EAST WEST UNIVERSITY

Map-Centric Belief Rule Based System to Assess Dyslexia with Uncertainty

Submitted By:

Sarder Md. Naim Uddin

ID: 2012-1-68-013

Md. Ismile Hossain

ID: 2012-2-60-006

Supervised By:

Md. Sarwar Kamal

Senior Lecturer

Department of Computer Science and Engineering

East West University

A Project submitted in partial fulfillment of the requirements for the degree of Bachelors of Science in Computer Science and Engineering to the Department of Computer Science and Engineering

December 2016

Abstract

Dyslexia is a serious problem on students in all respects. During this digitalization era, competitions are increasing day by day with enormous stress. Automated tool will be helpful to predict the dyslexia symptoms. Machine learning techniques with artificial intelligences are significant to develop the framework. However, excessive datasets on current medical diagnosis is a big challenge to design the system with numerous uncertainties. Map-Centric Belief Rule Based system (MapBRB) is an integrated framework that handles large datasets as well as predicts the symptoms of dyslexia at tertiary level students. Map-Centric phase clean the noises and irregularities from collected datasets. This process constructed by two pivotal parts as mapping and reducing. Mapping organizes the datasets and reducing part removes the irrelevant factors. Followed by Map-Centric, BRB operates the with inference engine (IE) and evidential reasoning(ER). During this experiment, uncertainty factors for dyslexic students are also checked and addressed. Few factors of dyslexia symptoms lead vague meaning and confusing. In this regards, BRB process the vague data with appropriate dimensions. Both qualitative and quantitative datasets have been addressed. Substantial comparisons have been performed among MapBRB and mapping less fuzzy system, mapping less BRB system and mapping less prediction system.

Declaration

We hereby declare that, this project was done under CSE497 and has not been submitted elsewhere for requirement of any degree or for any reason except for publication.

_____

Sarder Md. Naim Uddin

ID: 2012-1-68-013

Department of Computer Science and Engineering

East West University

_____

Md. Ismile Hossain

ID: 2012-2-60-006

Department of Computer Science and Engineering

East West University

Letter of Acceptance

The thesis is submitted by Sarder Md. Naim Uddin, ID: 2012-1-68-013 and Md. Ismile Hossain, ID: 2012-2-60-006 to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted as satisfactory for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Engineering on December 8th , 2016.

_____

Sarwar Kamal

Senior Lecturer **(Supervisor)**

Department of Computer Science and Engineering

East West University

_____

Dr. Mozammel Huq Azad Khan

Professor and Chairperson

Department of Computer Science and Engineering

East West University

Acknowledgement

First of all, we would like to thank almighty Allah for giving us the strength and proper knowledge to complete my thesis work.

We would like to thank our honorable teacher and supervisor, Sarwar Kamal, Senior Lecturer, Department of Computer Science and Engineering, East West University for his at most direction, encouragement, kind guidance, sharing knowledge and constant inspiration throughout this thesis work.

Our deep gratefulness for all faculty members of Computer Science and Engineering whose friendly attitude and enthusiastic that has given us for four years.

Finally, we would like to convey our special thanks to our parents whom have always given us tremendous support. Without their love and encouragement, we would not had achieve this far.

Table of Contents

Contents

Chapter 4: Implementation & Result

Chapter 5: Conclusion & Future Work

List of Figures

Chapter 1
Introduction

1.1 Overview

Dyslexia is a language disability, affecting reading, writing, speaking and listening. It affects students in all respect. Students with dyslexia take a long time to read each word, re-read paragraph to understand them, difficulties to find the right word to say, cannot think of creative solution to problem, reading out loud is embarrassing etc. Symptoms are varying in degree from person to person. Our system will be helpful to predict the dyslexia symptoms. We collect data from students on different types of symptoms. Apply Map-Centric on those collected data. Map-Centric is constructed by two pivotal parts as mapping and reducing. Mapping organizes the datasets and reducing part remove the noise and irregularities from collected datasets. Finally apply Belief Rule Base on those significant data to predict the symptoms of dyslexia at tertiary level students. During this experiment, uncertainty factors for dyslexic students are also checked and addressed.

1.2 Objective of the Research

The objective of the thesis is to develop a system that can predict the probability of having Dyslexia with which category of severity level by using MapCentric and Belief Rule Base.

1.3 Methodology of the research

For faster and parallel data processing MapReduce is very useful. The most popular framework for MapReduce is Hadoop. But we don't use Hadoop framework for MapReduce rather we use MapReduce concept to implement Mapper and Reducer function. We also use Belief Rule which allows the capturing of uncertain information. As a result Belief Rule processes the vague data with appropriate dimensions.

1.4 Result of the Research

Our system is constructed with MapCentric Belief Rule Based system (MapBRB) which is an integrated framework that can predict the probable percentage of having dyslexia by analysis of the previous datasets. The system can also tell which category severity level the user is affected by Dyslexia.

1.5 Outline

i. Chapter 2, which contain a short description about Dyslexia, MapCentric and Belief Rule techniques.
ii. Chapter 3 contains the architecture of the system.
iii. Chapter 4 covers the brief description of each component related with the system.
iv. Chapter 5 is about conclusion and future work.

Chapter 2
Background Study

2.1 What is Dyslexia?

The word Dyslexia comes from Greek and means ‘difficulty with words’. Dyslexia is a language based learning disability. Dyslexia refers different types of symptoms, which result students with dyslexia usually experience difficulties with language skills such as reading, writing, spelling, listening and pronouncing words. Symptoms are varying in degree from person to person. Generally, dyslexia can be said to be a processing problem. This means that a dyslexic brain processes information differently from a non-dyslexic brain. The ability to reading, writing, spelling, listening and pronouncing words can be significantly affected by this processing difference. The International Dyslexia Association estimates that 15-20 percent of the American population have some of the symptoms of dyslexia, including slow or inaccurate reading, poor spelling, poor writing, or mixing up similar words.

2.1.1 Symptoms of Dyslexia

People with dyslexia can have mild to severe impairment. Signs of the condition vary widely from person to person. People with dyslexia may have the following signs and symptoms:

- **Visual confusion:** Mixing up similar words such as "was" and "saw", "cat" and "cot".
- **Short term memory:** Dyslexics often have severe short term memory problems. Like: can read a word on one page, but won't recognize it on the next page.
- **Coordination problems:** Dyslexics can sometimes suffer from physical issues such as clumsiness, problems with word pronunciation etc.
- **Left-right confusion:** Dyslexics people commonly gets "left" and "right" mixed up. Like: b–d confusion is a classic warning sign. One points to the left, the other points to the right, and they are left–right confused.
- **Up-down confusion:** b–p, n–u, or m–w confusion. One points up, the other points down and they are up-down confused.

- **Reading problem:** Re-read paragraphs to understand them, difficult to recite the alphabet, embarrassing to read aloud, loose place or miss out lines when reading.

- **Math difficulties:** Memorizing addition, subtraction facts and multiplication tables, remembering the sequence of steps in long division.

- **North, South, East, West confusion:** People with dyslexia get lost when driving around, even in cities where they've lived for many years. Often have difficulty reading or understanding maps.

- **Directionality problems:** Get confused when given several instructions at once. Confusion about sequence of direction such as first–last, before–after, next–previous, over–under.

- **Writing and speaking problems:** When writing, it is difficult to organize thoughts on the paper. When speaking face difficulties to find out the right word to say.

- **Others difficulties:** Cannot think of creative solutions for a specific problem. Get confuse the names of objects, for example table and chair.

2.1.2 Causes of Dyslexia

Researchers have found that dyslexia is caused by a difference in the way the dyslexic brain processes information. Experts do not know precisely what causes dyslexia, but several recent studies now indicate that a gene on the short arm of chromosome #6 is responsible for dyslexia. That gene is dominant, making dyslexia highly heritable. As a result Dyslexia often runs in families.

.

2.2 MapReduce

MapReduce is a programming model suitable for processing of huge amount of structured and unstructured data, stored in the Hadoop Distributed File System (HDFS). Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, C++ and C#. The implementation of MapReduce is combined with several systems, including Google's internal implementation. The popular open implementation of Hadoop which can be obtained along with the HDFS file system from Apache Foundation.

MapReduce can process huge data, with the help of two functions: Map and Reduce. A MapReduce job usually splits the input dataset into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system.

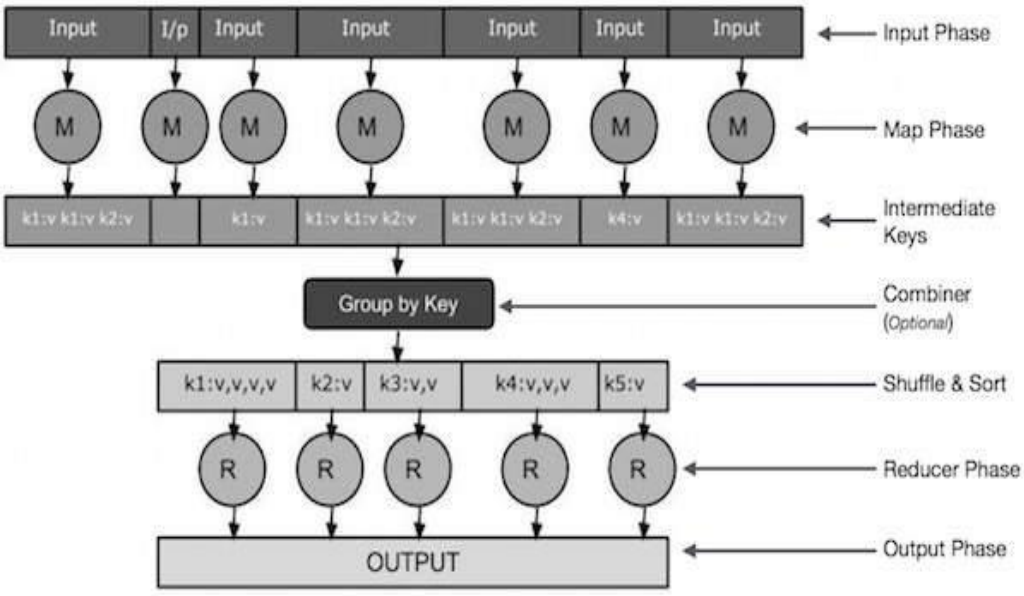

Figure 2.1: MapReduce Architecture

- **Input Phase:** Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

- **Map:** Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.

- **Intermediate Keys:** They key-value pairs generated by the mapper are known as intermediate keys.

- **Combiner:** A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.

- **Shuffle and Sort:** The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

- **Reducer:** The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

- **Output Phase:** In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

2.3 Belief Rule Base (BRB)

A Belief Rule Base (BRB) is a knowledge representation schema which allows the capturing of uncertain information. Belief rules are the key elements of a BRB (Belief Rule Base), which include belief degree. It is an extended form of traditional If-Then rules. A belief rule consists of an antecedent part and a consequent part. The antecedent attribute takes referential values, and each consequent is associated with belief degrees. The knowledge representation parameters are rule weights, antecedent attribute weights and belief degrees in consequents, which can handle uncertainty. A belief rule can be defined as follows:

$$R_k : \begin{cases} IF\ \left(P_1\ is\ A_1^k\right) \wedge \left(P_2\ is\ A_2^k\right) \wedge \dots\dots\dots \wedge \left(P_{T_k}\ is\ A_{T}{}_k^k\right) \\ THEN\ \{(C_1, \beta_{1k}), (C_2, \beta_{2k}), \dots\dots\dots, (C_N, \beta_{Nk})\} \end{cases} \qquad (1)$$

$$R_k : \left(\beta_{jk} \geq\ 0, \sum_{j=1}^{N}\ \beta_{jk} \leq 1\right)$$

with a rule weight θ_k , attribute weights δ_{k1}, δ_{k2}, δ_{k3}, … … , δ_{kT_k} where $k \in \{1,\dots\dots, L\}$

Here, P_1, P_2, P_3,……,P_{T_k} represent the antecedent attributes in the *k*-th rule. A_i^k represents one of the referential values of the *i*-th antecedent attribute P_i in the *k*-th rule. C_j is one of the consequent reference values of the belief rule. β_{jk} (= 1, … … , N, k = 1, … … , L) is the degree of belief to which the consequent reference value C_j is believed to be true. If $\sum_{j=1}^{N} \beta_{jk} = 1$ the *k*-th rule said to be complete; otherwise, it is incomplete. $\sum_{j=1}^{N} \beta_{jk} = 0$ denotes total ignorance about the output given the input in the *k*-th rule. T_k is the total number of antecedent attributes used in *k*-th rule. L is the number of all belief rules in the rule base. N is the number of all possible referential values of consequents in a rule.

Chapter 3
Architecture of Proposed System

In this chapter we will discuss about our system architecture. The details discussion of all this steps in at Chapter 4.

3.1 Overview

The steps that are involved in our system appear in Figure 3.1. The basic steps are:

i. Input data (survey data)
ii. Mapping
iii. Reducing
iv. Belief Rule

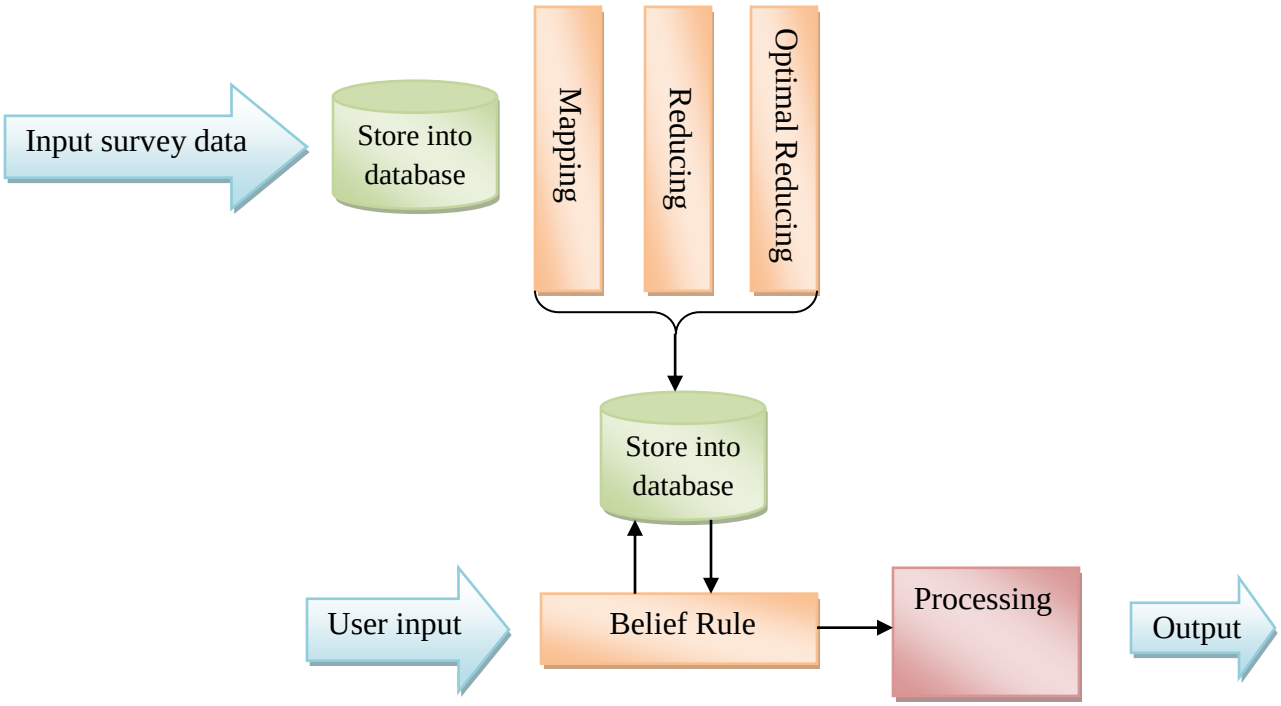

Figure 3.1: Architecture of our System

3.2 Input Data (survey data)

Input manager ask the user about 15 questions and each question contain 4 multiple choice. User must give answer for all questions. The user inputs are stored into database for further use.

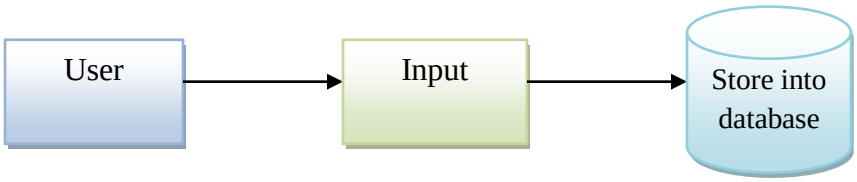

Figure 3.2: Input Architecture

3.3 Mapping

The Map task takes a set of data and converts it into another set of data. Mapper function mapping the each input by (key-value pairs) and give the output to the reducer function.

Figure 3.3: Mapping Architecture

3.4 Reducing

The Reduce task takes the output from the Map as an input and combines those data (key-value) pairs into a smaller set of data.

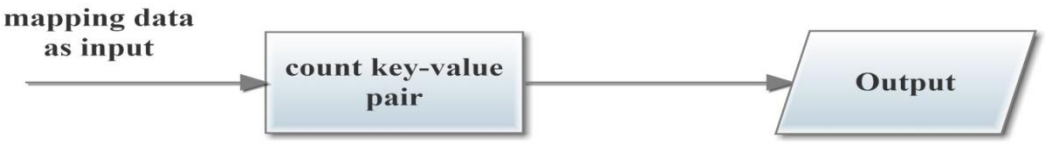

Figure 3.3: Reducing Architecture

3.5 Optimal Reducing

Optimal reducing takes reducing data as input. Then it reduces the most frequently given answer for each question and stored it into database.

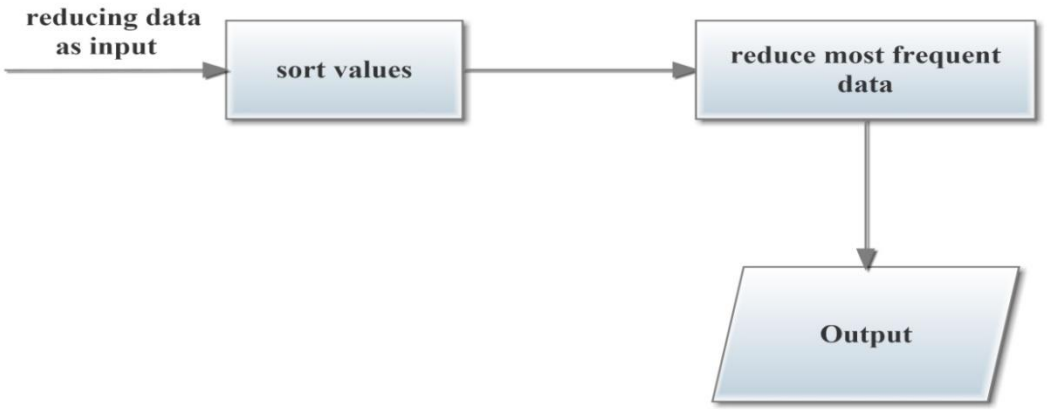

Figure 3.4: Optimal Reducing Architecture

3.5 Belief Rule Base (BRB)

Belief Rule takes input from user and access the stored data after reducing and optimal reducing is complete. Then it apply knowledge base (BRB) algorithm for processing the output.

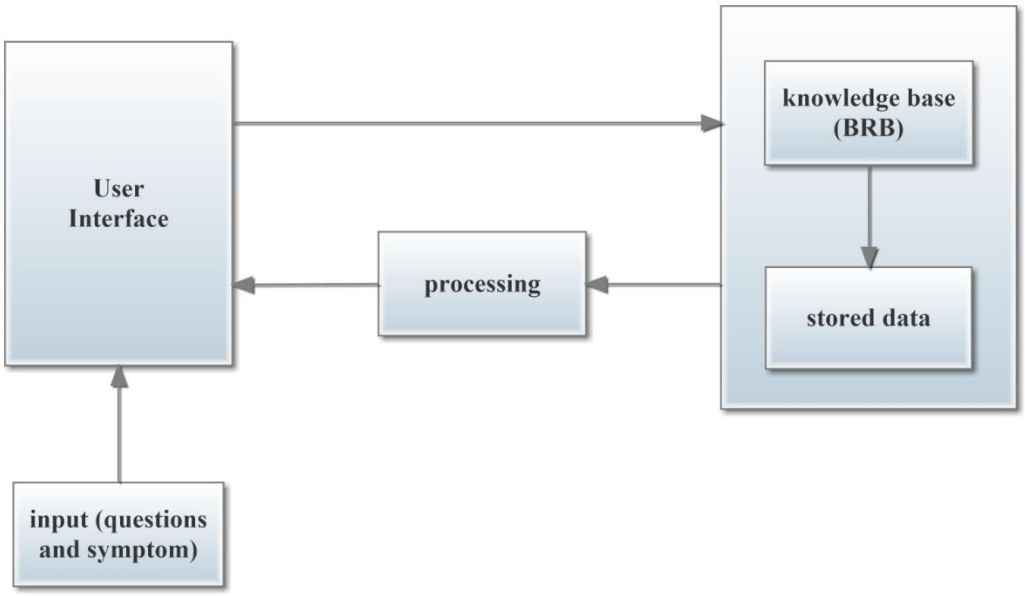

Figure 3.5: BRB Architecture

Chapter 4
Implemation & Result

4.1 Background

For developing the system we use Visual Studio 2013 and SQL Server Management Studio 2012 software. Visual Studio 2013 is used for C# coding and SQL Server Management Studio 2012 is used as database server. For designing this project HTML, CSS are used which is open source. IIS Express web server is used as web server. The short description are given below:

- **IIS Express Web Server:** IIS Express is a lightweight, self-contained version of IIS optimized for developers. IIS Express makes it easy to use the most current version of IIS to develop and test websites.

- **SQL Server Management Studio 2012:** Microsoft SQL Server Management Studio Express (SSMSE) is a free, easy-to-use graphical management tool for managing SQL Server 2012 Express Edition and SQL Server 2012 Express Edition with Advanced Services. The tool includes both script editors and graphical tools which work with objects and features of the server.

- **C# Programming Language:** C# (pronounced "C-sharp") is an object-oriented programming language from Microsoft that aims to combine the computing power of C++ with the programming ease of Visual Basic. C# is based on C++ and contains features similar to those of Java. C# is designed to work with Microsoft's .Net platform.

- **.NET Framework:** A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

  The .NET Framework contains three major parts:

  i. the Common Language Runtime
  ii. the Framework Class Library
  iii. ASP.NET

- **ASP.NET:** ASP.NET is an open source server-side Web application framework designed for Web development to produce dynamic Web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.

- **HTML:** The short form of HTML is Hyper Text Markup Language and it is a language that are used to create electronic documents, especially pages on the World Wide Web (WWW) that contain connections called hyperlinks to other pages. Every web page you see on the Internet, including this one contains HTML code that helps format and show text and images in an easy to read format. Without HTML a browser would not know how to format a page and would only display plain text with no formatting that contained no links.

- **CSS:** Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language.

- **Bootstrap:** Bootstrap is the world's most popular framework for building responsive, mobile-first sites and applications. Inside you'll find high quality HTML, CSS, and JavaScript to make starting any project easier than ever.

4.2 Input Data (survey data)

For collecting the input data we have used a user interface. In this user interface there are 15 questions. Each question has 4 parameters. These parameters indicate the consequent of the question. The user will give his/her name and id to fill up the survey questions. Each question consequent and user details will be stored into database. The database information will be stored in a structured way. We use this data set for MapReduce techniques.

4.3 Mapping

In the mapping procedure the stored data is used for mapping. The mapper function uses the question and symptom as the key and generates a unique value for each key in the mapping procedure. Then the key value pairs are stored in the database for the reducing procedure. When the same key value pairs are tried to store in the database it will not insert again rather then it will update.

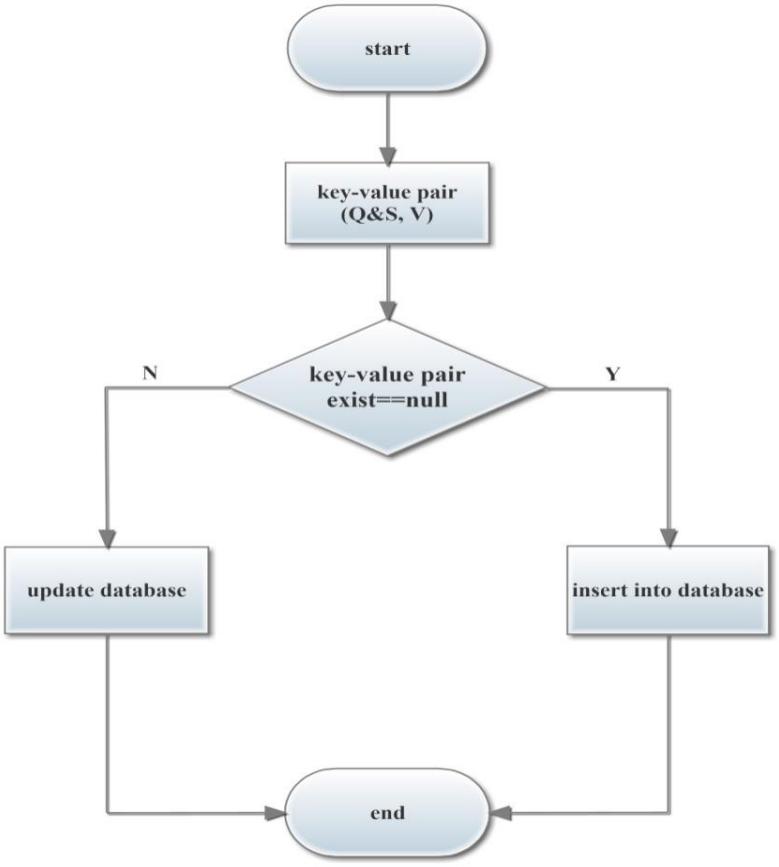

Figure 4.1: Mapping Flowchart

4.4 Reducing

In the reducing procedure the reducer function uses the mapping data. The reducer function counts the each key value pair from the mapping database. After counting the pairs the counted result is stored as (key, counted result). For each same key it does the same operation. If already existing pair is tried to insert again in the reduce database it will not insert rather than it will update.

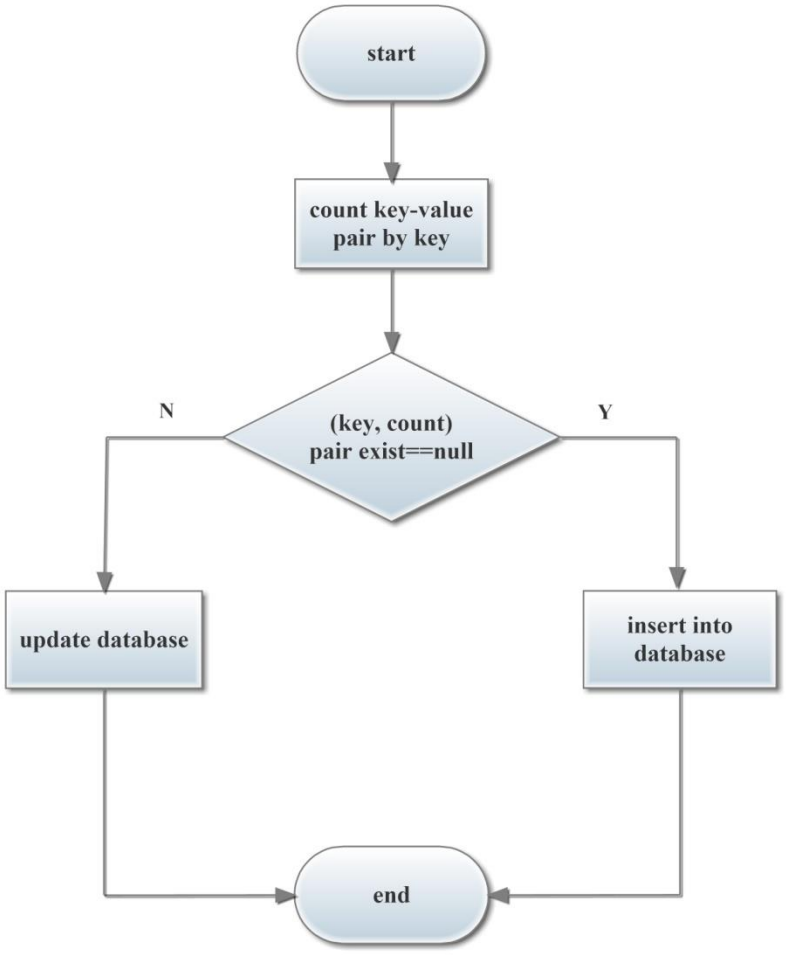

Figure 4.2: Reducing Flowchart

4.5 Optimal Reducing

Optimal reducing reduces the most frequent datasets from the previous reducing part. There are 4 types of consequent for each question. Optimal reducing reduces two frequent datasets from the reducing datasets. So that it will increase the system searching capability. BRB uses the optimal reducing database. If we keep the most frequent uses dataset in another table BRB can get the datasets easily and searching capability will increase to produce output.

4.5.1 Get Value to Reduce

Optimal reducing divided into three steps. Here first step is to getting the values to reduce. In this step at first it will get all the symptom value for each question. Then it sorts the values in ascending order and keeps them in a stack. After that it pop the first two values which are two frequent datasets.

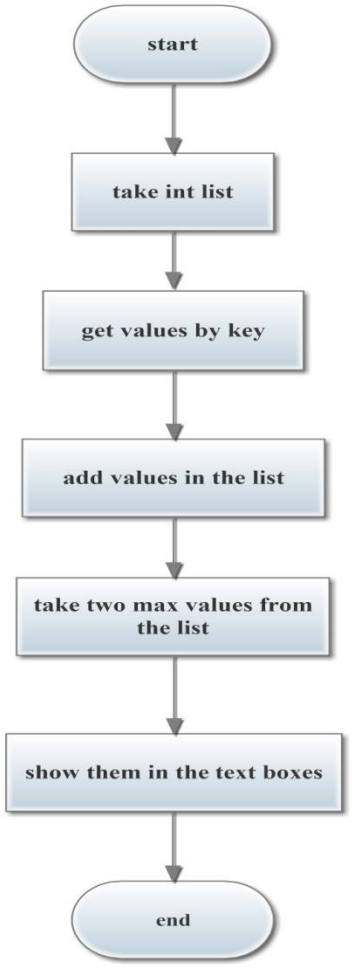

Figure 4.3: Get value to reduce flow chart

4.5.2 Reduce First Highest Value

In this step it works with the most frequent dataset. After getting the most frequent dataset value it checks is it exists in database multiple times. If the highest value exists in single time in database then it will get all the information of the highest value and store them in optimal reducing database. Otherwise it will go to next step.

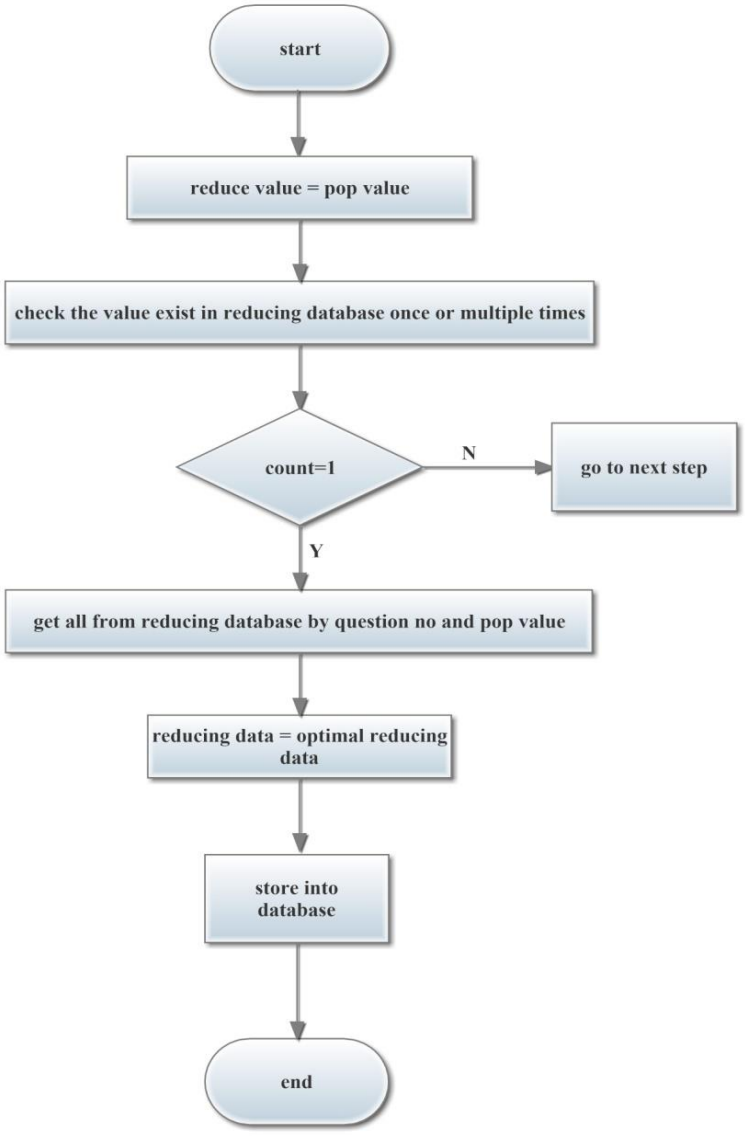

Figure 4.4: Reduce First Highest Value Flowchart

4.5.3 Reduce Second Highest Value

In this step the pop value exits in the database multiples times. If the value is first highest value then it will get all the information of max Id from database otherwise it will get the min id details from the database. After getting the data it stores them into optimal reducing table.

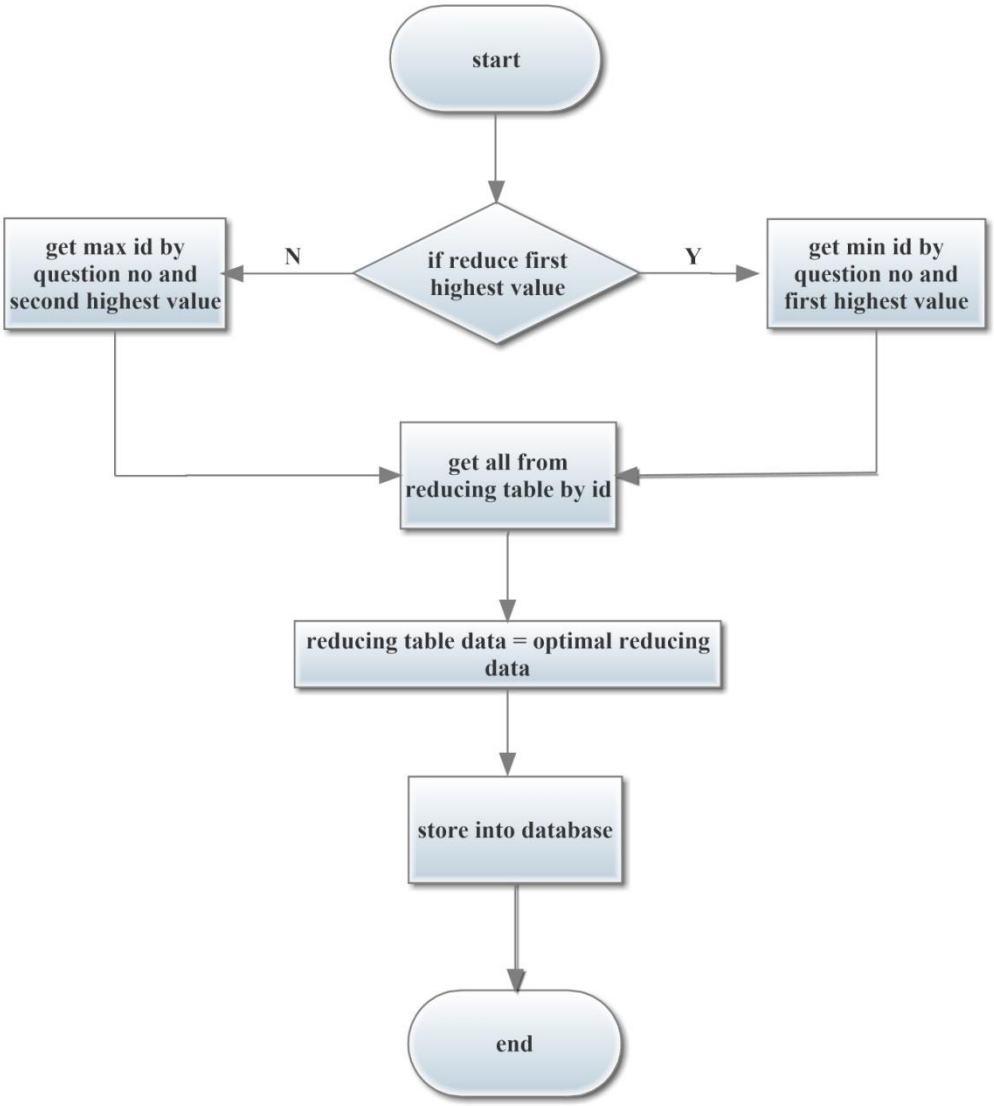

Figure 4.5: Reduce Second Highest Value Flowchart

4.6 Belief Rule Base (BRB)

BRB works in two steps. These are belief rule and processing unit. Belief rule is the key element of the BRB. Belief rule contains antecedent attributes and referential values. For each antecedent attribute and referential value it shows a consequent and a belief degree. Belief rule shows the output by consequent and belief degree. For this purpose we find out the severity level as consequent and the average probability as belief degree. To apply the belief rule the system use the map reduce data. Next step is processing unit. Processing unit uses the equation written in below:

$\boldsymbol{C_{i=1}^{n}} : \sum_{j=1}^{n} S_{ij}$ Where c_i is the count of similar referential values and S_{ij} are the referential values

$\boldsymbol{\beta_{TP_i^n}} : \sum_{j=1}^{n} \beta_{ij}$ Where β_{TP_i} is the total belief degree of similar referential values and β_{ij} is the individual belief degree of similar referential values

$\boldsymbol{Avg_i^n} : \beta_{TP_i}/C_i$ Where Avg_i is the average belief degree of similar referential values

$\boldsymbol{Max_{avg}} : Avg_i^n$ Where Max_{avg} defines the severity level.

$\boldsymbol{\gamma_f} : \frac{\sum_{k=1}^{N} \boldsymbol{\beta_{jk}}}{\boldsymbol{N}}$ Where β_{jk} is the belief degree of each antecedent attribute and N is the total number antecedent attribute. $\boldsymbol{\gamma_f}$ defines the total probability

Processing unit calculates the each symptom average probability. Then from the each average probability processing unit choose the max probability to determine the severity level of Dyslexia. After that it calculates the all symptoms average probability which shows the chance of being affected by Dyslexia.

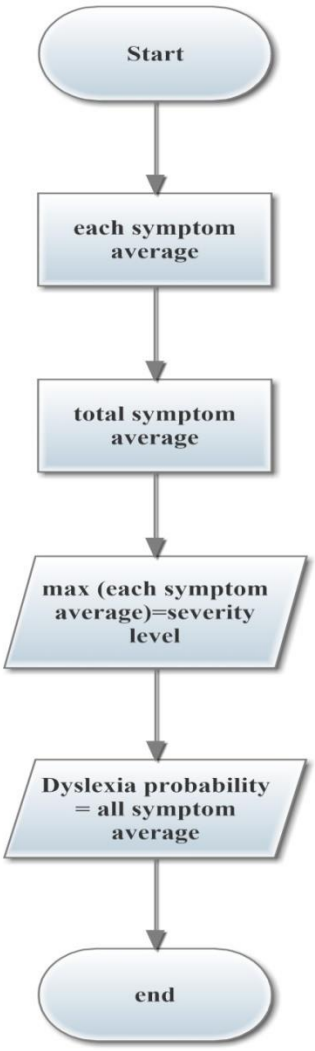

Figure 4.6: Belief Rule Base Flowchart

4.7 Working Procedure

To work with this project we have learnt about C#, ASP.NET, MapReduce, Belief Rule Base. As the domain is new, so before start working a developer should have in-depth knowledge about all these related topics.

Our main motivation was to develop a system which can predict the dyslexia probability with the severity. In SQL- Server lot of datasets are stored. Mapper and Reducer function keep the dataset in a structured way.

Now we are directly discussing the whole working procedure below:

- For our system we are using some base datasets as input. These datasets are collected from the student. We build a survey site to collect the data and also collected data manually from classroom. Then we stored the data into the database manually. There are 15 questions and each question contain 4 multiple choice. The user chooses one answer for each question.

Id	StudentId	Name	QuestionNo	Question	Symptom
1001	2011-3-60-001	Iearul Abid	1	Do You Confuse Visually Similar words such as Cat And Cot?	Rarely
1002	2011-3-60-001	Iearul Abid	2	Do You lose Your Place or miss out lines When Reading?	Occasionally
1003	2011-3-60-001	Iearul Abid	3	Do You confuse the names of objects,for example table for chair?	Rarely
1004	2011-3-60-001	Iearul Abid	4	Do You have trouble telling left from right?	Occasionally
1005	2011-3-60-001	Iearul Abid	5	Is map reading or finding your way to a strange place confuse?	Rarely
1006	2011-3-60-001	Iearul Abid	6	Do You re-read paragraphs to understand them?	Often
1007	2011-3-60-001	Iearul Abid	7	Do You Confused when given several instructions at once?	Occasionally
1008	2011-3-60-001	Iearul Abid	8	Do You make mistakes when taking down telephone messages?	Occasionally
1009	2011-3-60-001	Iearul Abid	9	Do You find it difficult to find the right word to say?	Often
1010	2011-3-60-001	Iearul Abid	10	how often Do You think of creative solutions to problem?	Occasionally
1011	2011-3-60-001	Iearul Abid	11	How easy Do You find it to sound out words such as e-le-phant?	Often
1012	2011-3-60-001	Iearul Abid	12	When Writing Do You find it difficult to organize thoughts on paper?	Occasionally
1013	2011-3-60-001	Iearul Abid	13	Do You learn your multiplication tables easily?	Rarely
1014	2011-3-60-001	Iearul Abid	14	How easy Do You find it to recite the alphabet?	Rarely
1015	2011-3-60-001	Iearul Abid	15	How hard Do You find it to read aloud?	Rarely

Figure 4.7: Base Data Input in Our System

- In this step we have used mapper function. Mapper function makes key-value pair for each input. In our system question no and the symptom is the key. So there are combinations of 60 key in the dataset. The mapper function generates unique value for each of the key.

Id	QuestionNo	Symptom	Value
459	1	Rarely	1001
460	1	Rarely	1016
461	1	Rarely	1031
462	1	Rarely	1046
463	1	Rarely	1076

558	1	Occasionally	2306
559	1	Occasionally	2336
560	1	Occasionally	2351
561	1	Occasionally	2411
562	1	Occasionally	2471

578	1	Often	1301
579	1	Often	1376
580	1	Often	1406
581	1	Often	1421
582	1	Often	1481

606	1	MostOfTheTime	1766
607	1	MostOfTheTime	2141
608	1	MostOfTheTime	3161
609	1	MostOfTheTime	3176
610	1	MostOfTheTime	3206

Figure 4.8: Mapping Output

- Reducer function uses the mapping data as input. Reducer function counts the each key-value pair and stores them into database in an organized way.

Id	QuestionNo	Symptom	Total
1011	1	Rarely	81
1012	1	Occasionally	37
1013	1	Often	30
1014	1	MostOfTheTime	5

Figure 4.9: Reducing Output

- After getting the datasets from the reducing procedure we used optimal reducer function to increase the search capability. In reducing datasets there are four combination of key. Optimal function store the most frequently used data from the reducing datasets. For doing that task we work in three steps. At first we bring two values (most frequently fill up in the survey) in the textbox for each question. Then by the textbox value we bring all the information of this value from the reducing database. Then store the information into another database.

Id	QuestionNo	Total	Symptom
34	1	81	Rarely
35	1	37	Occasionally
36	2	49	Occasionally
37	2	53	Often
38	3	93	Rarely
39	3	38	Occasionally
40	4	81	Rarely
41	4	31	Occasionally

Figure 4.10: Optimal Reducing Output

- After processing all the data then we operate BRB operation.BRB works in two steps. First step is using the belief rule and second step is processing unit. Belief rule is the key elements of the BRB, Which includes belief degree. For each of the antecedent attributes and referential values it generates a consequent and a belief degree. Belief degree is the probability of each consequent. There are four referential values for each antecedent attributes. These are rarely, occasionally, often, most of the time. Here belief rule produce 15 pair of consequent and belief degree. For each question total probability is equal 1.

- Processing unit aggregates all the consequent and belief degree. After aggregating these data processing unit do some mathematical operation to produce final output .Final output shows the severity and the total probability of dyslexia.

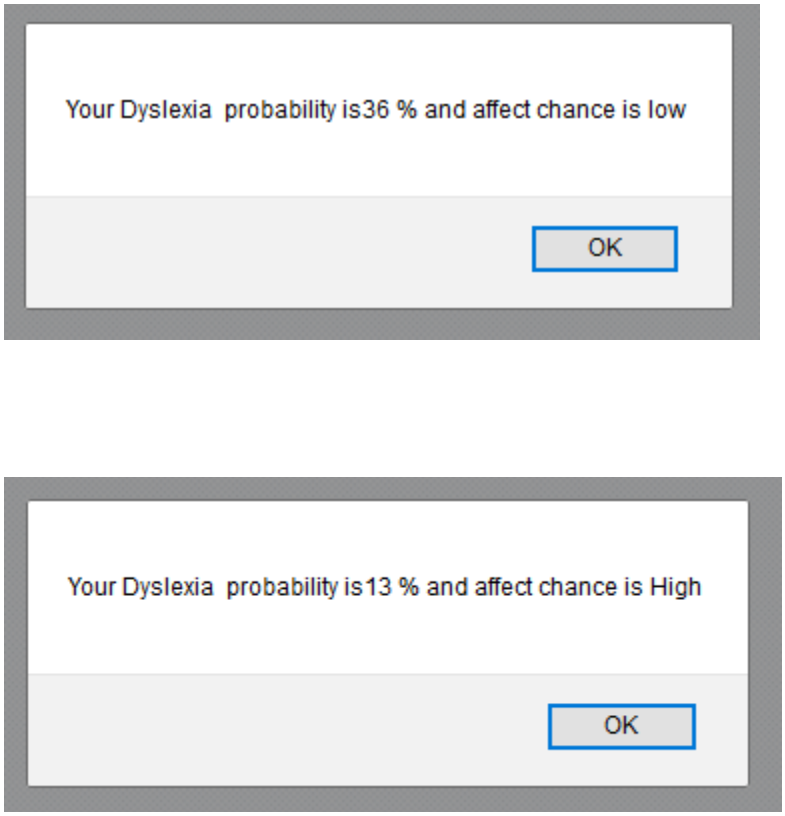

Figure 4.11: Example of Final Output

Chapter 5
Conclusion & Future Work

5.1 Conclusion

In our system we describe the development and application of BRB to predict the probability of having dyslexia based on signs or symptoms. The BRB based system is a robust system that can aid in assessing dyslexia. The system will help patient assesses improvement in dyslexia severity as well. The system provides a percentage of assessment which is more reliable and informative than any other expert's opinion without a degree of belief. The lacking of the system is that data is not processing in parallel way. We have processed the data automatically but we did not use any framework. Though we did not processed the data by a framework but our system give a suitable output which is helpful to predict the dyslexia.

5.2 Future Work

- We will add framework for parallel data processing.
- We will extend our work to assess dyslexia in some specific areas. We will try to find the percentage of dyslexic people living in a specific region.
- By collecting the data of few years we will try to make a system that will predict the upcoming year's situation of dyslexia.

References

[1] American Academy of Pediatrics, et al. (2009, reaffirmed 2014). Joint statement-Learning disabilities, dyslexia, and vision. Retrieved on May 20, 2016

[2] Shaywitz SE, et al. (2006). Dyslexia (specific reading disability). In FD Burg et al., eds., Current Pediatric Therapy, Philadelphia: Saunders Elsevier. Retrieved on May 25, 2016

[3] http://www.dyslexiasw.com/advice/all-about-dyslexia/history-of-dyslexia Retrieved on June 1, 2016

[4] https://www.safaribooksonline.com/library/view/mapreduce-design-patterns/9781449341954/ch01.html Retrieved on June 4, 2016

[5] http://www.webmd.com/children/helping-children-with-dyslexia#1 Retrieved on July 7, 2016

[5] https://www.tutorialspoint.com/map_reduce/map_reduce_introduction.htm Retrieved on July 18, 2016

[6] http://www.guru99.com/introduction-to-mapreduce.html Retrieved on July 26, 2016

[7] J.B. Yang, J. Liu, D.L. Xu, J. Wang, H.W. Wang, Optimal learning method for training belief rule based systems, IEEE Transactions on Systems, Man, and Cybernetics. Retrieved on August 6, 2016

[8] Rahaman, Saifur, MS Hossain. "A Belief Rule Based (BRB) System to Assess Asthma Suspicion". In *Computer and Information Technology (ICCIT), 2013 16th International Conference on*, IEEE, 2013. Retrieved on July 17, 2016

[9] M. S. Hossain, M. S. Khalid, S. Akter, and S. Dey, "A belief rule based expert system to diagnosed Influenza," *9th International Forum on Strategic Technology (IFOST)*, pp. 113-116, 2014. Retrieved on Septembe 3, 2016

[10] MS. Hossain, PO. Zander, S. Kamal and L.Chowdhury,"Belief Rule Based Expert Systems to Evaluate E-Government: A Case Study", *Expert Systems: The Journal of Knowledge Engineering*, Jhon Wiley & Sons Ltd.,(Early View), 2015. Retrieved on September 24, 2016. Retrieved on October 8, 2016

[11] DL. Xu, J. Liu, JB. Yang, GP. Liu, J. Wang, I. Jenkinso and J. Ren, " Inference and learning methodology of belie rule-based expert system for pipeline leak detection*", Expert Systems with Applications* 32 (2007) 103–113, 2007. Retrieved on October 24, 2016

Appendix

1. Mapping function

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using System.Web.Configuration;
using DyslexiaWebForm.Models;

namespace DyslexiaWebForm.Gateway
{
    public class MappingGateway
    {
        private SqlConnection connection = new SqlConnection();

        private string connectionString =

WebConfigurationManager.ConnectionStrings["DyslexiaConnection"].ConnectionString;

        public int MappingFunction(Mapping mapping)
        {

            int count = CountForMapping(mapping);
            Queue<int> queue = new Queue<int>();
            foreach (var i in GetId(mapping))
            {
                queue.Enqueue(i);
            }
            int rowaffected = 0;
            for (int i = 1; i <= count; i++)
            {

                int value = queue.Dequeue();
                if (!IsIdExist(value))
                {
                    connection.ConnectionString = connectionString;
                    string query = "insert into t_mapping
values('"+mapping.QuestionNo+"','"+mapping.Symptom+"','"+value+"') ";
                    SqlCommand command=new SqlCommand(query,connection);
                    connection.Open();
                    rowaffected = command.ExecuteNonQuery();
                    connection.Close();
                }
                else
                {
                    connection.ConnectionString = connectionString;
                    string query = "update t_mapping set QuestionNo='"+
mapping.QuestionNo + "', Symptom='" + mapping.Symptom + "' where Value='" + value + "' ";
                    SqlCommand command = new SqlCommand(query, connection);
                    connection.Open();
                     rowaffected = command.ExecuteNonQuery();
```

```
                connection.Close();
            }

        }

        return rowaffected;
    }

    public int CountForMapping(Mapping mapp)
    {
        int count = 0;
        string query = "SELECT COUNT(Id) FROM t_dyslexiainput where QuestionNO='" +
mapp.QuestionNo + "' And Answer='" + mapp.Symptom + "'";

        connection.ConnectionString = connectionString;

        SqlCommand command = new SqlCommand(query, connection);

        connection.Open();
        count = Convert.ToInt32(command.ExecuteScalar().ToString());
        connection.Close();
        return count;
    }

    public List<int> GetId(Mapping mapp)
    {
        connection.ConnectionString = connectionString;
        string query = "SELECT Id FROM t_dyslexiainput where QuestionNO='" +
mapp.QuestionNo + "' And Answer='" + mapp.Symptom + "'";
        SqlCommand command=new SqlCommand(query,connection);
        connection.Open();

        SqlDataReader reader = command.ExecuteReader();
        List<int> intList = new List<int>();
        while (reader.Read())
        {

            intList.Add((int)reader["Id"]);

        }
        connection.Close();
        return intList;

    }
    public bool IsIdExist(int id )
    {
        connection.ConnectionString = connectionString;
        string query = "select * from t_mapping where Value='" + id + "'";

        SqlCommand command = new SqlCommand(query, connection);
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        bool IsRowExists = false;
        if (reader.HasRows)
        {
            IsRowExists = true;
        }
        connection.Close();
```

```
            return IsRowExists;
        }
        public List<Mapping> MappinglList(Mapping mapp)
        {
            connection.ConnectionString = connectionString;
            string query = "SELECT * FROM t_mapping";
            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();

            SqlDataReader reader = command.ExecuteReader();
            List<Mapping> mappingsList = new List<Mapping>();
            while (reader.Read())
            {
                Mapping mapping=new Mapping();
                mapping.Id = (int) reader["Id"];
                mapping.QuestionNo = (int) reader["QuestionNo"];
                mapping.Symptom = reader["Symptom"].ToString();
                mapping.Value = (int) reader["Value"];
                mappingsList.Add(mapping);
            }
            connection.Close();
            return mappingsList;

        }
    }
}
```

2. Reducing

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Web;
using System.Web.Configuration;
using DyslexiaWebForm.Manager;
using DyslexiaWebForm.Models;

namespace DyslexiaWebForm.Gateway
{
    public class ReducingGateway
    {
        ReducingManager reducingManager=new ReducingManager();
        private SqlConnection connection = new SqlConnection();

        private string connectionString =
```

```
WebConfigurationManager.ConnectionStrings["DyslexiaConnection"].ConnectionString;

        public int Reducing(Reducing reducing)
        {
            string query = "SELECT COUNT(Id) FROM t_mapping where QuestionNO='" +
reducing.QuestionNo + "' And Symptom='" + reducing.Symptom + "'";

            connection.ConnectionString = connectionString;

            SqlCommand command = new SqlCommand(query, connection);

            connection.Open();


            reducing.RarelyTotal = Convert.ToInt32(command.ExecuteScalar().ToString());


            if (!reducingManager.IsDataExists(reducing))
            {

                string queryresult = "insert into t_reducing values('" +
reducing.QuestionNo + "','" +
                                     reducing.Symptom + "','" + reducing.RarelyTotal +
"')";
                SqlCommand commandNext = new SqlCommand(queryresult, connection);
                int rowAffected = commandNext.ExecuteNonQuery();
                connection.Close();
                return rowAffected;
            }
            else
            {
                string queryresult = "update t_reducing set
Total='"+reducing.RarelyTotal+"' where QuestionNO='"+reducing.QuestionNo+"' and
Symptom='"+reducing.Symptom+"' ";
                SqlCommand commandNext = new SqlCommand(queryresult, connection);
                int rowAffected = commandNext.ExecuteNonQuery();
                connection.Close();
                return rowAffected;
            }

        }



        public List<ReducingSummary> GetReducingSummaries()
        {
            string listquery = "select * from t_reducing";
            connection.ConnectionString = connectionString;
            SqlCommand command = new SqlCommand(listquery, connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            List<ReducingSummary> questionSummaries = new List<ReducingSummary>();
            while (reader.Read())
            {
                ReducingSummary question = new ReducingSummary();
                question.Id = (int)reader["Id"];
```

```
                question.QuestionNo = (int)reader["QuestionNo"];
                question.Symptom = reader["Symptom"].ToString();
                question.Total = (int)reader["Total"];
                questionSummaries.Add(question);
            }
            reader.Close();
            connection.Close();
            return questionSummaries;
        }

    }
}
```

3. Optimal Reducing

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using System.Web.Configuration;
using DyslexiaWebForm.Models;
using Microsoft.Ajax.Utilities;

namespace DyslexiaWebForm.Gateway
{
    public class OptimalReduceGateway
    {
        private SqlConnection connection = new SqlConnection();

        private string connectionString =

WebConfigurationManager.ConnectionStrings["DyslexiaConnection"].ConnectionString;
        ReducingLogicgateway reducingLogicgateway=new ReducingLogicgateway();
        public List<int> Reducing(OptimalReduce optimalReduce)
        {
            connection.ConnectionString = connectionString;
            List<int> getNumList = new List<int>();
            for (int i = 1; i <=4; i++)
            {

                if (i==1)
                {
                    string query1 = "select Total from t_reducing where QuestionNo='" +
optimalReduce.QuestionNo + "'  and Symptom='Rarely'";
                    SqlCommand command1 = new SqlCommand(query1, connection);
                    connection.Open();

                    SqlDataReader reader1 = command1.ExecuteReader();
                    if (reader1.Read())
                    {
                        int getNewTotal1 = 0;
                        getNewTotal1 = (int)reader1["Total"];
```

```
                    getNumList.Add(getNewTotal1);
                }
                connection.Close();

            }
            if (i==2)
            {
                string query2 = "select Total from t_reducing where QuestionNo='" +
optimalReduce.QuestionNo + "'  and Symptom='Occasionally'";
                SqlCommand command2 = new SqlCommand(query2, connection);
                connection.Open();

                SqlDataReader reader2 = command2.ExecuteReader();
                if (reader2.Read())
                {
                    int getNewTotal2 = 0;
                    getNewTotal2 = (int)reader2["Total"];
                    getNumList.Add(getNewTotal2);
                }
                connection.Close();

            }
            if (i==3)
            {
                string query3 = "select Total from t_reducing where QuestionNo='" +
optimalReduce.QuestionNo + "'  and Symptom='Often'";
                SqlCommand command3 = new SqlCommand(query3, connection);
                connection.Open();

                SqlDataReader reader3 = command3.ExecuteReader();
                if (reader3.Read())
                {
                    int getNewTotal3 = 0;
                    getNewTotal3 = (int)reader3["Total"];
                    getNumList.Add(getNewTotal3);
                }
                connection.Close();

            }
            if (i==4)
            {
                string query4 = "select Total from t_reducing where QuestionNo='" +
optimalReduce.QuestionNo + "'  and Symptom='MostOfTheTime'";
                SqlCommand command4 = new SqlCommand(query4, connection);
                connection.Open();

                SqlDataReader reader4 = command4.ExecuteReader();
                if (reader4.Read())
                {
                    int getNewTotal4 = 0;
                    getNewTotal4 = (int)reader4["Total"];
                    getNumList.Add(getNewTotal4);
                }
                connection.Close();

            }

        }
```

```
            return getNumList;
        }

        public List<OptimalReduceSummary> GetAllReducingData()
        {
            string listquery = "select * from t_optimalreducing";
            connection.ConnectionString = connectionString;
            SqlCommand command = new SqlCommand(listquery, connection);
            connection.Open();

            SqlDataReader reader = command.ExecuteReader();
            List<OptimalReduceSummary> reduces = new List<OptimalReduceSummary>();

            while (reader.Read())
            {
                OptimalReduceSummary reducing=new OptimalReduceSummary();
                reducing.Id = (int)reader["Id"];
                reducing.QuestionNo = (int)reader["QuestionNo"];
                reducing.Symptom = reader["Symptom"].ToString();
                reducing.Total = (int)reader["Total"];
                reduces.Add(reducing);
            }
            reader.Close();
            connection.Close();
            return reduces;
        }

        public int Save2(OptimalReduce optimalReduce)
        {
            int rowaffection = 0;
            if (reducingLogicgateway.CheckCount2(optimalReduce)==1)
            {
                string querys = "SELECT * FROM t_reducing where QuestionNO='" +
optimalReduce.QuestionNo + "' And Total='" + optimalReduce.SecondHighestValue + "'";

                connection.ConnectionString = connectionString;

                SqlCommand commands = new SqlCommand(querys, connection);

                connection.Open();

                SqlDataReader readers = commands.ExecuteReader();

                while (readers.Read())
                {

                    optimalReduce.Answer = readers["Symptom"].ToString();


                }
                readers.Close();
                if (reducingLogicgateway.IsQuestionExist(optimalReduce.QuestionNo)==true)
                {
                    if (!reducingLogicgateway.IsDataExists(optimalReduce.QuestionNo,
optimalReduce.SecondHighestValue))
                    {
```

```
                        var lowestvalue =
reducingLogicgateway.Checklowestvalue(optimalReduce.QuestionNo);

                        if (lowestvalue <= optimalReduce.SecondHighestValue)
                        {
                            int id =
reducingLogicgateway.IsDataExistsById(optimalReduce.QuestionNo, lowestvalue);
                            string queryrsult = "update t_optimalreducing set
questionNo='" + optimalReduce.QuestionNo + "',Symptom='" + optimalReduce.Answer +
"',Total='" + optimalReduce.SecondHighestValue + "' where Id='" + id + "' ";
                            SqlCommand commandNext = new SqlCommand(queryrsult,
connection);
                            rowaffection = commandNext.ExecuteNonQuery();
                            connection.Close();

                        }
                        else
                        {
                            string queryresult = "Insert into t_optimalreducing
values('" + optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.SecondHighestValue + "')";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowaffection = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }

                    }
                    else
                    {
                        var lowestvalue =
reducingLogicgateway.Checklowestvalue(optimalReduce.QuestionNo);
                        int id =
reducingLogicgateway.IsDataExistsById(optimalReduce.QuestionNo, lowestvalue);

                        string queryresult = "update t_optimalreducing set Total='" +
optimalReduce.SecondHighestValue + "', QuestionNO='" + optimalReduce.QuestionNo +
"',Symptom='" + optimalReduce.Answer + "' where Id='"+id+"' ";
                        SqlCommand commandNext = new SqlCommand(queryresult, connection);
                        rowaffection = commandNext.ExecuteNonQuery();
                        connection.Close();

                    }
                }
                else
                {
                    string queryresult = "Insert into t_optimalreducing  values('" +
optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.SecondHighestValue + "')";
                    SqlCommand commandNext = new SqlCommand(queryresult, connection);
                    rowaffection = commandNext.ExecuteNonQuery();
                    connection.Close();
                }


            }
            if (reducingLogicgateway.CheckCount2(optimalReduce)>1)
```

```
            {
                int getId = reducingLogicgateway.MaxCount(optimalReduce);
                connection.ConnectionString = connectionString;
                string getquery = "select * from t_reducing where id='" + getId + "'";
                SqlCommand getCommand = new SqlCommand(getquery, connection);
                connection.Open();
                SqlDataReader getReader = getCommand.ExecuteReader();

                while (getReader.Read())
                {
                    optimalReduce.QuestionNo = (int)getReader["QuestionNo"];
                    optimalReduce.Answer = getReader["Symptom"].ToString();
                    optimalReduce.Total = (int)getReader["Total"];
                }
                getReader.Close();
                if (reducingLogicgateway.IsQuestionExist(optimalReduce.QuestionNo)==true)
                {
                    if (!reducingLogicgateway.IsDataExists(optimalReduce.QuestionNo,
optimalReduce.SecondHighestValue))
                    {


                        var lowestvalue =
reducingLogicgateway.Checklowestvalue(optimalReduce.QuestionNo);

                        if (lowestvalue <= optimalReduce.SecondHighestValue)
                        {
                            int id =
reducingLogicgateway.IsDataExistsById(optimalReduce.QuestionNo, lowestvalue);
                            string queryrsult = "update t_optimalreducing set
QuestionNo='" + optimalReduce.QuestionNo + "',Symptom='" + optimalReduce.Answer +
"',Total='" + optimalReduce.SecondHighestValue + "' where Id='" + id + "' ";
                            SqlCommand commandNext = new SqlCommand(queryrsult,
connection);
                            rowaffection = commandNext.ExecuteNonQuery();
                            connection.Close();

                        }
                        else
                        {
                            string queryresult = "insert into t_optimalreducing values('"
+ optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.SecondHighestValue + "')";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowaffection = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }

                    }
                    else
                    {
                        if
(!reducingLogicgateway.IsDataExistsBYSymptom(optimalReduce.QuestionNo,optimalReduce.Answe
r))
                        {
```

```
                            string queryresult = "insert into t_optimalreducing values('"
+ optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.SecondHighestValue + "')";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowaffection = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }
                        else
                        {
                            string queryresult = "update t_optimalreducing set Total='" +
optimalReduce.SecondHighestValue + "' where QuestionNO='" + optimalReduce.QuestionNo + "'
and Symptom='" + optimalReduce.Answer + "' ";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowaffection = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }



                    }
                }
                else
                {
                    string queryresult = "insert into t_optimalreducing values('" +
optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.SecondHighestValue + "')";
                    SqlCommand commandNext = new SqlCommand(queryresult, connection);
                    rowaffection = commandNext.ExecuteNonQuery();
                    connection.Close();
                }

            }

            return rowaffection;

        }
        public int Save1(OptimalReduce optimalReduce)
        {
            int rowAffectes=0;

            if (reducingLogicgateway.CheckCount1(optimalReduce)==1)
            {
            string queryf = "SELECT * FROM t_reducing where QuestionNO='" +
optimalReduce.QuestionNo + "' And Total='" + optimalReduce.FirstHighestValue + "'";

            connection.ConnectionString = connectionString;

            SqlCommand commandf = new SqlCommand(queryf, connection);

            connection.Open();

            SqlDataReader readerf = commandf.ExecuteReader();

            while (readerf.Read())
            {
```

```
                optimalReduce.Answer = readerf["Symptom"].ToString();


            }
            readerf.Close();
                if (reducingLogicgateway.IsQuestionExist(optimalReduce.QuestionNo)==true)
                {
                    if (reducingLogicgateway.IsDataExists(optimalReduce.QuestionNo,
optimalReduce.FirstHighestValue)==true)
                    {

                        var highestvalue =
reducingLogicgateway.Checkhighestvalue(optimalReduce.QuestionNo);

                        if (highestvalue <= optimalReduce.FirstHighestValue)
                        {
                            int id =
reducingLogicgateway.IsDataExistsById(optimalReduce.QuestionNo, highestvalue);
                            string queryrsult = "update t_optimalreducing set
QuestionNo='" + optimalReduce.QuestionNo + "',Symptom='" + optimalReduce.Answer +
"',Total='" + optimalReduce.FirstHighestValue + "' where Id='" + id + "' ";
                            SqlCommand commandNext = new SqlCommand(queryrsult,
connection);
                            rowAffectes = commandNext.ExecuteNonQuery();
                            connection.Close();

                        }
                        else
                        {
                            string queryresult = "insert into t_optimalreducing values('"
+ optimalReduce.QuestionNo + "','" +
                                                    optimalReduce.Answer + "','" +
optimalReduce.FirstHighestValue + "')";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowAffectes = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }



                    }
                    else
                    {
                        string queryresult = "update t_optimalreducing set Total='" +
optimalReduce.FirstHighestValue + "' where QuestionNO='" + optimalReduce.QuestionNo + "'
and Symptom='" + optimalReduce.Answer + "' ";
                        SqlCommand commandNext = new SqlCommand(queryresult, connection);
                        rowAffectes = commandNext.ExecuteNonQuery();
                        connection.Close();


                    }
                }
                else
                {
                    string queryresult = "insert into t_optimalreducing values('" +
optimalReduce.QuestionNo + "','" +
```

```
                                            optimalReduce.Answer + "','" +
optimalReduce.FirstHighestValue + "')";
                    SqlCommand commandNext = new SqlCommand(queryresult, connection);
                    rowAffectes = commandNext.ExecuteNonQuery();
                    connection.Close();
                }


            }
            if (reducingLogicgateway.CheckCount1(optimalReduce)>1)
            {
                int getId = reducingLogicgateway.MinCount(optimalReduce);
                connection.ConnectionString = connectionString;
                string getquery = "select * from t_reducing where id='" + getId + "'";
                SqlCommand getCommand=new SqlCommand(getquery,connection);
                connection.Open();
                SqlDataReader getReader = getCommand.ExecuteReader();

                while (getReader.Read())
                {
                    optimalReduce.QuestionNo = (int) getReader["QuestionNo"];
                    optimalReduce.Answer = getReader["Symptom"].ToString();
                    optimalReduce.Total = (int) getReader["Total"];
                }
                getReader.Close();
                if (reducingLogicgateway.IsQuestionExist(optimalReduce.QuestionNo)==true)
                {
                    if (!reducingLogicgateway.IsDataExists(optimalReduce.QuestionNo,
optimalReduce.FirstHighestValue))
                    {
                        var highestvalue =
reducingLogicgateway.Checkhighestvalue(optimalReduce.QuestionNo);

                        if (highestvalue <= optimalReduce.FirstHighestValue)
                        {
                            int id =
reducingLogicgateway.IsDataExistsById(optimalReduce.QuestionNo, highestvalue);
                            string queryrsult = "update t_optimalreducing set
QuestionNo='" + optimalReduce.QuestionNo + "',Symptom='" + optimalReduce.Answer +
"',Total='" + optimalReduce.FirstHighestValue + "' where Id='" + id + "' ";
                            SqlCommand commandNext = new SqlCommand(queryrsult,
connection);
                            rowAffectes = commandNext.ExecuteNonQuery();
                            connection.Close();

                        }
                        else
                        {
                            string queryresult = "insert into t_optimalreducing values('"
+ optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.FirstHighestValue + "')";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowAffectes = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }
```

```
                    }
                    else
                    {
                        if
(!reducingLogicgateway.IsDataExistsBYSymptom(optimalReduce.QuestionNo,
optimalReduce.Answer))
                        {
                            string queryresult = "insert into t_optimalreducing values('"
+ optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.FirstHighestValue + "')";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowAffectes = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }
                        else
                        {
                            string queryresult = "update t_optimalreducing set Total='" +
optimalReduce.FirstHighestValue + "' where QuestionNO='" + optimalReduce.QuestionNo + "'
and Symptom='" + optimalReduce.Answer + "' ";
                            SqlCommand commandNext = new SqlCommand(queryresult,
connection);
                            rowAffectes = commandNext.ExecuteNonQuery();
                            connection.Close();
                        }


                    }
                }
                else
                {
                    string queryresult = "insert into t_optimalreducing values('" +
optimalReduce.QuestionNo + "','" + optimalReduce.Answer + "','" +
optimalReduce.FirstHighestValue + "')";
                    SqlCommand commandNext = new SqlCommand(queryresult, connection);
                    rowAffectes = commandNext.ExecuteNonQuery();
                    connection.Close();
                }


            }

            return rowAffectes;

        }
    }
}

public bool IsDataExists( int questionNo,int total)
        {
            connection.ConnectionString = connectionString;
            string query = "select * from t_optimalreducing where
QuestionNo='"+questionNo+"' and Total='"+total+"'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            bool IsRowExists = false;
```

```
            if (reader.HasRows)
            {
                IsRowExists = true;
            }
            connection.Close();

            return IsRowExists;
        }

        public bool IsQuestionExist(int questionNO)
        {
            connection.ConnectionString = connectionString;
            string query = "select * from t_optimalreducing where
QuestionNo='"+questionNO+"'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            bool IsRowExists = false;
            if (reader.HasRows)
            {
                IsRowExists = true;
            }
            connection.Close();

            return IsRowExists;
        }



        public int CheckCount1(OptimalReduce optimalReduce)
        {
            connection.ConnectionString = connectionString;
            string query = "SELECT COUNT(Id) FROM t_reducing where QuestionNO='" +
optimalReduce.QuestionNo + "' And Total='" + optimalReduce.FirstHighestValue + "'";

             SqlCommand command=new SqlCommand(query,connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;
        }
        public int CheckCount2(OptimalReduce optimalReduce)
        {
            connection.ConnectionString = connectionString;
            string query = "SELECT COUNT(Id) FROM t_reducing where QuestionNO='" +
optimalReduce.QuestionNo + "' And Total='" + optimalReduce.SecondHighestValue + "'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;
        }

        public int MinCount(OptimalReduce optimalReduce)
        {
            connection.ConnectionString = connectionString;
```

```
            string query = "SELECT Min(Id) FROM t_reducing where QuestionNO='" +
optimalReduce.QuestionNo + "' And Total='" + optimalReduce.FirstHighestValue + "'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;
        }

        public int MaxCount(OptimalReduce optimalReduce)
        {
            connection.ConnectionString = connectionString;
            string query = "SELECT Max(Id) FROM t_reducing where QuestionNO='" +
optimalReduce.QuestionNo + "' And Total='" + optimalReduce.SecondHighestValue + "'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;
        }
        public int IsDataExistsById(int QuestionNO,int Value)
        {
            connection.ConnectionString = connectionString;
            string query = " select Id from t_optimalreducing where
QuestionNo='"+QuestionNO+"' and Total='"+Value+"'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            int rowaffected = 0;
            if (reader.Read())
            {
                 rowaffected = (int)reader["Id"];
            }

            reader.Close();

            connection.Close();

            return rowaffected;
        }

        public int Checkhighestvalue(int questionNo)
        {
            connection.ConnectionString = connectionString;
            string query = "SELECT Max(Total) FROM t_optimalreducing where QuestionNO='"
+questionNo + "'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;
        }

        public int Checklowestvalue(int questionNo)
```

```
        {
            connection.ConnectionString = connectionString;
            string query = "SELECT Min(Total) FROM t_optimalreducing where QuestionNO='"
+ questionNo + "'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;
        }

        public bool IsDataExistsBYSymptom(int questionNo, string answer)
        {
            connection.ConnectionString = connectionString;
            string query = "select * from t_optimalreducing where QuestionNo='" +
questionNo + "' and Symptom='" + answer + "'";

            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            bool IsRowExists = false;
            if (reader.HasRows)
            {
                IsRowExists = true;
            }
            connection.Close();

            return IsRowExists;
        }
```

4. Belief Rule Base (BRB)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using DyslexiaWebForm.Manager;
using DyslexiaWebForm.Models;

namespace DyslexiaWebForm.UI
{
    public partial class PredictDyslexia : System.Web.UI.Page
    {
        PredictManager predictManager=new PredictManager();
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void predictOneButton_Click(object sender, EventArgs e)
```

```
        {
            decimal totalPercentage = 0;
            decimal percentage = 0;
            Prediction prediction = new Prediction();
            List<decimal> list=new List<decimal>();
            decimal rarelyTotal = 0;
            int rarelyCount = 0;
            decimal occasionallyTotal = 0;
            int occasionallyCount = 0;
            decimal oftenTotal  = 0;
            int oftenCount = 0;
            decimal mostOfTheTimeTotal = 0;
            int mostOfTheTimeCount = 0;
            for (int i = 1; i < 16; i++)
            {

                if (i == 1)

                {

                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);
                    if (Rarely1RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely1RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal+rarelypercentage;
                       percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;
                    }
                    else if (Occasionally1RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally1RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often1RadioButton.Checked)
                    {
                        prediction.Symptom = Often1RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                         percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;

                    }
                    else if (MostOfTheTime1RadioButton.Checked)
```

```
                    {
                        prediction.Symptom = MostOfTheTime1RadioButton.Text;

                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;
                    }


                    list.Add(percentage);
                }
                if (i == 2)
                {

                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely2RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely2RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;


                    }
                    else if (Occasionally2RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally2RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often2RadioButton.Checked)
                    {
                        prediction.Symptom = Often2RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOfTheTime2RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime2RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }

                    list.Add(percentage);


                }
                if (i == 3)
                {

                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely3RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely3RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally3RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally3RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often3RadioButton.Checked)
                    {
                        prediction.Symptom = Often3RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
```

```
                    oftenCount = oftenCount + 1;

                }
                else if (MostOfTheTime3RadioButton.Checked)
                {
                  prediction.Symptom = MostOfTheTime3RadioButton.Text;
                  int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                  decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                  mostOfTheTimeTotal = mostOfTheTimeTotal + mostOfTheTimepercentage;
                  percentage = mostOfTheTimepercentage;
                  mostOfTheTimeCount = mostOfTheTimeCount + 1;

                }

                list.Add(percentage);

            }
            if (i == 4)
            {
                prediction.QuestionNo = i;

                prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                if (Rarely4RadioButton.Checked)
                {
                    prediction.Symptom = Rarely4RadioButton.Text;
                    int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                    decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                    rarelyTotal = rarelyTotal + rarelypercentage;
                    percentage = rarelypercentage;
                    rarelyCount = rarelyCount + 1;

                }
                else if (Occasionally4RadioButton.Checked)
                {
                    prediction.Symptom = Occasionally4RadioButton.Text;
                    int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                    decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                    occasionallyTotal = occasionallyTotal + occasionallypercentage;
                    percentage = occasionallypercentage;
                    occasionallyCount = occasionallyCount + 1;

                }
                else if (Often4RadioButton.Checked)
                {
                    prediction.Symptom = Often4RadioButton.Text;
                    int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                    decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                    oftenTotal = oftenTotal + oftenpercentage;
                    percentage = oftenpercentage;
```

```
                        oftenCount = oftenCount + 1;

                    }
                    else if (MostOfTheTime4RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime4RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;
                    }

                    list.Add(percentage);
                }
                if (i == 5)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely5RadioButton.Checked)
                    {

                        prediction.Symptom = Rarely5RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally5RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally5RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often5RadioButton.Checked)
                    {
                        prediction.Symptom = Often5RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOfTheTime5RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime5RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }


                    list.Add(percentage);
                }
                if (i == 6)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely6RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely6RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally6RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally6RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often6RadioButton.Checked)
                    {
                        prediction.Symptom = Often6RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOfTheTime6RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime6RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal + mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }


                    list.Add(percentage);

                }
                if (i == 7)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely7RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely7RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally7RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally7RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often7RadioButton.Checked)
                    {
                        prediction.Symptom = Often7RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOfTheTime7RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime7RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;
                    }


                    list.Add(percentage);
                }
               if (i == 8)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely8RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely8RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally8RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally8RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often8RadioButton.Checked)
                    {
                        prediction.Symptom = Often1RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOfTheTime8RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime8RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }


                    list.Add(percentage);
                }
                if (i == 9)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely9RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely9RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally9RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally9RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often9RadioButton.Checked)
                    {
                        prediction.Symptom = Often9RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOfTheTime9RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime9RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal + mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }


                    list.Add(percentage);
                }
                if (i == 10)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely10RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely10RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally10RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally10RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;


                    }
                    else if (Often10RadioButton.Checked)
                    {
                        prediction.Symptom = Often10RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;
```

```
                    }
                    else if (MostOftheTime10RadioButton.Checked)
                    {
                        prediction.Symptom = MostOftheTime10RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }

                    list.Add(percentage);

                }
                if (i == 11)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely11RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely11RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;
                    }
                    else if (Occasionally11RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally11RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often11RadioButton.Checked)
                    {
                        prediction.Symptom = Often11RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;

                    }
```

```
                    else if (MostOfTheTime11RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime11RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal + mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }

                    list.Add(percentage);
                }
                if (i == 12)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely12RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely12RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally12RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally12RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often12RadioButton.Checked)
                    {
                        prediction.Symptom = Often12RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;

                    }
                    else if (MostOfTheTime12RadioButton.Checked)
                    {
```

```
                        prediction.Symptom = MostOfTheTime12RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }


                    list.Add(percentage);
                }
                if (i == 13)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely13RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely13RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally13RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally13RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often13RadioButton.Checked)
                    {
                        prediction.Symptom = Often13RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;

                    }
                    else if (MostOfTheTime13RadioButton.Checked)
                    {
```

```
                        prediction.Symptom = MostOfTheTime13RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }

                    list.Add(percentage);
                }
                if (i == 14)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely14RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely14RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally14RadioButton.Checked)
                    {
                       prediction.Symptom = Occasionally14RadioButton.Text;
                       int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                       decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                       occasionallyTotal = occasionallyTotal + occasionallypercentage;
                       percentage = occasionallypercentage;
                       occasionallyCount = occasionallyCount + 1;


                    }
                    else if (Often14RadioButton.Checked)
                    {
                        prediction.Symptom = Often14RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;

                    }
                    else if (MostOfTheTime14RadioButton.Checked)
                    {
```

```
                        prediction.Symptom = MostOfTheTime14RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }

                    list.Add(percentage);
                }
                if (i == 15)
                {
                    prediction.QuestionNo = i;
                    prediction.Sum = predictManager.GetSum(prediction.QuestionNo);

                    if (Rarely15RadioButton.Checked)
                    {
                        prediction.Symptom = Rarely15RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal rarelypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        rarelyTotal = rarelyTotal + rarelypercentage;
                        percentage = rarelypercentage;
                        rarelyCount = rarelyCount + 1;

                    }
                    else if (Occasionally15RadioButton.Checked)
                    {
                        prediction.Symptom = Occasionally15RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal occasionallypercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        occasionallyTotal = occasionallyTotal + occasionallypercentage;
                        percentage = occasionallypercentage;
                        occasionallyCount = occasionallyCount + 1;

                    }
                    else if (Often15RadioButton.Checked)
                    {
                        prediction.Symptom = Often15RadioButton.Text;
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal oftenpercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        oftenTotal = oftenTotal + oftenpercentage;
                        percentage = oftenpercentage;
                        oftenCount = oftenCount + 1;

                    }
                    else if (MostOfTheTime15RadioButton.Checked)
                    {
                        prediction.Symptom = MostOfTheTime15RadioButton.Text;
```

```
                        int total =
Convert.ToInt32(predictManager.GetSymptomValue(prediction));
                        decimal mostOfTheTimepercentage =
Convert.ToDecimal(predictManager.CalculatePercentage(total, prediction.Sum));
                        mostOfTheTimeTotal = mostOfTheTimeTotal +
mostOfTheTimepercentage;
                        percentage = mostOfTheTimepercentage;
                        mostOfTheTimeCount = mostOfTheTimeCount + 1;

                    }


                    list.Add(percentage);
                }


                }
            foreach (var data in list)
            {
                totalPercentage += data;
            }
            decimal predictPercentage = decimal.Divide(totalPercentage, 15);
            int Avarage = Convert.ToInt32(Math.Ceiling(predictPercentage));
            int rarelyAvarage = 0;
            int oftenAvarage = 0;
            int mostOftheTimeAvarage = 0;
            int occasionallyAvarage = 0;
            if (rarelyCount!=0)
            {
                rarelyAvarage = Convert.ToInt32(Math.Ceiling(decimal.Divide(rarelyTotal,
rarelyCount)));

            }
            if (occasionallyCount!=0)
            {
                 occasionallyAvarage =
Convert.ToInt32(Math.Ceiling(decimal.Divide(occasionallyTotal, occasionallyCount)));

            }
            if (mostOfTheTimeCount!=0)
            {
                 mostOftheTimeAvarage =
Convert.ToInt32(Math.Ceiling(decimal.Divide(mostOfTheTimeTotal, mostOfTheTimeCount)));

            }
            if (oftenCount!=0)
            {
                oftenAvarage = Convert.ToInt32(Math.Ceiling(decimal.Divide(oftenTotal,
oftenCount)));

            }



            var l = new List<int>() { rarelyAvarage, occasionallyAvarage,
oftenAvarage,mostOftheTimeAvarage };
            var max = l.Max();
            if (max==rarelyAvarage)
```

```
            {
                Response.Write("<script>alert('Your Dyslexia  probability is" + Avarage +
" % and affect chance is low ');</script>");
            }
            else if (max==occasionallyAvarage)
            {
                Response.Write("<script>alert('Your Dyslexia  probability is" + Avarage +
" % and affect chance is Mild');</script>");

            }
            else if (max==oftenAvarage)
            {
                Response.Write("<script>alert('Your Dyslexia  probability is" + Avarage +
" % and affect chance is Medium ');</script>");

            }
            else if (max==mostOftheTimeAvarage)
            {
                Response.Write("<script>alert('Your Dyslexia  probability is" + Avarage +
" % and affect chance is High ');</script>");

            }


        }
    }
}

using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using System.Web.Configuration;
using DyslexiaWebForm.Models;

namespace DyslexiaWebForm.Gateway
{
    public class PredictGateway
    {
        private SqlConnection connection = new SqlConnection();

        private string connectionString =

WebConfigurationManager.ConnectionStrings["DyslexiaConnection"].ConnectionString;

        public int GetSum(int questionNo)
        {
            connection.ConnectionString = connectionString;
            string query = "select Sum(Total) from t_reducing Where
QuestionNo='"+questionNo+"'";
            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            int rowAffected = Convert.ToInt32(command.ExecuteScalar().ToString());
            connection.Close();
            return rowAffected;

        }
```

```
        public int GetSymptomValue(Prediction prediction)
        {
            connection.ConnectionString = connectionString;
            int rowAffected = 0;
            if (IsDataExist(prediction)==true)
            {
                string query = "  select * from t_optimalreducing where QuestionNo='" +
prediction.QuestionNo + "' and Symptom='" + prediction.Symptom + "'  ";
                SqlCommand command=new SqlCommand(query,connection);

                connection.Open();
                SqlDataReader reader = command.ExecuteReader();
                if (reader.Read())
                {

                    rowAffected = (int) reader["Total"];
                }
                reader.Close();
                connection.Close();

            }
            else
            {
                string query = "  select * from t_reducing where QuestionNo='" +
prediction.QuestionNo + "' and Symptom='" + prediction.Symptom + "'  ";
                SqlCommand command=new SqlCommand(query,connection);
                connection.Open();
                SqlDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {

                    rowAffected = (int)reader["Total"];
                }
                reader.Close();
                connection.Close();
            }
            return rowAffected;
        }

        public bool IsDataExist(Prediction prediction)
        {
            connection.ConnectionString = connectionString;
            string query = "  select * from t_optimalreducing where QuestionNo='" +
prediction.QuestionNo + "' and Symptom='" + prediction.Symptom + "'  ";
            SqlCommand command=new SqlCommand(query,connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            bool IsRowExists = false;
            if (reader.HasRows)
            {
                IsRowExists = true;
            }
            connection.Close();
        return IsRowExists;
        }
    }
```