Count(); j++)
        {

            if (DupVal.Count() == 2)
            {
```

```
subSet = DupVal[j + 1];
if (!subSet.Except(dVar).Any())
{
    JustCopy[need] = DupVal[j + 1];
    DupVal = JustCopy;
    yako = true;
    break;
}

subSet = DupVal[j];
if (!subSet.Except(dVar).Any())
{
    JustCopy[need] = DupVal[j];
    DupVal = JustCopy;
    yako = true;
    break;
}


break;
}
else
{
        skiprow[j] = 0;
        currentValue = DupVal[j];
        for (int dump = 0; dump < DupVal.Count();
                                 dump++)
            CurrentSet[dump] =
           (waste1.ToArray()).ToList();

        bekitha = 0;
        for (k = 0; k < DupVal.Count(); k++)
        {
            if (skiprow[k] == 1)
            {
                CurrentSet[bekitha] = new List<int>();
                CurrentSet[bekitha] =
                DupVal[k].ToList();
                bekitha++;
            }
        }

       int kkk=0;
       for(int ii=0;ii<DupVal.Count();ii++)
               {
                   if(skiprow[ii] == 1)
                       kkk++;
               }
       if(kkk==0)
       {
           skiprow[j] = 1;
           break;
       }
        //CurrentSet = DupVal.Except(DupVal);

        if (bekitha == 0)
        {
```

```
                    break;
                }
                else if (bekitha == 1)
                {
                    subSet = CurrentSet[0];
                    if (!subSet.Except(dVar).Any())
                    {
                        skiprow[j] = 0;
                    }
                    else
                    {
                        skiprow[j] = 1;
                    }
                }
                else if (bekitha == 2)
                {
                    subSet =
                CurrentSet[0].Intersect(CurrentSet[1]).ToList();
                    if (!subSet.Except(dVar).Any())
                    {
                        skiprow[j] = 0;
                    }
                    else
                    {
                        skiprow[j] = 1;
                    }
                }
                else
                {
                    subSet = CurrentSet[0];
                    for (k = 0; k < bekitha - 1; k++)
                    {

                        subSet = subSet.Intersect(CurrentSet[k +
                       1]).ToList();

                    }
                }

                if (!subSet.Except(dVar).Any())
                {
                    skiprow[j] = 0;

                }
                else
                {
                    skiprow[j] = 1;
                }


        }
    }

    int bekebeku=0;
    if (yako == false)
    {
```

```
            int hereOnly = 0;
            for (k = 0; k < DupVal.Count(); k++)
            {
                if (skiprow[k] == 1 && CurrentSet.Count()!=0)
                {
                    CurrentSet[hereOnly] = DupVal[k].ToList();
                    hereOnly++;
                }
            }

            bekebeku = hereOnly;
            DupVal = CurrentSet;

            for (k = 0; k < bekebeku; k++)
            {
                Rafterredudncy[overAll] = new List<int>();
                Rafterredudncy[overAll] = DupVal[k].ToList();
                overAll++;
            }

            Rafterredudncy[overAll] = new List<int>();
            Rafterredudncy[overAll] = (waste1.ToArray()).ToList();
            overAll++;
        }


        else
        {

            Rafterredudncy[overAll] = new List<int>();
            Rafterredudncy[overAll] = DupVal[0].ToList();
            overAll++;

            Rafterredudncy[overAll] = new List<int>();
            Rafterredudncy[overAll] = (waste1.ToArray()).ToList();
            overAll++;
        }
    }

    else
    {
        Rafterredudncy[overAll] = new List<int>();
        Rafterredudncy[overAll] = DupVal[0].ToList();
        overAll++;

        Rafterredudncy[overAll] = new List<int>();
        Rafterredudncy[overAll] = (waste1.ToArray()).ToList();
        overAll++;
    }

Rafterredudncy[overAll] = new List<int>();
Rafterredudncy[overAll] = (waste2.ToArray()).ToList();
int[] waste3 = new int[999999];
int numberofrules = 0;
List<int>[] Rafterredudncy1 = new List<int>[overAll];
for (k = 0; k < overAll; k++)
{
    Rafterredudncy1[k] = Rafterredudncy[k];
```

```
                        if( Rafterredudncy1[k].Count()==0)
                        {
                            numberofrules++;
                        }
                    }

                    for (k = 0; k < overAll; k++)
                      {
                          if (Rafterredudncy1[k].Count() != 0)
                          {
                              for (norisk = k+1; norisk < overAll; norisk++)
                              {
                                  if (Rafterredudncy1[k].SequenceEqual
                                      (Rafterredudncy1[norisk]))
                                      Rafterredudncy1[k] =
                                      (waste3.ToArray()).ToList();
                              }
                          }
                      }

string rule = null;
string ruleConcept = null;

Console.WriteLine("\nRule set  for the concept " + "(" + head[head.Length - 1] + "," +
str[rpt] + ")");
                    ruleConcept="Rule set  for the concept " + "(" + head[head.Length -
1] + "," + str[rpt] + ")\n";


                    if (!File.Exists(outputPath))
                    {
                        File.Create(outputPath);
                        TextWriter tw = new StreamWriter(outputPath);
                        tw.WriteLine(ruleConcept);
                        tw.Close();
                    }
                    else if (File.Exists(outputPath))
                    {
                        TextWriter tw = new StreamWriter(outputPath,true);
                        tw.WriteLine(ruleConcept);
                        tw.Close();
                    }

                    bool enoOndhu = false;
                    for(k=0;k<overAll;k++)
                    {
                        rule = null;
                        do
                        {
                            if(rule!=null)
                            {
                                rule = rule + "&";
                            }
                            for (int yappa = 0; yappa < h; yappa++)
                            {
                                if(Rafterredudncy1[k].SequenceEqual(MainTerm[yappa]))
                                {
```

```
                                rule = rule + table2.Rows[yappa][0].ToString();
                                enoOndhu = true;
                                break;
                            }
                        }

                        k++;
                        if (k >= overAll)
                            break;


                    } while ((Rafterredudncy1[k].Count() != 0));

                    if (k != 0 && enoOndhu==true)
                    {
                        enoOndhu = false;
                        rule = rule + "->" + "(" + head[head.Length - 1] + "," +
                               str[rpt] + ")\n";
                        if (!File.Exists(outputPath))
                        {
                            File.Create(outputPath);
                            TextWriter tw = new StreamWriter(outputPath);
                            tw.WriteLine(rule);
                            tw.Close();
                        }
                        else if (File.Exists(outputPath))
                        {
                            TextWriter tw = new StreamWriter(outputPath, true);
                            tw.WriteLine(rule);
                            tw.Close();
                        }

                    }

                    Console.WriteLine("{0}", rule);

                }

            }

            if (File.Exists(outputPath))
            {
Console.WriteLine("\n**************************************************\n");
                    Console.WriteLine("Please find the output file at : .\t" +"'"+
outputPath+"'"+"\n\n\n");
                    Console.WriteLine("NOTE:(Refer this NOTE only if you are not able to
find the file in the above mentioned location, this NOTE remains invalid if the file is
found in above location(Safety Net)): If the given output path directory does not exists
then by default the ouput file will be created at " +
Directory.GetCurrentDirectory()+"\\" + outputPath+".txt");
            }

            Console.WriteLine("\nPress enter to exit");
           Console.ReadLine();
        }

        catch(Exception ex)
```

```
            {
                Console.WriteLine("Exception is {0}", ex);
                Console.ReadLine();

            }


        }
    }
}
```

Rule checker :

```
import java.util.ArrayList;
import java.util.Iterator;


public class RuleChecker {

	public float errorRate(ReadingInput inpInst,ReadingResults resInst)
	{
		float errorRate=0.0f;
		int totalCase=0;
		int errorCases=0;
		int ruleLength=0;
		ArrayList<String> notPresent=new ArrayList<String>();
		ArrayList<String> finalInputSet=inpInst.completeInput;

		String rule1=resInst.rule.toString();
		String[] individualInput= rule1.split("-");
		ruleLength=individualInput.length;
		System.out.println(individualInput[0]);

		//To count the all possible combination of the given input rules attributes
		for(int i=0;i<ruleLength-1;i++)
		{
			notPresent.clear();
			Iterator<String> itr= finalInputSet.iterator();
			while(itr.hasNext())
			{
				String rule=itr.next();

				if(!rule.contains(individualInput[i]))
				{
					notPresent.add(rule);
				}
```

```
				}

				finalInputSet.removeAll(notPresent);
			}

			totalCase=finalInputSet.size();

			//To count the valid cases as per the decision
			Iterator<String> itr= finalInputSet.iterator();
			while(itr.hasNext())
			{
				String rule=itr.next();
				if(!rule.contains(individualInput[ruleLength-1]))
				{
					errorCases++;
				}
			}

			//Calculating the error rate
			errorRate=(float)errorCases/totalCase;
			System.out.println(errorRate);
			return errorRate;
		}


}
```

Reading input:

```
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;


public class ReadingInput {

	public int attCount;
	public ArrayList<String> attributes;
	public ArrayList<String> decision;
	Map<String,Integer> attriCount= new HashMap<String,Integer>();
	Map<String,Integer> decCount= new HashMap<String,Integer>();
	public ArrayList<String> completeInput;
```

```
public ReadingInput()
{
        attCount        = 0;
        attributes      = new ArrayList<String>();
        decision        = new ArrayList<String>();
        completeInput = new ArrayList<String>();
}

public void readingInputTextFile(String filename)
{
        Scanner sc2 = null;
        try
        {
                sc2 = new Scanner(new File(filename));

                //To get attribute count
                if(sc2.hasNextLine())
                {
                        Scanner s2 = new Scanner(sc2.nextLine());
                        while (s2.hasNext()) {
                String s = s2.next();
                if(s.equals("a"))
                        attCount++;
              }
                        s2.close();
                }

                //To store attributes and Decision
                if(sc2.hasNextLine())
                {
                        int curCount=0;
                        Scanner s2 = new Scanner(sc2.nextLine());
                        while (s2.hasNext()) {
                String s = s2.next();
                if(curCount==attCount)
                {
                                decision.add(s);
                                System.out.println("Decison:" + s);
                                curCount++;
                }
                else if(!(s.equals("[")||s.equals("]")))
                {
                                 attributes.add(s);
                                 System.out.println("Attribute: " +s);
                                 curCount++;
```

```
        }

      }
                  s2.close();
        }


        //to store complete rule
         while (sc2.hasNextLine()) {
        Scanner s2 = new Scanner(sc2.nextLine());
        int count=0;
        StringBuffer cRule= new StringBuffer();
        boolean firstCheck=true;
      while (s2.hasNext()) {
        String s = s2.next();

        if(count==attCount)
        {

                        StringBuffer str=new StringBuffer();
                        str.append("|"+decision.get(0)+":"+s);

                        decCount.put(str.toString(),1);

                  cRule.append(str);
        }
        else
        {

                        StringBuffer str=new StringBuffer();
                        str.append(attributes.get(count)+":"+s);

                        if(firstCheck)
                        {
                              cRule.append(attributes.get(count)+":"+s);

                              firstCheck=false;
                        }
                        else
                        {
cRule.append("|"+attributes.get(count)+":"+s);
                        }


                        if(attriCount.containsKey(str.toString()))
```

```
                                        {
                                                attriCount.put(str.toString(),
attriCount.get(str.toString())+1);
                                        }
                                        else
                                        {
                                                attriCount.put(str.toString(),1);
                                        }

                              count++;
                         }



                    }

                    completeInput.add(cRule.toString());
                    s2.close();
                     }

                      Set<String> keys= attriCount.keySet();
                      for(String key:keys)
                      {
                              System.out.println(key+":--"+ attriCount.get(key));


                      }
                      System.out.println(completeInput);

          }catch(Exception ex){
             System.out.println(ex);
          }
                finally
                {
                      sc2.close();

                }

     }
}
```

Reading Results:

```
import java.io.File;
import java.util.ArrayList;
import java.util.Scanner;
```

```
public class ReadingResults {

	public ArrayList<String> singleRule;
	public String rule;


	ReadingResults()
	{
		singleRule=new ArrayList<String>();
		rule=null;
	}

	public void storeRule(String filename)
	{
			Scanner sc2 = null;
			try
			{
				sc2 = new Scanner(new File(filename));

				//To get attribute count
				while(sc2.hasNextLine())
				{
					rule=sc2.nextLine().toString();
					singleRule.add(rule);

				}

				System.out.println(singleRule);
				//System.out.println(rule);
			}
			catch(Exception ex){
		System.out.println(ex);
			}
	}



}
```