Final Report

QUERY AND SOLUTION TEXT MAPPING USING SELF ORGANIZING MAPS

GROUP G-6:

Prashant Patel
SUID 678713856
Ppatel03@syr.edu

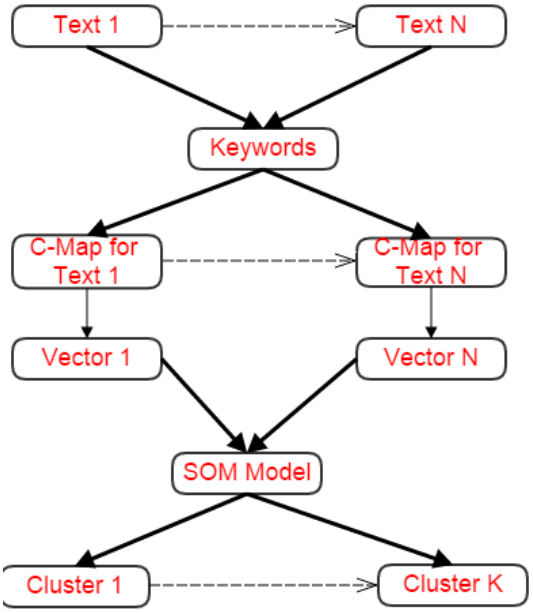
Monisha Lakshmipathi
SUID 755157314
mlakshmi@syr.edu

CSE 684: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Contents

QUAD Chart

System



Application

- **Medical Diagnosis**
- **Digital Library**
- **Thesauras website**
- **High frequency financial Data(Australian stock exchange)**
- **Meteorology and oceanography**

System Description

***Input*: X= {X_1…….X_n},**
where X_i corresponds to observations with respect to the i^{th} record

***Outputs*: C= {$C_{1.....}C_m$} ,**
where C_i is the clusters in solution space such that

C_i = {X_a….X_b}
such that {X_a….X_b} belong to the same cluster.

Timeline

Task No.	Task set	Completion Date
1	Vector generation using Cooccurence	**10th Nov**
2	Vector generation using TFID	**15th Nov**
3	SOM Initilization	**28th Nov**
4	SOM Mapping	**2nd Dec**
5	SOM Learning	**5th Dec**

Abstract

In this report, we have designed an automated diagnosis system using several patients' medical records. The paper discusses the solution where the SOM algorithm is trained using a dataset that comprises of several patients' medical history to construct a Kohonen map. Given the Kohonen map, an appropriate diagnosis is found by retrieving the text query from the new patient's symptoms

Introduction

There is a major growth in the interest of medical expert system for independent decisions in medical and engineering applications. Exponential increase has been documented in the accuracy of diagnostic tests, from careful observations of external symptoms and sophisticated lab tests and complex imaging methods that allow detailed non-enveloping internal examinations. The process of obtaining evidence to identify a probable cause of patient's key symptoms from all other possible causes of the symptom are known as establishing a medical diagnosis. Medical data mining has great potential for exploring the hidden patterns in the data sets of the medical field which can be utilized for medical diagnosis.

Data mining technology provides a user oriented approach to the underlying patterns in the data. However, there is a lack of effective analysis tools to discover hidden relationships and trends in data. The project aims at providing a tool that efficiently and accurately processes large observation-sets for hidden patterns and effectively maps the relationship in the diagnosis-prediction space with the help of SOM algorithm paired with a text retrieval feature.

The report gives a brief description of the proposed tool. We first discuss the state of art approaches and the assumptions that the approaches strongly believe. Later, the problem is well defined and the proposed solutions is explained in detail. Later sections of the report talks about the technical details of the proposed solution and issues it deals with.

Literature Survey

Healthcare related data mining is one of the most rewarding and challenging field of application in data mining and knowledge discovery. The reason for challenging is due to the data sets of huge, complex, diverse, hierarchical, time series and varying in quality. As the available healthcare datasets are fragmented and diffused in nature, thereby making the process of data integration is a highly challenging task [2].

Algorithms have been formed to for medical data pattern recognition and accurate results have been achieved by some of them [3].But they are applicable only for a small dataset. They fail to provide accurate results once you feed them with real time huge generalized medical datasets. Bayesian classification is used by Liad[4], an expert system program to estimate the posterior probabilities of various diagnoses under consideration, given the symptoms present in a case.

DXplain [5] is a Clinical decision support system (CDSS) that assists clinicians by generating stratified diagnoses based on user input of patient signs and symptoms, laboratory results, and other clinical findings. DXplain generates ranked differential diagnoses using a pseudo-probabilistic algorithm. But it still lies in the laboratory and hence is not out for public use.

The characteristics of the text representation are:

- The dimensionality is very large, but the underlying data is sparse. The document could have just about few words but the lexicon of the documents from which it is taken may be of the order of 100.
- An advantage of large documents is the existence of high correlated words. This implies that the number principal components in the document is much smaller compared to the feature space. This forces us to design the algorithm carefully during the clustering process such that it can account for the word correlations.
- It is important to perform normalization appropriately during the clustering task.

The sparse and high dimensional representation of the different documents necessitate the design of text-specific algorithms for document representation and processing, a topic heavily studied in the information retrieval literature where many techniques have been proposed to optimize document representation for improving the accuracy of matching a document with a query. Most of these techniques can also be used to improve document representation for clustering.

PROBLEM STATEMENT

Consider N unlabeled string inputs. It is tedious to map the relationship between N input texts and group them. Often the input data is large enough causing certain semantic and text-retrieval algorithms to slow down and create a performance hindrance for visualization, exploration and search. Hence this system should enable processing of large input data set and effectively map the relationship and hence accurately label them for further study.

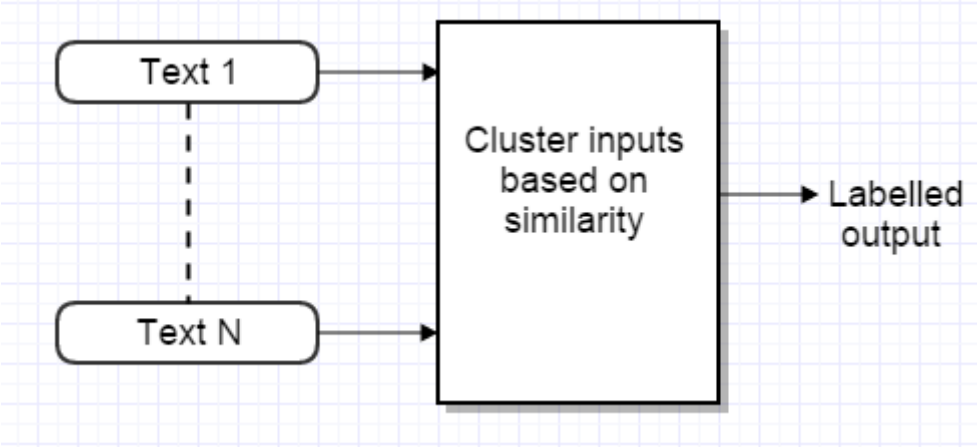


Fig 1: System Design

Problem Description

Consider N unlabeled string inputs of clinical and laboratory observations. It is tedious to map the relationship between N observations and group them under a certain category of disease. Often the observation set is large enough to create a performance hindrance of mapping the right set of query space (observations) to solution space (diagnosis). Hence this system should enable efficient and accurate processing of large observations set and effectively map the relationship and hence accurately label them with the right attribute diagnosis prediction space.

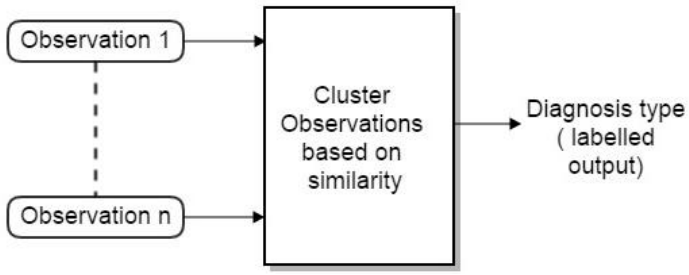


Fig.2: System design of the Diagnosis system

Mathematical Representation:

Input: X= $\{X_1 \ldots\ldots X_n\}$, where X_i corresponds to observations with respect to the i^{th} record

Outputs: C= $\{C_{1 \ldots\ldots} C_m\}$, where C_i is the clusters in solution space such that

$C_i = \{X_a \ldots . X_b\}$ such that $\{X_a \ldots . X_b\}$ belong to the same cluster.

Performance Metrics

The most basic and an efficient way to evaluate the performance of the system developed is to calculate average error probability. This method emphasizes on the number of errors generated over the number of inputs fed to the system. Initially lets take a set of labelled data as an input to the system. Feed the system with the labelled data without the labels. By doing so, we can measure the performance using the equation below,

Average Error $E= (N-W)^2$, where

N= Total number of inputs, W=Total number of outputs that were wrongly predicted.

Proposed Solution: Design and Implementation

In this section we discuss the solution of disease prediction and diagnosis based on classifying patients on the basis of clinical and laboratory type observations. The solution makes use of the SOM algorithm to efficiently cluster and organize the patient's clinical observations. For the purposes of dividing a given data set into m clusters, this approach makes use of the Self Organizing Map (SOM) in combination with the text retrieval approach. It thus provides a sound statistical basis for clustering.

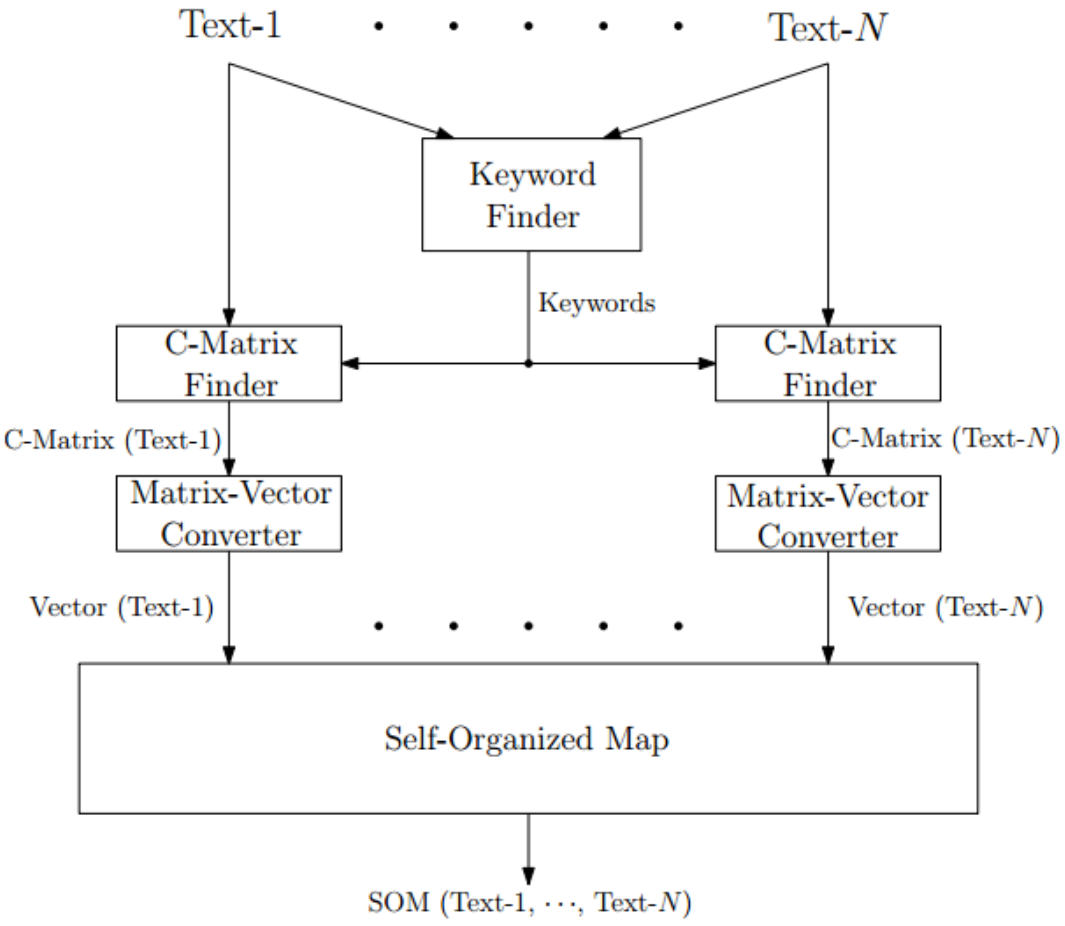


Fig:3 Detailed System Design

Proposed Solution: Design and Implementation

The solution runs through various stages as explained below,

Keywords Generation

Every input X_i is scanned for keywords. Keywords refer to the significant words present in the input set excluding all the propositions and other insignificant words that do not contribute any value to the system. Let K refer to the set of all keywords generated from the complete document set.

Co-occurrence Matrix

Co-occurrence matrix also known as dependence matrix is a statistical method of examining documents that considers the spatial relationship of keywords. The Co-occurrence functions characterize the documents by calculating how often pairs of keywords with specific relationship occur in a set of documents. It is the key data structure in going from user data to informative architecture [32]. Below fig is an illustration of the co-occurrence matrix in a document set.

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

The matrix can be efficiently stored by first calculating the co-occurrences of the keywords. Here a list of all the keywords that occur together is passed as the input to the matrix generator unlike a list of all the keywords. We can choose to include the pair of the same keywords occurring together, i.e. $\{K_i, K_i\}$ this gives rise to Co-occurrence matrix with diagonal keywords.

Vector Generation:

It is essential to quantize a text to perform various operations. By doing so, we reduce the dimensionality of the data. This step deals with vector generation as a result of text quantization.

$$V=\{ V_1 \ldots V_n\}$$

such that Vi be the normalized vector corresponding to matrix M_i and input X_i where M_i corresponds to co-occurrence matrix of order k×k.

Proposed Solution: Design and Implementation

$$\begin{matrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} = [1\,1\,0\,1\,0\,1]$$

SOM algorithm

A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map. The Kohonen technique creates a network that stores information in such a way that any topological relationships within the training set are maintained. SOM learns to classify the training data without any external supervision. Each node has a specific topological position and contains a vector of weights of the same dimension as the input vectors. Here the training data consists of vectors, V_i of k dimensions.

The training utilizes competitive learning. When a training example is fed to the network, its Euclidean distance to all weight vectors is computed. The neuron whose weight vector is most similar to the input is called the best matching unit (BMU). The weights of the BMU and neurons close to it in the SOM lattice are adjusted towards the input vector. The magnitude of the change decreases with time and with distance (within the lattice) from the BMU. The update formula for a neuron v with weight vector W(t+1) is

$$\mathbf{W(t+1) = W(t) + L(t)(V(t) – W(t))}$$

t = time-step
L = learning rate which decreases with time
W = weight
V = input vector

This process is repeated for each input vector for a number of cycles. The network winds up associating output nodes with groups or patterns in the input data set. If these patterns can be named, the names can be attached to the associated nodes in the trained net.

During mapping, there will be one single winning neuron: the neuron whose weight vector lies closest to the input vector. This can be simply determined by calculating the Euclidean distance between input vector and weight vector.

Why SOM algorithm?

SOM is a variation of k-means, here the means are "connected" in a sort of elastic 2D lattice, such that they move each other when the means update. This gives the self-organizing property, since the means will tend to pull their neighbor means closer, and the resulting 2D lattice is warped through the higher dimensional space in some smooth way rather than being folded on itself. The purpose of the clustering here is to achieve one of the 2D visualization for the unsupervised data.

Kohonen self-organizing maps provide cluster centers that have topological structure. For instance, you can restrict clusters to have a 2D grid topology. In this case, cluster centers next to each other are more similar than clusters farther away. K-means does not have this property.

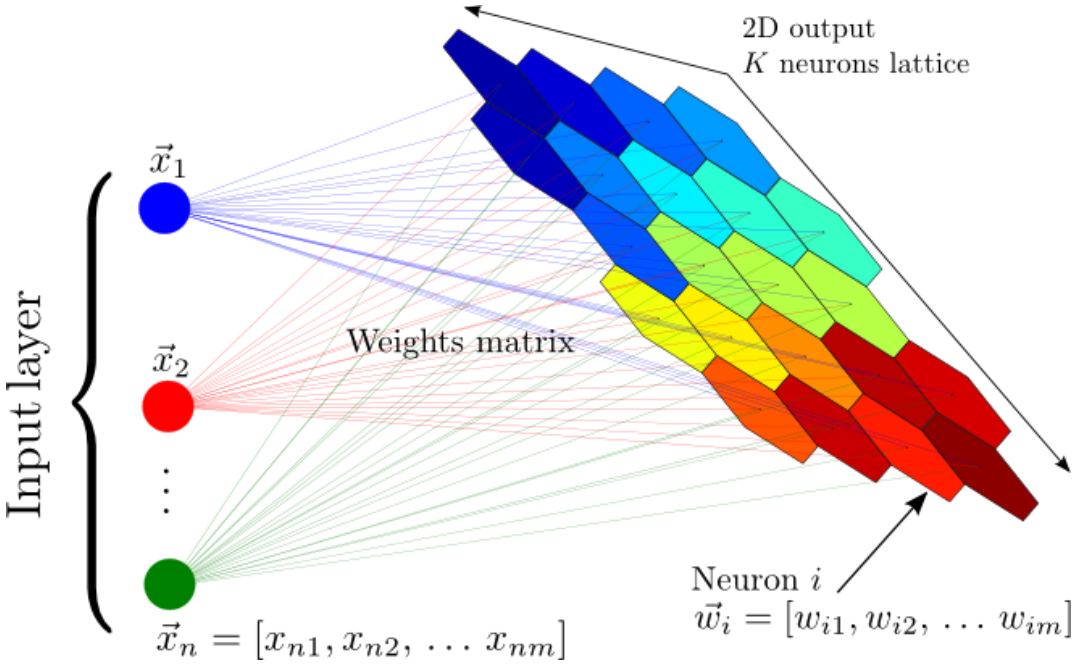


Fig:5

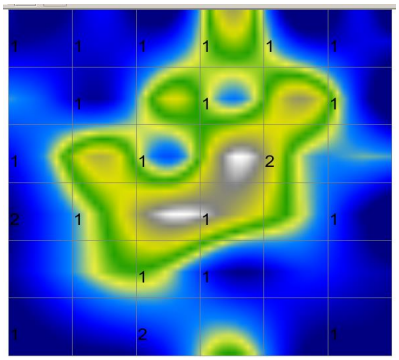
Test and Test Results

Input: Set of medical data related to different types of diseases [36].

Types of Output and Color Representations:

a. Smooth Data Histogram :

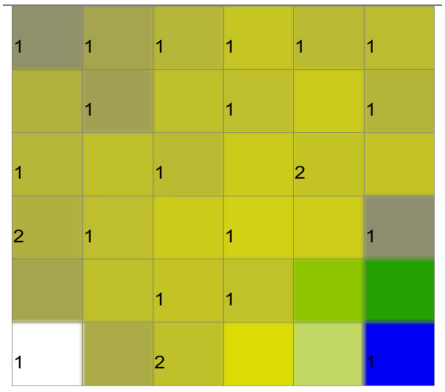
Smoothed Data Histograms are a visualization method that show the data distribution of the data samples by counting a number of most likely positions for each sample. The results can be visualized in a very intuitive landscape-like way, with islands and mountains in densely occupied regions and oceans in between [14].



For this example, the mountains are colored with grey and surrounded by green islands to denote densely populated regions.

b. Activity Histogram

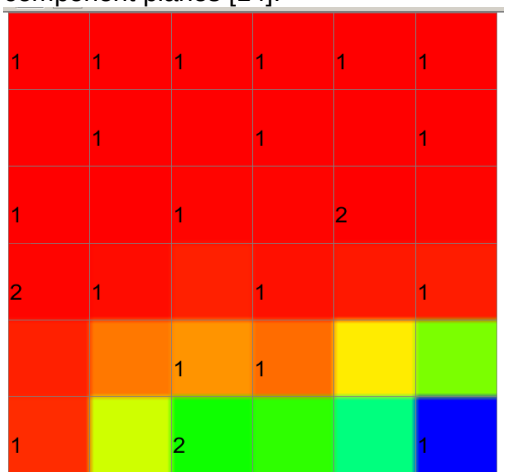
Displays a color-coding of the activity of a certain input vector over the whole map, i.e. in principle the vector's distances to all weight vectors [14].



For this example, dark grey color represents that BMU was mapped very closely to the vector.

c. Cluster Component Planes

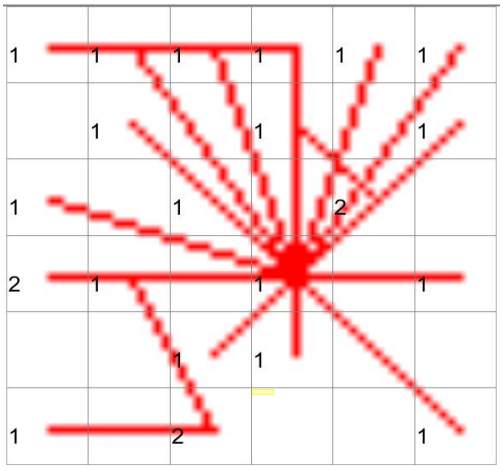
Component planes show the values of a single component (attribute, feature) over the whole map, thus they are useful to see in which map areas a certain attribute has high influence. They can also be used to identify related attributes, by **clustering** the component planes [14].



For this example, components with red color fall have high influence under same cluster.

d. Neighborhood KNN

The Neighborhood Graph is a novel visualization technique that takes the density of the data into account. The method defines graphs resulting from nearest neighbor- and radius-based distance calculations in the data space, and shows projections of these graph structures on the map. It can then be observed how relations between the data are preserved by the projection, yielding interesting insights into the topology of the mapping, and helping to identify outliers as well as dense regions [14].

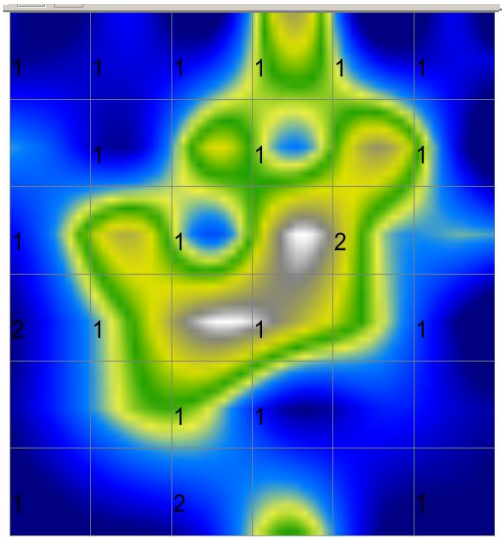


For this example, there are projections shown in red to represent relationship between the data.

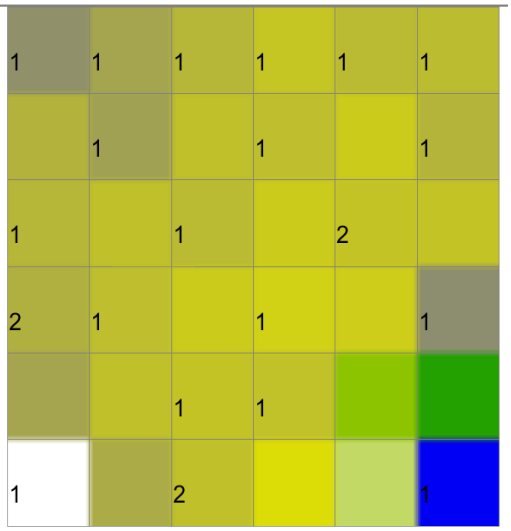
Test and Test Results

Output:

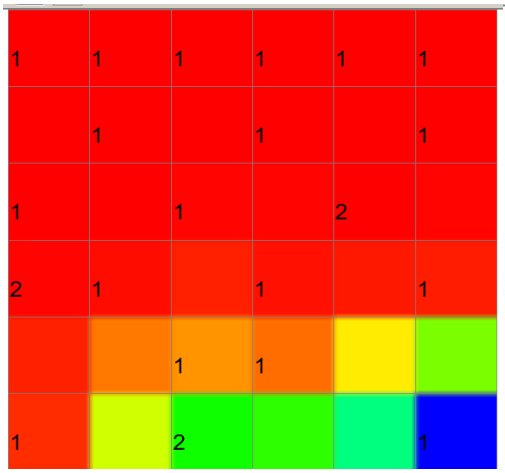
1. Co-occurrence Matrix with individual term weights



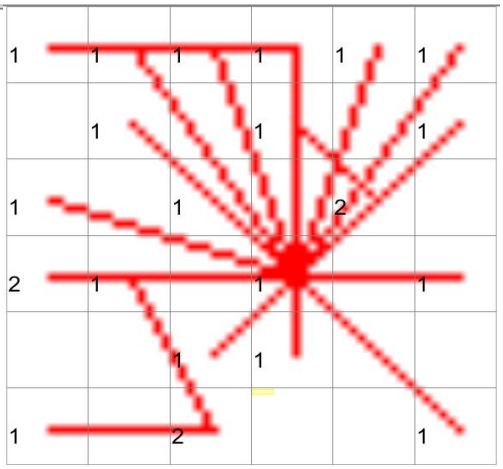
Smooth Data Histogram



Activity Histogram



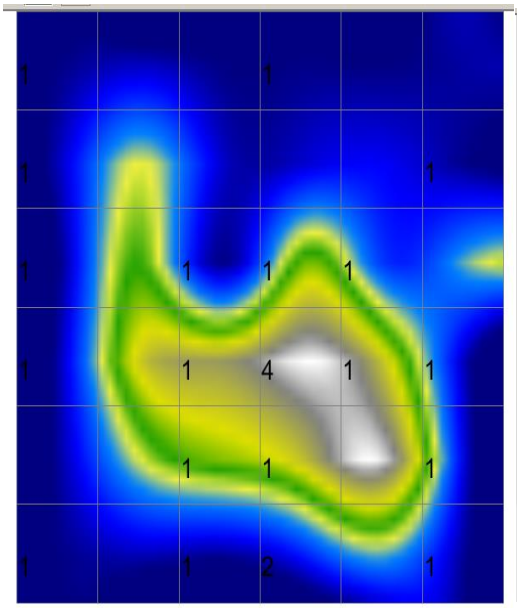
Cluster Component Planes



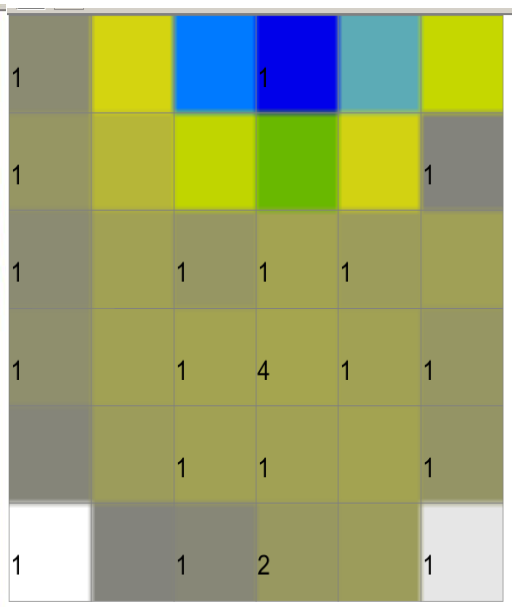
Neighborhood – knn

2. Co-occurrence Matrix of two adjacent words

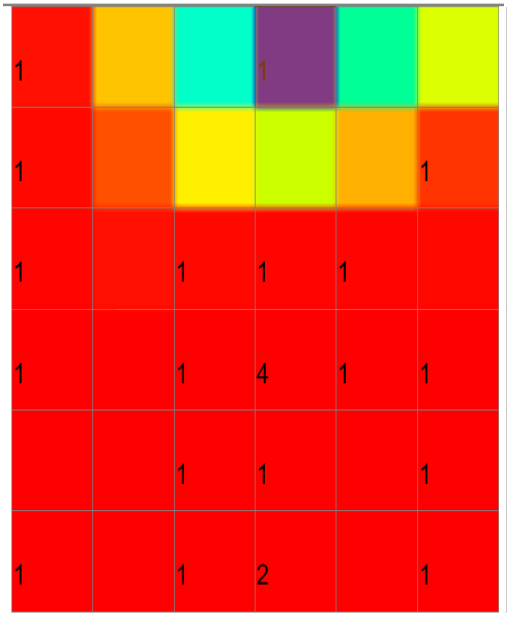
Test and Test Results



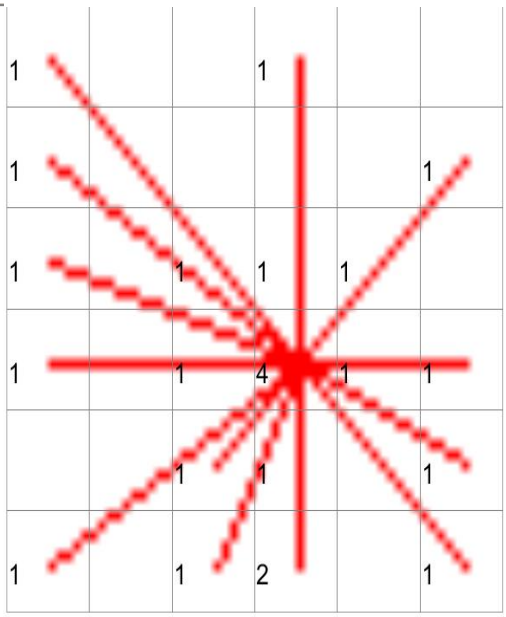
Smooth Data Histogram



Activity Histogram



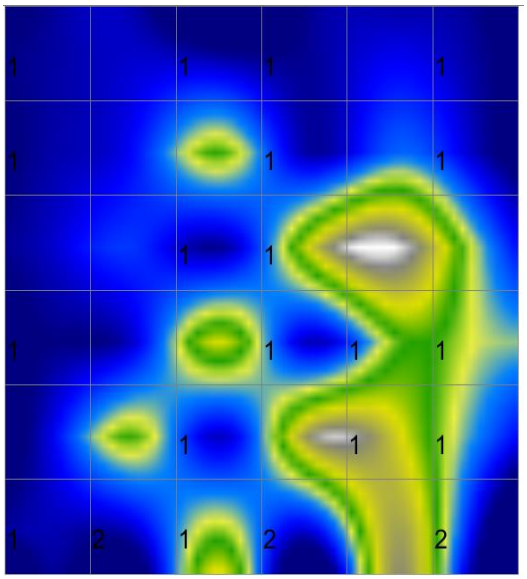
Cluster Component Planes



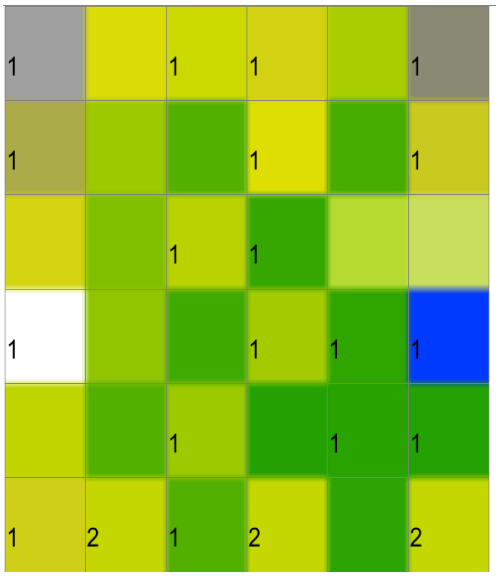
Neighborhood – knn

Test and Test Results

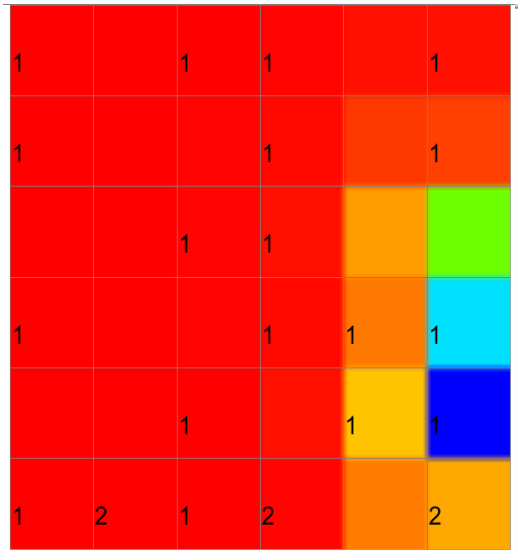
3. Term frequency and the inverse document frequency



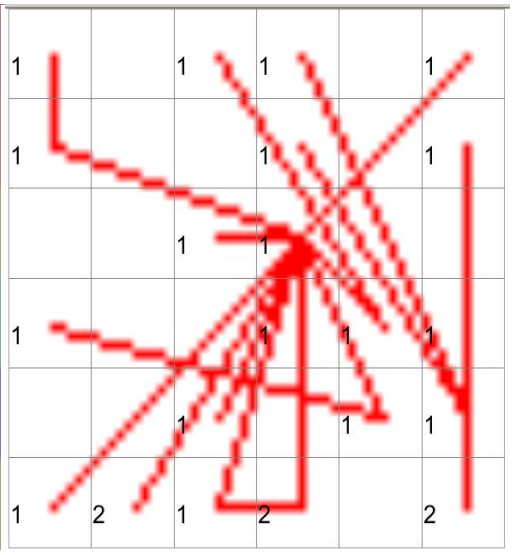
Smooth Data Histogram



Activity Histogram



Cluster Component Planes



Neighborhood – knn

Conclusion and future work

In this report, we have designed and implemented an automated diagnosis system using several patients' medical records. Having considered the challenges faced by the state-of-the-art approaches in the past literature, we have proposed a solution where the SOM algorithm is trained using a dataset that comprises of several patients' medical history to construct a Kohonen map. Given the Kohonen map, an appropriate diagnosis is found by retrieving the text query from the new patient's symptoms. In addition to evaluating the performance of our proposed system over several datasets, we have also compared our proposed system with another SOM-based system where texts are quantized into vectors using the TF-IDF method. In our test results, we have observed that our proposed approach outperforms the SOM-based system that employs TF-IDF method.

In the future, we will also consider a novel quantization approach for a TF-IDF based automated diagnosis system where N keywords are paired at a time so that stronger semantic relations are found in a given text. Another interesting approach for text-quantization is to consider a graph-based approach where a directional graph is constructed from a given text, with its nodes being the keywords and edges representing the semantic relationship between the keywords in the given text. We will characterize the asymmetric relationship between keywords within and across sentences in the given document by considering the in/out degrees at any given node in our future work.

Graph-based Approach

Consider a query [12] Croup is a childhood illness that affects the upper airways
After performing the “Keyword Generation” process, we get,

K= {Croup, Childhood, Illness, Affects, Upper, Airways}

Graph generated for the keyword “Croup” would be as shown in Fig 6.Modify the same graph for all the keywords in K.
The -----> represents the relations between the two words. Lesser the number, stronger is the relation.

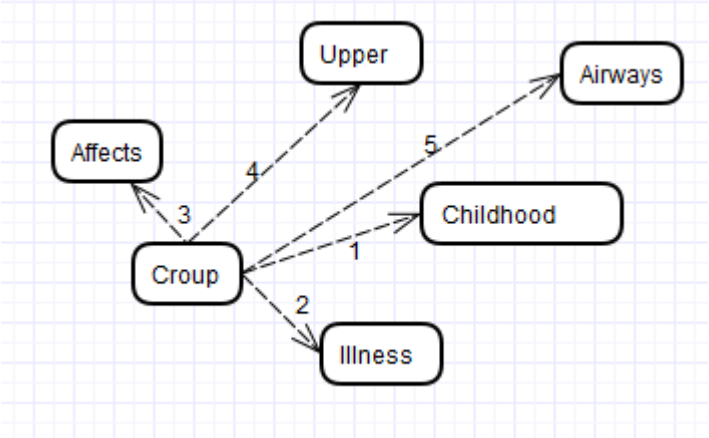


Fig 6: Graph based approach for Keyword “Croup”

References

1) http://www.ijcstjournal.org/volume-1/issue-2/IJCST-V1I2P4.pdf
2) K.J. and G.W. Moore Cios, "Uniqueness of medical data mining ," Artificial Intelligence in Medicine, pp. 1-24, 2002
3) M. Berlingerio, F. B. F. Giannotti, and F. Turini, “Mining clinical data with a temporal dimension: A case study,” in Proc. IEEE Int. Conf. Bioinf. Biomed., Nov. 2–4, 2007, pp. 429–436.
4) H. R. Warner and O. Bouhaddou, “Innovation review: Iliad—A medical diagnostic support program,” Top Health Inf. Manage., vol. 14, no. 4, pp. 51–58, 1994.
5) H. R. Warner and O. Bouhaddou, “Innovation review: Iliad—A medical diagnostic support program,” Top Health Inf. Manage., vol. 14, no. 4, pp. 51–58, 1994
6) “Text retrieval using Self-organized document maps” by Krista Lagus.
7) “Term-weighting approaches in automated text retrieval” by Gerard Salton and Christopher Buckley.
8) “How to use the Kohonen algorithm to simultaneously analyze individuals and modalities in a survey” by Marie Cottrell and Patrick Letremy.
9) Sky-Metaphor Visualisation for Self-Organising Maps - by Khalid Latif and Rudolf Mayer
10) The Self-Organizing Map, Teuvo Kohonen http://www.eicstes.org/EICSTES_PDF/PAPERS/The%20Self-Organizing%20Map%20(Kohonen).pdf
11) https://en.wikipedia.org/wiki/Self-organizing_map - SOM definition.
12) http://www.ai-junkie.com/ann/som/som1.html – SOM example.
13) https://www.willamette.edu/~gorr/classes/cs449/Unsupervised/SOM.html - SOM experimental conclusions.
14) http://www.ifs.tuwien.ac.at/dm/somtoolbox/ - SOM weight adjustment approach
15) http://www.nltk.org/ python NLTK library to generate stemmed words.
16) http://www.cs.odu.edu/~jbollen/IR04/readings/article1-29-03.pdf
17) http://arxiv.org/ftp/math/papers/0610/0610585.pdf
18) http://dl.acm.org/citation.cfm?id=1111221
19) http://dl.acm.org/citation.cfm?id=1745462.1745582
20) http://dl.acm.org/citation.cfm?id=1745462.1745569
21) http://publik.tuwien.ac.at/files/pub-inf_4711.pdf
22) http://www.eicstes.org/EICSTES_PDF/PAPERS/The%20Self-Organizing%20Map%20(Kohonen).pdf
23) https://en.wikipedia.org/wiki/Self-organizing_map
24) http://www.ai-junkie.com/ann/som/som4.html
25) https://www.willamette.edu/~gorr/classes/cs449/Unsupervised/SOM.html
26) http://www.dsi.unifi.it/~simone/Papers/Mysore07.pdf
27) http://www.charuaggarwal.net/text-cluster.pdf

References

28) https://www.pynchon.net/articles/10.7766/orbit.v1.2.44/
29) A. Jain, R. C. Dubes. Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs, NJ, 1998.
30) L. Kaufman, P. J. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis, Wiley Interscience, 1990.
31) P. Anick, S. Vaithyanathan. Exploiting Clustering and Phrases for Context-Based Information Retrieval. ACM SIGIR Conference, 1997.
32) R. Ng, J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. VLDB Conference, 1994.
33) D. Cutting, D. Karger, J. Pedersen, J. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. ACM SIGIR Conference, 1992
34) H. Schutze, C. Silverstein. Projections for Efficient Document Clustering, ACM SIGIR Conference, 1997.
35) http://boxesandarrows.com/beyond-cardsorting-free-listing-methods-to-explore-user-categorizations/
36) http://www.health.govt.nz/your-health/conditions-and-treatments/diseases-and-illnesses

Appendix: Source Code

1. **InputVectorCooccurenceDiagonalCreation.java**

```
/**
 *
 * Author : Prashant Patel and Monisha LakshmiPathi
 *
 * Description : Create the Co-occurence based on adjacent terms and individual terms.
 *
 */


package g6.ai.vectors;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import g6.ai.adapter.FileAdapter;
import g6.ai.stemmer.WordStemmer;
import helper.GenericHelper;
import constants.StemConstants;
import constants.VectorConstants;

public class InputVectorCooccurenceDiagonalCreation implements IVectorCreation {
        FileAdapter fileAdapter;

        public InputVectorCooccurenceDiagonalCreation() {
                // TODO Auto-generated constructor stub
                fileAdapter = new FileAdapter();
        }

        @Override
        public void createInputVectors(Map<Integer, String> inputDataMap,Map<Integer,
String> stemmedDataMap,
                        Map<Integer, List<Double>> inputVectorDataMap,List<String>
templateVectors, VectorConstants.VECTOR_CREATION_TYPE creationType, String fileName){


                inputDataMap = fileAdapter.getDataMapFromInputExcelFile(fileName);


                createStemmedTextAndGenerateVectors(inputDataMap, stemmedDataMap,
inputVectorDataMap, templateVectors);

        }

        /**
         *
         * create Cooccurences
         *
         * @param inputDataMap
         * @param stemmedDataMap
         * @param inputVectorDataMap
         */
```

Appendix: Source Code

```
	public void createStemmedTextAndGenerateVectors(Map<Integer, String>
inputDataMap,Map<Integer, String> stemmedDataMap,
			Map<Integer, List<Double>> inputVectorDataMap, List<String>
templateVectors){

		List<String> totalSelectedWords = new ArrayList<String>();


		for(Map.Entry<Integer, String> dataMap : inputDataMap.entrySet()){

			List<String> selectedWordsForEachDocument = new
ArrayList<String>();

			Integer key = dataMap.getKey();

			String data = dataMap.getValue();

			//stem each word for this text
			storeStemmedDataIntoList(data, totalSelectedWords,
selectedWordsForEachDocument);

			//create the stemmed word text for
			String stemmedText =
GenericHelper.convertListToString(selectedWordsForEachDocument);

			stemmedDataMap.put(key, stemmedText);
		}

		createInputVectorFromCooccurences(totalSelectedWords, stemmedDataMap,
inputVectorDataMap, templateVectors);
	}

	/**
	 *
	 * @param totalSelectedWords
	 * @param stemmedDataMap
	 * @param inputVectorDataMap
	 * @param templateVectors
	 */
	public void createInputVectorFromCooccurences(List<String> totalSelectedWords,
Map<Integer, String> stemmedDataMap,
			Map<Integer, List<Double>> inputVectorDataMap, List<String>
templateVectors){
		//create the template Vectors
		createTemplateVectors(totalSelectedWords, templateVectors);

		//iterate every stemmed text to create the Input Vector
		for(Map.Entry<Integer, String> entry : stemmedDataMap.entrySet()){

			Integer key = entry.getKey();

			String stemmedInputString = stemmedDataMap.get(key);

			//input vector
			List<Double> vector = new ArrayList<Double>();

			//iterate through the template vector to check for occurrence
			for(String dimension : templateVectors){

				//if the dimension exists in the input vector string
```

Appendix: Source Code

```
                                if(stemmedInputString.indexOf(dimension) != -1){
                                        //store the occurence of the dimension
                                        vector.add(new
Double(GenericHelper.getCountOfOccurences(stemmedInputString, dimension)));
                                } else {
                                        vector.add(new Double(0));
                                }

                        }

                        //store this data into input vector Map
                        inputVectorDataMap.put(key, vector);
                }


        }


        /**
         * creating the template
         *
         * @param totalSelectedWords
         * @param templateVectors
         */
        public void createTemplateVectors(List<String> totalSelectedWords, List<String>
templateVectors){

                int index = 0 ;

                while(index < totalSelectedWords.size()){
                        String stemmedWord  = totalSelectedWords.get(index);

                        templateVectors.add(stemmedWord);

                        // if the next index leads to the end of the String
                        if(index != totalSelectedWords.size() -1 ){
                                String nextStemmedWord = totalSelectedWords.get(index
+ 1 );


                                templateVectors.add(stemmedWord+" "+nextStemmedWord);
                        }

                        //incrementing the  index
                        index++;

                }

        }

        /**
         * stem each word from the document
         *
         * @param data
         * @param totalSelectedWords
         * @param selectedWordsForEachDocument
         */
        public void storeStemmedDataIntoList(String data, List<String>
totalSelectedWords,
                        List<String> selectedWordsForEachDocument ){
                //split by simple space, we will then remove stopwords
```

```
                String[] dataArray = data.split("\\s");

                //filtering the text for stop words
                for(String keyword : dataArray){
                        //if it is not in the list of stopwords
                        if(!StemConstants.STOP_WORDS_LIST.contains(keyword)){
                                //create the stemmer
                                WordStemmer wordStemmer = new WordStemmer();
                                String stemmedWord =
wordStemmer.getStemmedWord(keyword);
                                if(!totalSelectedWords.contains(stemmedWord)){
                                        totalSelectedWords.add(stemmedWord);
                                }
                                selectedWordsForEachDocument.add(stemmedWord);
                        }
                }

        }


        public static void main(String[] args) {
                // TODO Auto-generated method stub

        }

}
```

2. AIFacade.java

```
/**
 *
 * Author : Prashant Patel and Monisha Lakshmipati
 *
 * Description : Facade Design Pattern - presents simple interface of the complex
subsystem like
 *                               user input, vector file creation, Self-organizing map
training and running commands
 *                               by simply exposing the simple interfaces.
 *
 *
 */
package g6.ai.facade;

import g6.ai.adapter.FileAdapter;
import g6.ai.commands.CommandExecuter;
import g6.ai.inputs.ConsoleUserInput;
import g6.ai.vectors.IVectorCreation;
import g6.ai.vectors.InputVectorCreationFactory;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import constants.VectorConstants;

public class AIFacade {
        Map<Integer, List<Double>> inputVectorDataMap;
```

```
	Map<Integer, String> stemmedDataMap;
	Map<Integer, String> inputDataMap;
	List<String> templateVectors;

	public AIFacade() {
		// TODO Auto-generated constructor stub
		inputVectorDataMap = new HashMap<Integer, List<Double>>();
		stemmedDataMap = new HashMap<Integer,String>();
		inputDataMap = new HashMap<Integer, String>();
		templateVectors = new ArrayList<>();
	}

	public void executeSOM(){

		//take vector creation based user inputs
		int option = new ConsoleUserInput().getVectorCreationUserOption();

		//logic to create input vectors
		createVectors(option);

		//logic to write input vector and template vector files
		writeInputVectorAndTemplateVectorFiles();

		//logic to run SOM and display the SOMViewer
		executeSOMToolBoxCommands();
	}

	/**
	 * creating the input vectors
	 */
	public void createVectors(int option){

		VectorConstants.VECTOR_CREATION_TYPE vectorCreationType =
VectorConstants.VECTOR_CREATION_TYPE_MAP.get(option);

		IVectorCreation iVectorCreation =
InputVectorCreationFactory.getVectorCreationInstance(
				vectorCreationType);

		iVectorCreation.createInputVectors(inputDataMap, stemmedDataMap,
inputVectorDataMap, templateVectors,

	VectorConstants.VECTOR_CREATION_TYPE.COOCCURENCE_DIAGONAL,
				"input-data.xlsx");
		System.out.println(" ============== Input Vectors ==============");
		System.out.println(inputVectorDataMap);
		System.out.println(" ============== Template Vectors
==============");
		System.out.println(templateVectors);
	}

	/**
	 * writing data files
	 */
	public void writeInputVectorAndTemplateVectorFiles(){
		FileAdapter fileAdapter = new FileAdapter();
		fileAdapter.createInputAndTemplateVectorFiles(inputVectorDataMap,
templateVectors);
	}
```

```
	/**
	 *  execute the  commands of SOM Toolbox
	 */
	public void executeSOMToolBoxCommands(){
		CommandExecuter commandExecuter = new CommandExecuter();

		//run the command to create dwm file
		commandExecuter.runGrowingSOMAndSOMViewerCommandForSOMToolbox();

		//run the somviewer command to show the visualization
		//commandExecuter.runSOMViewerCommandForSOMToolbox();
	}


	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

3. SOMTrainerHelper.java

```
/**
 *
 * Helper class containing all necessary utilities for training SOM Nodes
 *
 */
package som.helper;

//static import
import static som.constants.ITrainerConstants.TIME_CONSTANT;
import static som.constants.ITrainerConstants.START_LEARN_RATE;
import static som.constants.ITrainerConstants.INTIAL_RADIUS;
import static som.constants.IMatrixConstants.somMatrixColumnSize;
import static som.constants.IMatrixConstants.somMatrixRowSize;

import static java.lang.Math.exp;


import java.util.ArrayList;
import java.util.List;

import som.beans.SOMDimensionRelation;

public class SOMTrainerHelper {

	public static double getExponentialComponent(int noOfIteration){
		return exp(-noOfIteration/TIME_CONSTANT);
	}

	public static double getNeighbourhoodRadius(int noOfIteration){
		return INTIAL_RADIUS * getExponentialComponent(noOfIteration);
	}

	//it is the theta
	public static double getBMUDistanceFactor(double dist, double radius){
```

```
                return exp(-dist*dist/(2*radius*radius));
        }

        public static double getLearningRate(int noOfIteration){
                return START_LEARN_RATE * getExponentialComponent(noOfIteration);
        }

        public static List<SOMDimensionRelation> getListofSOMVectorsUnderRadius(int
radius, int iPosOfBMU, int jPosOfBMU){
                List<SOMDimensionRelation> dimensionList = new
ArrayList<SOMDimensionRelation>();

                for(int start = 1; start <= radius; start++){
                        storePosSpirally(iPosOfBMU-start, jPosOfBMU-start,
iPosOfBMU+start, jPosOfBMU+start, dimensionList);
                }
                return dimensionList;
        }


        public static void storePosSpirally(int minRow, int minCol, int maxRow, int
maxCol,List<SOMDimensionRelation> dimensionList){
                int i = minRow;
                int j = minCol;
                while(j<=maxCol){
                        if(j < somMatrixColumnSize){
                                addDimensionToList(i, j, dimensionList);
                                System.out.print("("+i+","+j+") ");
                        }
                        j++;
                }
                j--;
                i++;
                while(i <= maxRow){
                        if(i < somMatrixRowSize){
                                addDimensionToList(i, j, dimensionList);
                                System.out.print("("+i+","+j+") ");
                        }
                        i++;
                }
                i--;
                j--;
                while(j >= minCol){
                        if(j >= 0){
                                addDimensionToList(i, j, dimensionList);
                                System.out.print("("+i+","+j+") ");
                        }
                        j--;
                }
                j++;
                i--;
                while(i >= minRow){
                        if(i >= 0){
                                addDimensionToList(i, j, dimensionList);
                                System.out.print("("+i+","+j+") ");
                        }
                        i--;
                }
                System.out.println();
```

```
        }

        public static void addDimensionToList(int i , int j, List<SOMDimensionRelation>
dimensionList){
                SOMDimensionRelation dimension = new SOMDimensionRelation(i, j, 0.0);
                dimensionList.add(dimension);
        }

        public static List<Integer> getAdditionOfTwoList(List<Integer> list1,
List<Double> list2){
                List<Integer> resultList = new ArrayList<Integer>();
                for(int i = 0; i<list1.size(); i++){
                        int value = (int) ((list1.get(i)+list2.get(i)));
                        resultList.add(i, value );
                }
                return resultList;
        }

        public static List<Double> getSubtractionOfTwoList(List<Integer> list1,
List<Integer> list2, double multiplicationFactor){
                List<Double> resultList = new ArrayList<Double>();
                for(int i = 0; i<list1.size(); i++){
                        Double value = (double) ((list1.get(i)-
list2.get(i))*multiplicationFactor);
                        resultList.add(i, value );
                }
                return resultList;
        }
}
```

4. SOMTrainer.java

```
/**
 *
 * Adjust the weights of the SOM using decaying formula given in the following url
 * http://www.ai-junkie.com/ann/som/som1.html
 *
 */
package som.trainer;

//static import

import static som.constants.ITrainerConstants.MAX_NO_ITERATIONS;
import static som.helper.SOMTrainerHelper.getNeighbourhoodRadius;
import static som.helper.SOMTrainerHelper.getLearningRate;
import static som.helper.SOMTrainerHelper.getListofSOMVectorsUnderRadius;
import static som.helper.SOMTrainerHelper.getSubtractionOfTwoList;
import static som.helper.SOMTrainerHelper.getAdditionOfTwoList;
import static som.helper.SOMTrainerHelper.getBMUDistanceFactor;

import static som.constants.IMatrixConstants.somMatrixColumnSize;
import static som.constants.IMatrixConstants.somMatrixRowSize;
import static som.constants.IMatrixConstants.somMatrix;
import static som.constants.IGenericConstants.randomSOMVectorMap;



// standard java library static import
```

Appendix: Source Code

```
import static java.lang.Math.abs;

import java.util.ArrayList;
import java.util.List;


import som.beans.SOMDimensionRelation;
import som.helper.GenericHelper;



public class SOMTrainer implements ISOMTrainer{
	private int neighbourRadius = -1;
	private double learningRate = -1;


	public void adjustTrainingWeights(int noOfIteration, int iPosOfBMU, int
jPosOfBMU, List<Integer> inputVector){
		if(noOfIteration <= MAX_NO_ITERATIONS){
			calculateDependentValues(noOfIteration);
			List<SOMDimensionRelation> dimensionRelationList =
getListofSOMVectorsUnderRadius(
					this.neighbourRadius, iPosOfBMU, jPosOfBMU);
			List<Integer> bmuCoordinate = new ArrayList<Integer>();
			bmuCoordinate.add(iPosOfBMU);
			bmuCoordinate.add(jPosOfBMU);

	storeNewWeightOfNeighbourhoodVectors(dimensionRelationList,inputVector,bmuCoord
inate);

		}
	}

	public void calculateDependentValues(int noOfIteration){
		this.neighbourRadius = (int)
abs(getNeighbourhoodRadius(noOfIteration));
		this.learningRate = getLearningRate(noOfIteration);
		System.out.println("Neighbourhood Radius at iternation
"+noOfIteration+" : "+neighbourRadius);
		System.out.println("Learning Rate at iternation "+noOfIteration+" :
"+learningRate);
	}

	public void storeNewWeightOfNeighbourhoodVectors(List<SOMDimensionRelation>
dimensionRelationList,
			List<Integer> inputVector,List<Integer> bmuCoordinate){

		for(SOMDimensionRelation dimension : dimensionRelationList){
			int i = dimension.getxPosition();
			int j = dimension.getyPosition();
			if(!(i<0 || j<0 || i>=somMatrixRowSize ||
j>=somMatrixColumnSize)){
				int vectorPosition = somMatrix[i][j];
				List<Integer> somVector =
randomSOMVectorMap.get(vectorPosition);
				List<Integer> somVectorCoordinate = new
ArrayList<Integer>();
				somVectorCoordinate.add(i);
				somVectorCoordinate.add(j);
				double bmuDistanceFactor = getBMUDistanceFactor(
```

Appendix: Source Code

```
        GenericHelper.computeEuclideanDistanceForCoordinates(bmuCoordinate,
somVectorCoordinate),
                                                    this.neighbourRadius);
                                    List<Double> intermediateVector =
getSubtractionOfTwoList(somVector, inputVector,
                                                    bmuDistanceFactor*this.learningRate);
                                    List<Integer> newSOMVector =
getAdditionOfTwoList(somVector, intermediateVector);
                                    randomSOMVectorMap.put(vectorPosition, newSOMVector);
                            }

                    }
            }



}
```