DYNAMIC UNSUPERVISED FEEDFORWARD NEURAL NETWORK CLUSTERING

ROYA ASADI

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

DYNAMIC UNSUPERVISED FEEDFORWARD NEURAL NETWORK CLUSTERING

ROYA ASADI

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

ABSTRACT

Artificial neural networks are computational models inspired by neurobiology for enhancing and testing computational analogues of neurons. In a feedforward neural network (FFNN), data processing occurs in only one forward interconnection from the input layer to the output layer without any backward loop. Unsupervised FFNN (UFFNN) clustering has great capabilities such as inherent distributed parallel processing architectures, adjusting the interconnection weights to learn and divide data into meaningful groups with special goals, classifying related data into similar groups without using any class label, controlling noisy data and learning the types of input data values based on their weights and properties. Generally in real environments, dynamic data is high volume and dimensional, therefore, the online dynamic UFFNN (ODUFFNN) clustering methods should be developed to have online incremental learning capability. Incremental learning refers to the ability of repeatedly training a network using new data or deleting unnecessary data, without destroying outdated prototype patterns. The ODUFFNN should also be compatible with the changes that occur in continuous data and should be able to control noisy data. We reviewed and investigated current ODUFFNN clustering methods and identified their limitations, main problems such as high training time, low accuracy and high time complexity and memory complexity of clustering, and some reasons of these problems.

In order to overcome the problems, we developed a dynamic UFFNN (DUFFNN) clustering model with only one epoch training. Dynamically after each entrance of the online input data, the DUFFNN learns and stores important information about the current online data, such as the non-random weights and consequently completes a codebook of the weights. Then, a unique and standard weight vector is extracted and updated from the codebook. Subsequently, a single layer DUFFNN calculates the exclusive distance threshold of each data based on the standard weight vector, and clusters the data based on the exclusive distance threshold. Based on the literature, after learning in order to improve the quality of the DUFFNN clustering result, the model assigns a class label to the input data through the training data. The class label of each initially unlabeled input data is predicted by considering a linear activation function and the exclusive distance threshold. Finally, the number of clusters and the density of each cluster are updated.

For evaluation purposes, the clustering performances of the DUFFNN were compared with several related clustering methods using the various datasets from the University of California at Irvine Machine Learning Repository, which illustrated great results. For example, the accuracy of the proposed model was measured through the number of clusters, the quantity of corectly classified nodes and also an *F-measure* which was 97.71% of the Breast Cancer, 97.24% of Iris, 73.41% of Spam, 90.52% of SPECT Heart, 86.62% of SPECTF Heart, 52.57% of Musk1, 84.31% of Musk2, 66.07% of Arcene, and 27.25% of Yeast datasets respectively, and the superior *F-measure* results between 98.14% and 100% accuracies for the breast cancer dataset from the University of Malaya Medical Center to predict the survival time of the patients.

ABSTRAK

Rangkaian neural buatan merupakan model komputer yang mendapat inspirasi dari bidang neurobiologi untuk ujian dan peningkatan neuron-neuron berasaskan analog-analog komputer. Bagi suatu rangkaian neural feedforward (FFNN), pemprosesan data berlaku dalam siri sambungan ke hadapan dari lapisan input ke lapisan output tanpa ulangan ke belakang. Pengelompokan unsupervised FFNN (UFFNN) mempunyai keupayaan yang hebat seperti memiliki senibina pemprosesan edaran selari, pelaras pemberat sambungan untuk belajar dan membahagi data kepada kumpulan-kumpulan yang mempuyai maksud dan matlamat-matlamat khas, pengkelasan data berkaitan kepada kumpulan-kumpulan serupa tanpa menggunakan apa-apa label kelas, pengawalan data tidak tepat dan pembelajaran jenis-jenis input data berdasarkan pemberat dan sifat masing-masing.

Kebiasaannya dalam persekitaran sebenar, data bergerak adalah besar dan mempunyai dimensi tinggi, untuk itu, kaedah-kaedah pengelompokan dinamik secara atas-talian UFFNN (ODUFFNN) perlu melibatkan keupayaan pembelajaran secara bertingkat. Pembelajaran bertingkat merujuk kepada keupayaan untuk berlatih secara berulang dengan menambah atau membuang nod-nod data dalam pembelajaran atas-talian tanpa menghapuskan corak-corak prototaip yang ketinggalan zaman. ODUFFNN sepatutnya juga serasi dengan perubahan-perubahan yang berlaku dalam data berterusan dan mampu mengawal data tidak tepat. Kami telah mengkaji dan menyiasat kaedah-kaedah pengelompokan ODUFFNN semasa dan mengenalpasti had-had mereka; masalah-masalah utama seperti masa latihan tinggi, ketepatan rendah serta memori dan kompleks masa yang tinggi sewaktu proses pengelompokan; dan beberapa faktor yang menyebabkan masalah-masalah tersebut.

Untuk mengatasi masalah-masalah tersebut, kami telah membangunkan satu model pengelompokan UFFNN yang dinamik (DUFFNN) dengan satu latihan epoch. Secara dinamiknya selepas setiap kemasukan data input secara atas-talian, DUFFNN akan mempelajari dan menyimpan informasi penting berkenaan data atas-talian semasa, seperti pemberat-pemberat tidak-rawak dan menyempurnakan satu buku-kod untuk pemberat-pemberat itu. Kemudian, satu vektor pemberat yang mengikut piawai dan unik akan diekstrak dan dikemaskini dari buku-kod tersebut. Selaras dengan itu, satu lapisan tunggal DUFFNN akan mengira ambang jarak khusus untuk setiap data berdasarkan vektor pemberat piawai itu. Data-data akan dikelompokan berdasarkan ambang jarak khusus yang ditentukan. Berdasarkan kajian literasi, untuk meningkatkan kualiti pengelompokan DUFFNN, suatu model itu perlu menetapkan satu label kelas kepada data input melalui data latihan. Label kelas untuk setiap input data yang tidak berlabel adalah diramalkan dengan mempertimbangkan satu fungsi pengaktifan linear dan ambang jarak khusus tersebut. Akhirnya, bilangan kelompok-kelompok dan ketumpatan setiap kelompok akan dikemaskinikan.

Untuk tujuan penilaian, prestasi pengelompokan DUFFNN dibandingkan dengan beberapa kaedah-kaedah pengelompokan berkaitan menggunakan pelbagai set-set data dari Irvine Machine Learning Repository dari University of California, yang memberikan keputusan yang menggalakkan. Ketepatan model yang dibangunkan telah diukur mengikut bilangan kelompok-kelompok, kuantiti nod-nod yang telah diklasifikasikan secara tepat, dan juga suatu ukuran F untuk set-set data yang berasingan iaitu 97.71% untuk Kanser Payudara, 97.24% untuk Iris, 73.41% untuk Spam, 90.52% untuk SPECT Heart, 86.62% untuk SPECTF Heart, 52.57% untuk Musk1, 84.31% untuk Musk2, 66.07% untuk Arcene, dan 27.25% untuk dataset Yeast, yang mana ukuran F tertinggi adalah di antara 98.14% dan 100% ketepatan untuk set data Kanser Payudara dari Pusat Perubatan Universiti Malaya untuk meramalkan jangkahayat para pesakit.

ACKNOWLEDGEMENTS

First of all, thank to Allah, the most Gracious and most Merciful.

Many people have made contributions in completing this work. My deepest appreciation and gratitude to Assoc. Prof. Datin Dr. Sameem Abdul Kareem selamat for her ideas and intellectual experiences that paved the way to this research work. Also, my most affectionate gratitude to my beloved parents, sisters, and brother that support me financially and spiritually.

Finally, sincere thanks towards everyone in the Faculty of Computer Science and Information Technology, University of Malaya for their intellectual and social contributions, especially for making my research years to be more memorable and enjoyable times.

TABLE OF CONTENTS

ABSTRACT iii

ABSTRAK v

ACKNOWLEDGEMENTS vii

TABLE OF CONTENTS viii

LIST OF FIGURES xiv

LIST OF TABLES xix

LIST OF SYMBOLS AND ABBREVIATIONS xxiii

LIST OF APPENDICES xxvi

CHAPTER 1: INTRODUCTION 1

1.1 Background and Motivation 1

1.1.1 Fundamental Unsupervised Feedforward Neural Network Clustering 2

1.1.2 An Overview of Current Online Dynamic Unsupervised Feedforward Neural Network Clustering 4

1.2 Problem Statement 8

1.2.1 Some Reasons of the Problems Related to the Structure and Features of the Data 11

1.2.2 Some Reasons of the Problems Related to the Topology and Algorithm of the Current ODUFFNN Clustering Method 13

1.3 Objectives of Research 16

1.4 Preliminary Research Questions 17

1.5 Scope of Research 18

1.6 Research Methodology 20

1.7 Overview of the Contributions of Thesis 23

1.8 Organization of Thesis 24

CHAPTER 2: CONCEPT IN UNSUPERVISED CLUSTERING 26

2.1 Introduction 26

2.2 Overviews of Unsupervised Clustering 26

2.3 Types of Data in Unsupervised Clustering 30

2.4 Distance Functions For Measuring the Dissimilarity of Objects 32

2.4.1 Distance Measuring of Interval-Scaled Variable 32

2.4.2 Distance Measuring Binary Variables 33

2.4.3 Distance Measuring Nominal Variables 35

2.4.4 Distance Measuring Ordinal Variables and Interval-Scaled Variables 35

2.4.5 Distance Measuring Ratio-Scaled Variables 36

2.4.6 Distance Measuring of Vector Object 36

2.5 Clustering Evaluation 37

2.5.1 External Indexes 37

2.5.2 Internal Index 44

2.5.3 Relative Index 45

2.6 The Overview of the Main Unsupervised Clustering Methods 46

2.6.1 Partitioning Clustering 47

2.6.2 Hierarchical Clustering 49

2.6.3 Density Based Clustering 51

2.6.4 Model-based Clustering 53

2.6.5 Grid-based Clustering 55

2.6.6 Other Categories of Clustering 57

2.7 The Time Complexity and Memory Complexity of the Main Unsupervised Clustering Methods 57

2.8 Summary 59

CHAPTER 3: CONCEPT IN Online Dynamic Unsupervised Feedforward Neural Network Clustering 60

3.1 Introduction 60

3.2 Concepts in Feedforward Neural Networks 61

3.2.1 Neural Network Models 62

3.2.2 Perspectives of Learning Problems 67

3.2.3 Architecture of Neural Networks 69

3.2.4 History of Early Artificial Feedforward Neural Network Models 71

3.2.5 Activation Functions 74

3.2.6 Preprocessing Techniques of Feedforward Neural Networks 77

3.3 Unsupervised Feedforward Neural Network Clustering 88

3.3.1 Self-organization Map 91

3.3.2 Growing Neural Gas 93

3.4 The Time Complexity and Memory Complexity of the Unsupervised Feedforward Neural Network Clustering and Some Related Clustering Methods 95

3.5 Online Dynamic Unsupervised Feedforward Neural Network Clustering 95

3.5.1 Evolving Self-organizing Map 96

3.5.2 Enhanced Self-organizing Incremental Neural Network for Online Unsupervised Learning 99

3.5.3 Dynamic Self-organizing Map 101

3.5.4 Incremental Growing with Neural Gas Utility Parameter 102

3.5.5 An Enhancing Dynamic Self-organizing Map 103

3.5.6 Enhanced Incremental Growing Neural Gas 104

3.6 Comparison and Discussion 105

3.7 Summary 108

CHAPTER 4: RESEARCH METHODOLOGY 109

4.1 Introduction 109

4.2 Approaches to Research 109

4.2.1 Reviewing Related Literature 109

4.2.2 Problem Formulation 109

4.2.3 Definition of the Research Objectives 110

4.2.4 Proposed Model 110

4.2.5 System Design 112

4.2.6 System and Data Requirements for Running Experiment 113

4.2.7 Experimental Evaluation 124

4.3 Summary 127

CHAPTER 5: SYSTEM DESIGN 128

5.1 Introduction 128

5.2 The Real Unsupervised Feedforward Neural Network Clustering 128

5.2.1 Analysis of the RUFFNN Clustering Method 130

5.3 The Real Semi-supervised Feedforward Neural Network Clustering 141

5.4 The Dynamic Unsupervised Feedforward Neural Network Clustering 143

5.4.1 Analysis of the DUFFNN Clustering Method 145

5.5 The Dynamic Semi-supervised Feedforward Neural Network Clustering 156

5.6 The Time Complexity and Memory Complexity of the Purposes 157

5.7 Summary 160

CHAPTER 6: EXPERIMENTAL RESULTS AND EVALUATION ON THE RUFFNN AND RSFFNN CLUSTERING METHODS 162

6.1 Introduction 162

6.2 Experimental Evaluation on the RUFFNN and RSFFNN Clustering Methods .. 162

6.2.1 The RUFFNN and RSFFNN Clustering on the Breast Cancer Wisconsin.. 163

6.2.2 The RUFFNN and RSFFNN Clustering on Iris 166

6.2.3 The RUFFNN and RSFFNN clustering on Spambase 169

6.2.4 The RUFFNN and RSFFNN clustering on Arcene 172

6.2.5 The RUFFNN and RSFFNN clustering on Yeast 175

6.2.6 The RSFFNN Clustering on the Breast Cancer (UMMC) 177

6.3 Summary.. 180

CHAPTER 7: EXPERIMENTAL RESULTS AND EVALUATION ON THE DUFFNN AND DSFFNN CLUSTERING METHODS... 181

7.1 Introduction 181

7.2 Experimental Evaluation on the DUFFNN and DSFFNN Clustering................. 181

7.2.1 The DUFFNN and DSFFNN Clustering on the Breast Cancer Wisconsin..182

7.2.2 The DUFFNN and DSFFNN Clustering on the Iris............................... 186

7.2.3 The DUFFNN and DSFFNN Clustering on Spambase.......................... 189

7.2.4 The DUFFNN and DSFFNN Clustering on SPECT Heart 193

7.2.5 The DUFFNN and DSFFNN Clustering on the SPECTF Heart............ 197

7.2.6 The DUFFNN and DSFFNN Clustering on MUSK1 200

7.2.7 The DUFFNN and DSFFNN Clustering on the MUSK2 205

7.2.8 The DUFFNN and DSFFNN Clustering on Arcene 209

7.2.9 The DUFFNN and DSFFNN Clustering on Yeast................................ 212

7.2.10 The DSFFNN Clustering on the Breast Cancer (UMMC)..................... 215

7.3 Summary.. 217

CHAPTER 8: CONCLUSIONS AND FUTURE RESEARCH 218

8.1 Introduction 218

8.2 Achievement to Objectives of Research 218

8.3 Summary of the results and finding 219

8.4 Contributions of the research 223

8.5 Recommendation and Future Directions 228

REFERENCES 229

LIST OF PUBLICATIONS AND PAPERS PRESENTED 241

APPENDIX A 245

APPENDICES B 260

LIST OF FIGURES

Figure 1-1: The problems of the ODUFFNN clustering methods and two reasons of these problems 11

Figure 1-2: The solutions for the mentioned problems 15

Figure 1-3: The framework of the research methodology 22

Figure 2-1: The relations of data mining and other fields 27

Figure 2-2: The KDD process 29

Figure 2-3: An example of two clusters 30

Figure 2-4: An example of irregular shape cluster 30

Figure 2-5: The data matrix and the dissimilarity matrix 31

Figure 2-6: A sample of the distances between the clusters and within each cluster 46

Figure 2-7: The *K*-means clustering method 48

Figure 2-8: The hierarchical clustering method 49

Figure 2-9: The density based clustering method 51

Figure 2-10: The DBSCAN clustering method 52

Figure 2-11: An example of the COBWEB in decision tree clustering method 54

Figure 2-12: The grid-based based clustering method 55

Figure 3-1: The neuron 63

Figure 3-2: A sample neural network 66

Figure 3-3: Single-layer perceptrons feedforward network 69

Figure 3-4: Multilayer perceptrons feedforward network 70

Figure 3-5: Recurrent networks 70

Figure 3-6: Single layer unsupervised feedforward neural network with Hebbian learning 72

Figure 3-7: The single layer perceptron sample 73

Figure 3-8: The multiLayer perceptron sample 74

Figure 3-9: Identity function 74

Figure 3-10: Binary step function 75

Figure 3-11: Sigmoid function 76

Figure 3-12: Bipolar sigmoid function 76

Figure 3-13: A sample topology of the competitive clustering 89

Figure 3-14: A simple topology of the SOM 91

Figure 3-15: One stage of the SOM process as an example 92

Figure 3-16: Classification of Iris dataset by using the GNG (Costa & Oliveira, 2007) 94

Figure 4-1: The sample of breast cancer Wisconsin dataset 115

Figure 4-2: The sample of the Iris dataset 115

Figure 4-3: The sample of the Spambase dataset 116

Figure 4-4: The sample of SPECT Heart dataset 117

Figure 4-5: The sample of SPECTF Heart dataset 118

Figure 4-6: The sample of the Musk1 dataset 119

Figure 4-7: The sample of the Musk2 dataset 120

Figure 4-8: The sample of the Arcene dataset 121

Figure 4-9: The sample of the Yeast dataset 122

Figure 4-10: The sample of breast cancer from UMMC dataset 123

Figure 5-1: The design of real unsupervised feedforward neural network model for clustering 129

Figure 5-2: The algorithm of the real unsupervised feedforward neural network clustering 131

Figure 5-3: Standard normal distribution for each attribute value of input instance X_i 133

Figure 5-4: Smoothing the *BMW* components of the breast consin (original) dataset by using the RUFFNN method 137

Figure 5-5: Distribution of the normal input data attributes and their distances from the gravity centre of the dataset 139

Figure 5-6: The outlook of clustering the Iris dataset by RUFFNN before using class labels 140

Figure 5-7: The design of changed section of the RUFFNN suitable for real semi-supervised feedforward neural network clustering method 141

Figure 5-8: The algorithm of the real semisupervised feedforward neural network clustering method 142

Figure 5-9: An example of the real semi-supervised feedforward neural network clustering method 143

Figure 5-10: The design of the dynamic unsupervised feedforward neural network clustering method. 144

Figure 5-11: The algorithm of the dynamic unsupervised feedforward neural network clustering 147

Figure 5-12: An example for clustering the data nodes to three clusters by the DUFFNN 154

Figure 5-13: The outlook of clustering the online Iris data to three clusters based on a unique total threshold of each data point by the DUFFNN 155

Figure 5-14: The design of changed section of the DUFFNN suitable for dynamic semi-supervised feedforward neural network clustering method 156

Figure 5-15: Dynamic semi-supervised feedforward neural network clustering algorithm 157

Figure 6-1: The *SW* vector of the Breast cancer Wisconsin (original) dataset using the RUFFNN method 164

Figure 6-2: The clusters of the RUFFNN method on breast cancer Wisconsin (original) dataset 165

Figure 6-3: The *SW* vector of the Iris dataset using the RUFFNN method 167

Figure 6-4: The clusters of the RUFFNN method on the Iris dataset 168

Figure 6-5: The *SW* vector of the Spambase dataset using the RUFFNN method 170

Figure 6-6: The clusters of the RUFFNN method on the Spambase dataset 171

Figure 6-7: The clusters of the RUFFNN method on the Arcene dataset 173

Figure 6-8: The final *SW* vector of the Yeast dataset using the RUFFNN method 175

Figure 7-1: The final *BMW* vector of the breast cancer Wisconsin (original) using the DUFFNN method 184

Figure 7-2: The clusters of the DUFFNN method on the breast cancer Wisconsin (original) 184

Figure 7-3: Comparison of the clustering density of *CCN %* on the breast cancer Wisconsin data points by the DUFFNN, DSFFNN and some related methods 185

Figure 7-4: The final *BMW* vector of the Iris by the DUFFNN method....................... 187

Figure 7-5: The clusters of the DUFFNN method on the Iris 187

Figure 7-6: Comparison of the clustering density of *CCN %* on the Iris data points by the DUFFNN, DSFFNN and some related methods 188

Figure 7-7: The final *BMW* vector of the Spambase dataset by the DUFFNN method 190

Figure 7-8: The clusters of the DUFFNN method on the Spambase 191

Figure 7-9: Comparison of the clustering density of *CCN %* on the Spambase data points by the DUFFNN, DSFFNN and some related methods 192

Figure 7-10: The final *BMW* vector of the SPECT Heart dataset using the DUFFNN method 194

Figure 7-11: The clusters of the DUFFNN method on the SPECT Heart 195

Figure 7-12: Comparison of the clustering density of *CCN %* on the SPECT Heart data points by the DUFFNN, DSFFNN and some related methods 196

Figure 7-13: The final *BMW* vector of the SPECTF Heart dataset of the DUFFNN method 198

Figure 7-14: The clusters of the DUFFNN method on the SPECTF Heart 199

Figure 7-15: Comparison of the clustering density of *CCN %* on the SPECTF Heart data points by the DUFFNN, DSFFNN and some related methods 200

Figure 7-16: The final *BMW* vector of the MUSK1 dataset by the DUFFNN method 203

Figure 7-17: The clusters of the DUFFNN method on the MUSK1 203

Figure 7-18: Comparison of the clustering density of *CCN %* on the MUSK1 data points by the DUFFNN, DSFFNN and some related methods 204

Figure 7-19: The final *BMW* vector of the MUSK2 using the DUFFNN method 207

Figure 7-20: The clusters of the DUFFNN method on the MUSK2 207

Figure 7-21: Comparison of the clustering density of *CCN %* on the MUSK2 data points by the DUFFNN, DSFFNN and some related methods 208

Figure 7-22: The clusters of the DUFFNN method on the Arcene 210

Figure 7-23: Comparison of the clustering density of *CCN %* on the Arcene data points by the DUFFNN, DSFFNN and some related methods 211

Figure 7-24: The final *BMW* vector of the Yeast dataset by the DUFFNN method 213

Figure 7-25: Comparison of the clustering density of *CCN %* on the Yeast data points by the DUFFNN, DSFFNN and some related methods 214

LIST OF TABLES

Table 2-1: A contingency table of the binary variables...33

Table 2-2: Comparing characters to compute the F-measure 41

Table 2-3: Computing *F-measure* of clustering of a dataset with three classes 42

Table 2-4: The time complexity and memory complexity of the main unsupervised clustering methods 58

Table 3-1: The time complexities and memory complexities of some UFFNN clustering and related clustering methods...95

Table 3-2: Some bold advantages of current online dynamic unsupervised feedforward neural network clustering models 106

Table 4-1: The system information of model implementation 113

Table 4-2: The features of the breast cancer Wisconsin dataset 114

Table 4-3: The features of the Iris dataset...115

Table 4-4: The features of the Spambase dataset...116

Table 4-5: The features of SPECT Heart dataset...117

Table 4-6: The features of SPECTF Heart dataset...118

Table 4-7: The features of the Musk1 dataset...119

Table 4-8: The features of the Musk2 dataset...120

Table 4-9: The features of the Arcene dataset 121

Table 4-10: The features of the Yeast dataset...122

Table 4-11: The observed data of breast cancer based on the interval of survival time 123

Table 5-1: The code book of the weight vectors and the standard weight vector 135

Table 5-2: The online input data *X*...149

Table 5-3: The code book of the weight vectors and the *BMW* vector 151

Table 5-4: Comparison of time complexity and memory complexity of the proposed clustering methods with some related methods 159

Table 6-1: The clustering results of the RUFFNN method the breast cancer Wisconsin (Original) dataset 163

Table 6-2: The *SW* vector of the breast cancer Wisconsin (original) dataset using the RUFFNN method 164

Table 6-3: Comparison of the clustering results on the breast cancer Wisconsin dataset by the RUFFNN, RSFFNN and some related methods 165

Table 6-4: The clustering results of the RUFFNN method on the Iris dataset 167

Table 6-5: The *SW* vector of the Iris dataset using the RUFFNN method 167

Table 6-6: Comparison of the clustering results on the Iris dataset by the RUFFNN, RSFFNN and some related methods 168

Table 6-7: The clustering results of the RUFFNN method on the Spambase dataset .. 169

Table 6-8: The *SW* vector of the Spambase dataset by the RUFFNN method 170

Table 6-9: Comparison of the clustering results on the Spambase dataset by the RUFFNN, RSFFNN and some related methods 171

Table 6-10: The clustering results of the RUFFNN method on the Arcene dataset 172

Table 6-11: The *SW* vector of the Arcene dataset by the RUFFNN method 173

Table 6-12: Comparison of the clustering results on the Arcene dataset by the RUFFNN, RSFFNN and some related methods 174

Table 6-13: The *SW* vector of the Yeast dataset by the RUFFNN method 175

Table 6-14: The clusters of the RUFFNN method on the Yeast dataset 176

Table 6-15 : Comparison of the clustering results on the Yeast dataset by the RUFFNN, RSFFNN and some related methods 176

Table 6-16: The RSFFNN results for each subset of the UMMC breast cancer data... 178

Table 6-17: Comparison of the results of the PCA-BPN, the SOM-BPN and the RSFFNN for each subset of the UMMC breast cancer dataset 179

Table 7-1: The clustering result of the DUFFNN method on the breast cancer Wisconsin 183

Table 7-2: The final *BMW* vector of the breast cancer Wisconsin (original) using the DUFFNN method........ 183

Table 7-3: Comparison of the clustering results on the breast cancer Wisconsin data points by the DUFFNN, DSFFNN and some related methods........ 185

Table 7-4: The clustering result of the DUFFNN method on the Iris........ 186

Table 7-5: The final *BMW* vector of the Iris using the DUFFNN method........ 187

Table 7-6: Comparison of the clustering results on the Iris data points by the DUFFNN, DSFFNN and some related methods........ 188

Table 7-7: The clustering result of the DUFFNN method on the Spambase........ 189

Table 7-8: The final computed *BMW* vector from the received Spambase data using the DUFFNN method........ 190

Table 7-9: Comparison the clustering results on the Spambase data points by the DUFFNN, DSFFNN and some related methods........ 191

Table 7-10: The clustering result of the DUFFNN method on the SPECT Heart........ 193

Table 7-11: The final computed *BMW* vector from the received SPECT Heart data using the DUFFNN method........ 194

Table 7-12: Comparison of the clustering results on the SPECT Heart data points by the DUFFNN, DSFFNN and some related methods........ 195

Table 7-13: The clustering result of the DUFFNN method on the SPECT Heart........ 197

Table 7-14: The final computed *BMW* vector from the received SPECTF Heart data using the DUFFNN method........ 198

Table 7-15: Comparison of the clustering results on the SPECTF Heart data points by the DUFFNN, DSFFNN and some related methods........ 199

Table 7-16: The clustering result of the DUFFNN method on the Musk1........ 201

Table 7-17: The final computed *BMW* vector from the received MUSK1 data using the DUFFNN method........ 202

Table 7-18: Comparison of the clustering results on the MUSK1 data points by the DUFFNN, DSFFNN and some related methods...204

Table 7-19: The clustering result of the DUFFNN method of the Musk2....................205

Table 7-20: The final *BMW* vector of the MUSK2 using the DUFFNN method 206

Table 7-21: Comparison of the clustering results on the MUSK2 data points by the DUFFNN, DSFFNN and some related methods...208

Table 7-22: The clustering result of the DUFFNN method on the Arcene 209

Table 7-23: The *BMW* vector of the Arcene dataset by the DUFFNN method............210

Table 7-24: Comparison of the clustering results on the Arcene data points by the DUFFNN, DSFFNN and some related methods...211

Table 7-25: The final computed *BMW* vector from the received Yeast data using the DUFFNN method...212

Table 7-26: The clusters of the DUFFNN method on the Yeast dataset 213

Table 7-27: Comparison of the clustering results on the Yeast data points by the DUFFNN, DSFFNN and some related methods...214

Table 7-28: The results of implementation of the DSFFNN for each subset of the breast cancer from UMMC...216

Table 7-29: Comparison of the results of the PCA-BPN, the SOM-BPN and the DSFFNN for each subset of the UMMC breast cancer dataset 216

Table 8-1: Comparison of the DUFFNN clustering method with some current online dynamic unsupervised feedforward neural network clustering methods......................222

LIST OF SYMBOLS AND ABBREVIATIONS

ADALIN	:	Adaptive linear neuron
AGNES	:	Agglomerative nesting
AI	:	Artificial intelligence
ANN	:	Artificial neural network
ARI	:	Adjusted rand index
BMU	:	Best matching unit
BMW	:	Best matching weight
BPN	:	Back propagation network
CCN	:	Correctly classified nodes
CPU	:	Central processing unit
DBSCAN	:	Density based special clustering of applications with noise
DIANA	:	Divisive analysis
DM	:	Data mining
DPT	:	Delta pre-training
DSFFNN	:	Dynamic semi-supervised feedforward neural network
DSFFNN	:	Dynamic semi-supervised feedforward neural network
DSOM	:	Dynamic self-organizing map
DUFFNN	:	Dynamic unsupervised feedforward neural network
EII	:	Essential important information
ESOINN	:	Enhanced self-organizing incremental neural network
ESOM	:	Evolving self-organizing map
EVMS	:	Eastern virginia medical school
FFNN	:	Feedforward neural network
FM	:	Folkes and Mallow

GNG	:	Growing neural gas
IGNG	:	Incremental growing with neural gas
IGNGU	:	Incremental growing with neural gas utility parameter
KDD	:	Knowledge discovery in databases
K-NN	:	K nearest neighbour
LMS	:	Least mean square
MADALINE	:	Multiple ADALINE
MLP	:	Multilayer perceptrons
NCI	:	National cancer institute
NIPS	:	Neural information processing systems
NMI	:	Normalized mutual information
ODUFFNN	:	Online dynamic unsupervised feedforward neural network
OPTICS	:	Ordering points to identify the clustering structure
PAM	:	Partitioning around medoids
PCA	:	Principal component analysis
PCA	:	Principal component analysis
RBM	:	Restricted boltzmann machines
RN	:	Recurrent networks
RSFFNN	:	Real semi-supervised feedforward neural network
RUFFNN	:	Real unsupervised feedforward neural network
SCAWI	:	Statistically controlled activation weight initialization
Semi-ESOM	:	Semi-supervised evolving self-organizing map
SLP	:	Single-layer perceptrons
SND	:	Standard normal distribution
SOINN	:	Self-organizing incremental neural network
SOM	:	Self-organizing map

SSE	:	Sum of squared error
STING	:	Statistical information grid
SW	:	Standard weight
TLU	:	Threshold logic unit
UCI	:	University of California at Irvine
UFFNN	:	Unsupervised feedforward neural network
UMMC	:	University of Malaya Medical centre
VLS	:	Very large scale integrated
VQ	:	Vector quantization
WLA	:	Weight linear analysis

LIST OF APPENDICES

APPENDIX A: Algorithms 245

APPENDICES B:Datasets Features 260

CHAPTER 1: INTRODUCTION

1.1 Background and Motivation

An artificial neural network (ANN) has its roots in mathematics, statistics, numerical analysis, biology and psychology, and is an artificial representation of the human brain that has the capability to simulate its learning process. An ANN is one of the numerous algorithms used in machine learning and data mining. Neural networks are flexible algorithms that allow users to encode nonlinear relationships between input and the desirable outputs (Dasarathy, 1990; Goebel & Gruenwald, 1999; Hegland, 2003; Kantardzic, 2011; Kemp, et al., 1997). An ANN is suitable to be applied for special applications such as pattern recognition and data classification through a learning process (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Bose & Liang, 1996).

Learning is an imperative property of the neural network. There are many types of learning rules used in the neural networks, which falls under the broad category of supervised learning, unsupervised learning, and reinforcement learning. Most applications to unsupervised learning in machine learning are statistical modelling, compression, filtering, and clustering (Andonie & Kovalerchuk, 2007; Han & Kamber, 2011; Hegland, 2003; Kantardzic, 2011).

In this study, the clustering aspect of unsupervised neural network learning is considered. Learning from observations with unlabeled data in an unsupervised neural network clustering is more desirable and affordable than learning by examples in supervised neural network classification, because of the costly preparation of the training set, time consuming and necessary efforts of human observers. However, the desired output of unsupervised learning is not presented to the unsupervised neural network, therefore, to assess the performance of unsupervised learning, there is no error or reward signal (Demuth, et al., 2008; Kohonen, 1997; Van der Maaten, et al., 2009). Clustering aims to mine useful information hidden among the multivariate data, discover similar groups, and identify meaningful distributions and prototypes in datasets (Deng & Kasabov, 2003; Rougier & Boniface, 2011). When using an unsupervised neural network for clustering, data is divided into meaningful groups with special goals, with related data classified as having higher similarities within groups and unrelated data as dissimilarities between groups (Hegland, 2003).

A feedforward neural network is a popular tool for statistical decision making and in this network, data processing has only one forward direction from the input layer to the output layer without any backward loop (Andonie & Kovalerchuk, 2007; Bose & Liang, 1996; McCloskey, 2000).

1.1.1 Fundamental Unsupervised Feedforward Neural Network Clustering

The unsupervised feedforward neural network (UFFNN) clustering has great capabilities, such as, the inherent distributed parallel processing architectures and the ability to adjust the interconnecting weights to learn and divide data into meaningful groups. Unsupervised neural network clustering methods classifies related data into similar groups without using any class label, and additionally controls noisy data and learns types of input data values based on their weights and properties (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Hegland, 2003; Jain, 2010; Rougier & Boniface, 2011).

The UFFNN clustering methods often use Hebbian learning, or competitive learning, or competitive Hebbian learning (Chakraborty, 2010; Hebooul, et al., 2015; Liu, et al., 2013; Martinetz, 1993). The similarities between Hebbian learning and competitive learning include unsupervised learning without error signal, and is strongly associated with biological systems. However, in competitive learning, only one output must be active; such that, only the weights of the winner are updated in each epoch. By contrast, no constraint is enforced by neighbouring nodes in Hebbian learning, and all weights are updated at each epoch. In the case of competitive Hebbian learning, the neural network method shares some properties of both competitive learning and Hebbian learning (Fritzke, 1997; Hebooul, et al., 2015; Liu, et al., 2013; Martinetz, Berkovich, & Schulten, 1993; McClelland, et al., 1999). Competitive learning can apply vector quantization (VQ) (Linde, et al., 1980) during clustering. Quantization is the process of mapping a large dataset to a smaller set. Linde et al. (Linde, et al., 1980) introduced an algorithm for the VQ design to obtain a suitable code book of weights for input data nodes clustering. VQ is based on the probability density functions through the distribution of the vector of the weights. VQ divides a large set of the data (vectors) into clusters, each of which is represented by its centroid node, as in *K-means* (MacQueen, 1967) which is a partitioning clustering method and some other clustering algorithms. Kohonen's self-organizing map (SOM) (Kohonen, 1982) maps multi-dimensional data onto lower dimensional subspaces, with the geometric relationships between points indicating their similarity. SOM generates subspaces with unsupervised learning neural network training through a competitive learning algorithm. The weights are adjusted based on their proximity to the "winning" nodes, that is, the nodes that most closely resembles an input instance (Germano, 1999; Honkela, 1998; Kohonen, 2000; Ultsch & Siemon, 1990). The existence of incomplete and noisy data affect the accuracy of clustering (Germano, 1999; Honkela, 1998; Kohonen, 2000). The weight vectors are dependent on the data that can significantly cluster and recognize inputs, but the initialization of the weights is often randomly (Germano, 1999; Honkela, 1998; Kohonen, 2000). The weights have to be updated in each epoch during learning and results in slow training time, especially for training of a large data. Also, the model cannot converge properly, through applying unsuitable weights. Therefore, using random or unsuitable weights affects accuracy and memory usage (Andonie & Kovalerchuk, 2007; Demuth, et al., 2008; Han & Kamber, 2011; Jain, 2010). The SOM creates fix code book of the weights. Therefore, the SOM clustering methods is not suitable for lifelong incremental learning (Kulkarni & Mulay, 2013; Wang, et al., 2013). The growing neural gas (GNG) (Fritzke, 1995) method is an example which uses the competitive Hebbian learning, in which the connection between the winner node and the second nearest node is created or updated in each training cycle. The GNG method can follow dynamic distributions by adding nodes and deleting them in the network during clustering by using the utility parameters. The disadvantages of the GNG include the increase in the number of nodes to obtain the input probability density and the requirement for predetermining the maximum number of nodes and thresholds which affect the accuracy of clustering, training time and memory usage (Furao, et al., 2007; Hamker, 2001; Hebboul, et al., 2011). The VQ, *K*-means and some UFFNN clustering methods, such as the SOM and GNG are suitable for stationary information environments and batch datasets in which the data does not change. The static UFFNN clustering methods are generally considered as the fundamental clustering methods and are adapted/modified to be used in non-stationary environments, and forms the current online dynamic UFFNN (ODUFFNN) clustering methods (Bouchachia, et al., 2007; Hebboul, et al., 2011; Kasabov, 1998; Schaal & Atkeson, 1998).

1.1.2 An Overview of Current Online Dynamic Unsupervised Feedforward Neural Network Clustering

Many real world environments, such as credit card transactions, intelligent multi-agent systems, and medical informatics, use online continuous data that are updated frequently (Bouchachia, et al., 2007; Hebboul, et al., 2011; Hsu, 2003; Kasabov, 1998; Rougier & Boniface, 2011). For example, we consider the credit card transaction records environment with 300,000 consumers and several hundred attributes. The problems of such environments are collection, storage, search, transfer, visualization and analysis of large volume of the data including data noise and the high dimensions of the data (Bouchachia, et al., 2007; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Schaal & Atkeson, 1998). In this research, the Arcene dataset was used as a benchmark, and as an example of high dimensional medical data with 10,000 attributes. The Arcene dataset was collected from two different sources: the national cancer institute (NCI) and the eastern Virginia medical school (EVMS) (Asuncion & Newman, 2007). Arcene's task is to distinguish cancer versus normal patterns from mass-spectrometric data (Asuncion & Newman, 2007). This dataset is one of 5 datasets of the neural information processing systems (NIPS) 2003 feature selection challenge. As such, most current publications in this area are on the selection of the best attributes so as to reduce the dimension of the Arcene dataset to get better clustering accuracy, to reduce the central processing unit (CPU) time usage and memory usage (Guyon, 2003; Guyon & Elisseeff, 2003). Data mining is the process of knowledge discovery in databases (KDD), and the knowledge discovery is a data analysis and extraction of the knowledge and potentially useful information based on the relationships between data values in large rational databases, and discovering meaningful patterns and rules from large quantities of data (Brachman & Anand, 1994). Consequently, clustering methods in data mining are used to recognize patterns and discover the knowledge of the data, in order to group them meaningfully (Bouchachia, et al., 2007). As the data environment is an online non-stationary, the online dynamic UFFNN (ODUFFNN) clustering methods should support online incremental learning. In these environments, the data are often highly massive, continuous and uninterrupted input data, the distributions of data are not known and the data distribution may change over time. Incremental learning refers to the ability of repeatedly training a network using new data or deleting unnecessary data, without destroying outdated prototype patterns. (Bouchachia, et al., 2007; Heboui, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Schaal & Atkeson, 1998). The ODUFFNN clustering method should be able to control noisy data, adapt its algorithm and adjust itself in a flexible way to new conditions of the environment over time dynamically for processing of both data and knowledge. The ODUFFNN should accommodate and prune the data and rules without destroying old knowledge, should learn a number of clusters and density of each cluster without predetermining the rules. In addition, the ODUFFNN method must control time, memory space and accuracy efficiently (Bouchachia, et al., 2007; Heboui, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Schaal & Atkeson, 1998).

Existing ODUFFNN clustering methods often use competitive learning as used in the dynamic self-organizing map (DSOM) (Rougier & Boniface, 2011), or competitive Hebbian learning as used in the evolving self-organizing map (ESOM) (Deng & Kasabov, 2003). Furao and Hasegawa introduced enhanced self-organizing incremental neural network (ESOINN) (Furao, et al., 2007) based on the GNG method (Furao & Hasegawa, 2006). The ESOINN method has one layer, and in this model, it is necessary that very old learning information is forgotten. The ESOINN model finds the winner and the second winner of the input vector, and then if it is obvious to create a connection between them or to remove the connection. The density, the weight of the winner and the subclass label of nodes will be updated in each epoch and the noise nodes based on the input values will be deleted. After learning, all nodes will be classified into different classes. However, it cannot solve the main problems of clustering. The disadvantages of the ESOINN method are: out of date learning information is forgotten and new learned patterns are lost. Therefore the topological structure of the incremental online data cannot be well represented. Furthermore, the initialization of the parameters for training is based on trial and error; and there is relearning in several epochs (Hebboul, et al., 2011; Rougier & Boniface, 2011). Hebboul et al. (2011) proposed incremental growing with neural gas utility parameter (IGNGU) as the online incremental unsupervised clustering, which is based on the structures of the GNG and Hebbian (Hebb, 1949) models, but without any restraint and control on the network structure. The structure of the IGNGU contains two layers of learning. The first layer creates a suitable structure of the clusters of the input data nodes with lower noise data, and computes the threshold. The second layer uses the output of the first layer in parallel and creates the final structure of clusters. However, the IGNGU could not control noise and the density overlapping (Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013; Wang, et al., 2013). Evolving self-organizing map (ESOM) (Deng & Kasabov, 2003) is an ODUFFNN method, that is based on the SOM and GNG methods. ESOM starts without nodes and the network updates itself with online entry, and if necessary, it creates new nodes during one training epoch. Similar to the SOM method, each node has a special weight vector and the strong neighbourhood relation is determined by the distance between connected nodes. If the distance is too big, it creates a weak threshold and the connection can be pruned. ESOM is a method based on a normal distribution and VQ in its own way, and creates normal sub clusters across the data space. ESOM is sensitive to noise nodes, prunes weak connections and isolated nodes based on the Hebbian learning (Hebb, 1949) with the forgetting constant parameter for controlling the growing of the weights. However, the ESOM is unable to control the growth of the number of clusters and the size of the network, and takes a long time to train as well as large memory usage; although, the clustering is carried out in just one epoch (Furao & Hasegawa, 2006; Hebooul, et al., 2015; Liu, et al., 2013; Rougier & Boniface, 2011). Dynamic self-organizing map (DSOM) (Rougier & Boniface, 2011) is similar to the SOM based on competitive learning. In order to update the weights of the neighbouring nodes, time dependency is removed, and the parameter of the elasticity or flexibility is considered which is learned through trial and error. If the parameter of the elasticity is too high, DSOM does not converge; and if it is too low, it may prevent DSOM to occur and is not sensitive to the relation between neighbouring nodes. In the DSOM based on initialized thresholds and other clustering task parameters, if no node is close enough to the input values, other nodes must learn according to their distance to the input value. The enhancing dynamic self-organizing map (EDSOM) (Wang, et al., 2013) is based on the SOM and GNG, and its bold advantage includes pruning the weak connections; also the enhanced incremental growing neural gas (Hi-GNG) (Liu, et al., 2013) is based on GNG and Hebbian, and applies the enhanced Hebbian learning, which makes the method robust to noisy data nodes. Reviewing current ODUFFNN clustering methods shows that they inherit the properties and constructions of the VQ, *K-means* and some fundamental UFFNN clustering methods, such as SOM and GNG. The current ODUFFNN methods also inherit the limitations and problems of the primary UFFNN clustering methods (Bouchachia, et al., 2007; Hebboul, et al., 2011; Kasabov, 1998; Prudent & Ennaji, 2005; Shen, et al., 2011). The ODUFFNN clustering is a valuable subject to research; nevertheless, high training time, low accuracy and high memory usage of clustering are common critical issues in the current ODUFFNN clustering methods.

1.2 Problem Statement

In unsupervised neural network clustering, data is divided into meaningful groups with special goals, with related data classified as higher similarities within groups and unrelated data as dissimilarities between groups without using any class label. Additionally, the unsupervised neural network clustering learns types of input data values based on their weights and properties (Andonie & Kovalerchuk, 2007; Bengio, 2013; Hegland, 2003; Jain, 2010). The UFFNN clustering methods such as SOM and

GNG are able to inhere distributed parallel processing architectures and adjust the interconnection weights to learn (Andonie & Kovalerchuk, 2007; Bengio, 2013; Hegland, 2003; Jain, 2010; Rougier & Boniface, 2011).

In the online non-stationary data environments, the data are often large and continuous, as well, the data distribution and network structure may change over time. In such environments, data noise and high dimensional data create problems during the clustering process (Bouchachia, et al., 2007; Hebboul, et al., 2011; Hsu, 2003; Kasabov, 1998; Rougier & Boniface, 2011). In the real online area, the static UFFNN clustering methods are not suitable, however, they are generally considered as the fundamental clustering methods and are adapted/modified to be used in non-stationary environments, and forms the current online dynamic UFFNN (ODUFFNN) clustering methods such as the ESOM and DSOM (Bouchachia, et al., 2007; Hebboul, et al., 2011; Kasabov, 1998; Schaal & Atkeson, 1998). Nevertheless, current ODUFFNN methods inherit the limitations and problems of the primary UFFNN clustering methods too (Bouchachia, et al., 2007; Furao, et al., 2007; Hebboul, et al., 2011; Kasabov, 1998; Prudent & Ennaji, 2005; Shen, et al., 2011).

The ODUFFNN clustering method should not be rigid and after the entrance of each online data, the ODUFFNN clustering model should be ready to change and update its structure, nodes, and connections. Therefore, the ODUFFNN clustering method should be capable of handling new data, controlling noisy data, accommodating and pruning data and rules incrementally and adjusting itself in a flexible way to new conditions of the environment over time dynamically, for processing of both data and knowledge. Relearning is a critical issue in the ODUFFNN clustering method with lifelong and incremental learning. The ODUFFNN clustering methods should train online data fast without the need for relearning. Also, the ODUFFNN should cluster the continuous data in one pass, because there is no capacity to store the whole information and details of the online new data, previous data and the connections of the data points in consequent steps (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). However, the current ODUFFNN clustering methods are not able to accommodate and adjust the data and rules without destroying old data and old knowledge (Bouchachia, et al., 2007; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). In addition, the ODUFFNN clustering method must control time, memory space and accuracy efficiently (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). Hence, the current ODUFFNN clustering methods generally suffer from high training time and low accuracy of clustering, besides, high time complexity and high memory complexity of clustering (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). There is a trade-off between training time, clustering accuracy, complexity and memory complexity of the ODUFFNN clustering methods, and there is surprisingly and comparatively very little previous works dealing with them together in one ODUFFNN clustering model (Furao & Hasegawa, 2006; Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013; Prudent & Ennaji, 2005; Rougier & Boniface, 2011). We identified the limitations, problems and some reasons of the problems associated with the current ODUFFNN clustering methods through the literature review, investigations and our experimentations, as summarized in Figure 1-1.

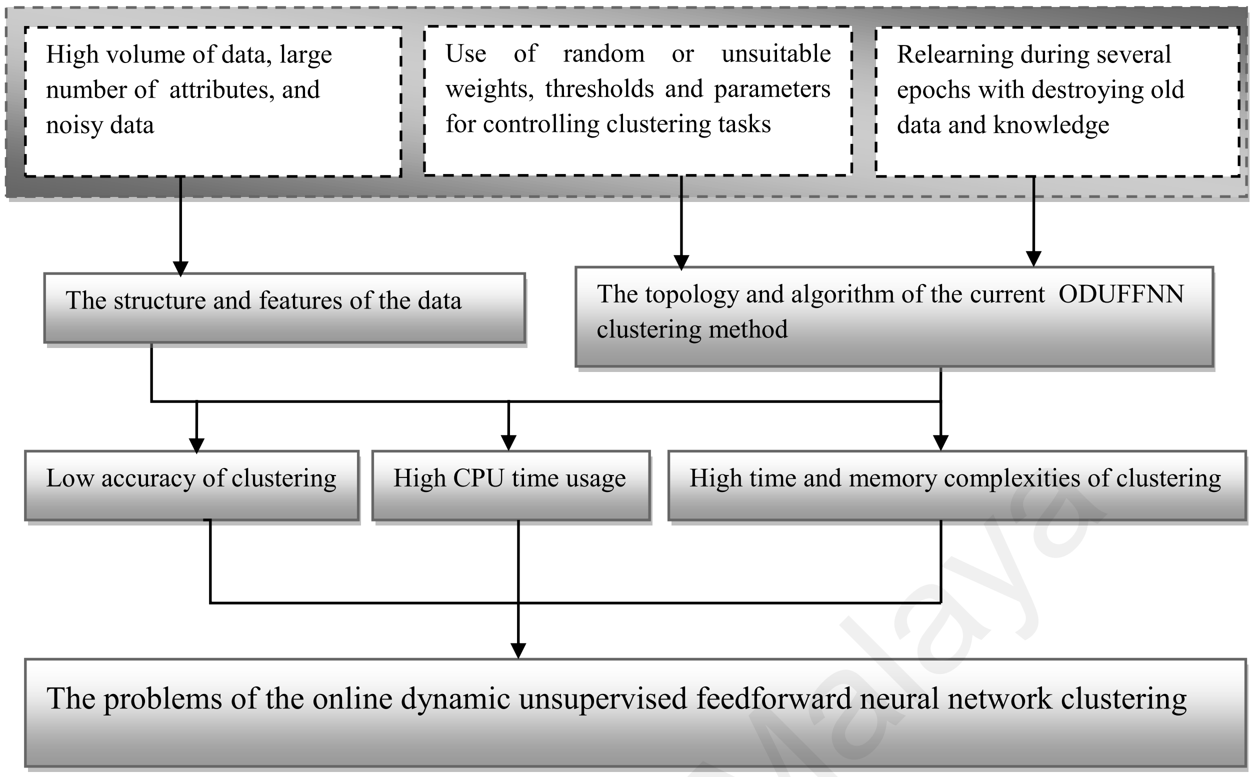

High volume of data, large number of attributes, and noisy data
Use of random or unsuitable weights, thresholds and parameters for controlling clustering tasks
Relearning during several epochs with destroying old data and knowledge
The structure and features of the data
The topology and algorithm of the current ODUFFNN clustering method
Low accuracy of clustering
High CPU time usage
High time and memory complexities of clustering
The problems of the online dynamic unsupervised feedforward neural network clustering

Figure 1-1: The problems of the ODUFFNN clustering methods and two reasons of these problems

Essentially, we recognized the reasons of the problems of the current ODUFFNN clustering methods are related to the structure and features of the data, and the topology and algorithm of the current ODUFFNN clustering method.

1.2.1 Some Reasons of the Problems Related to the Structure and Features of the Data

In real non-stationary data environments, the dynamic clustering models use online continuous data that are updated frequently, the number of received online data grows continuously and the conditions of both data and knowledge change over time. However, the space and memory of systems for the ODUFFNN clustering process are limited (Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Rougier & Boniface, 2011). In order to limit the memory usage and obtain higher speed of clustering, the ODUFFNN clustering models need to reduce the data dimensions. If the number of attributes or the dimensions of the input data vectors are large, the components of the vectors are highly correlated (redundant) (DeMers &

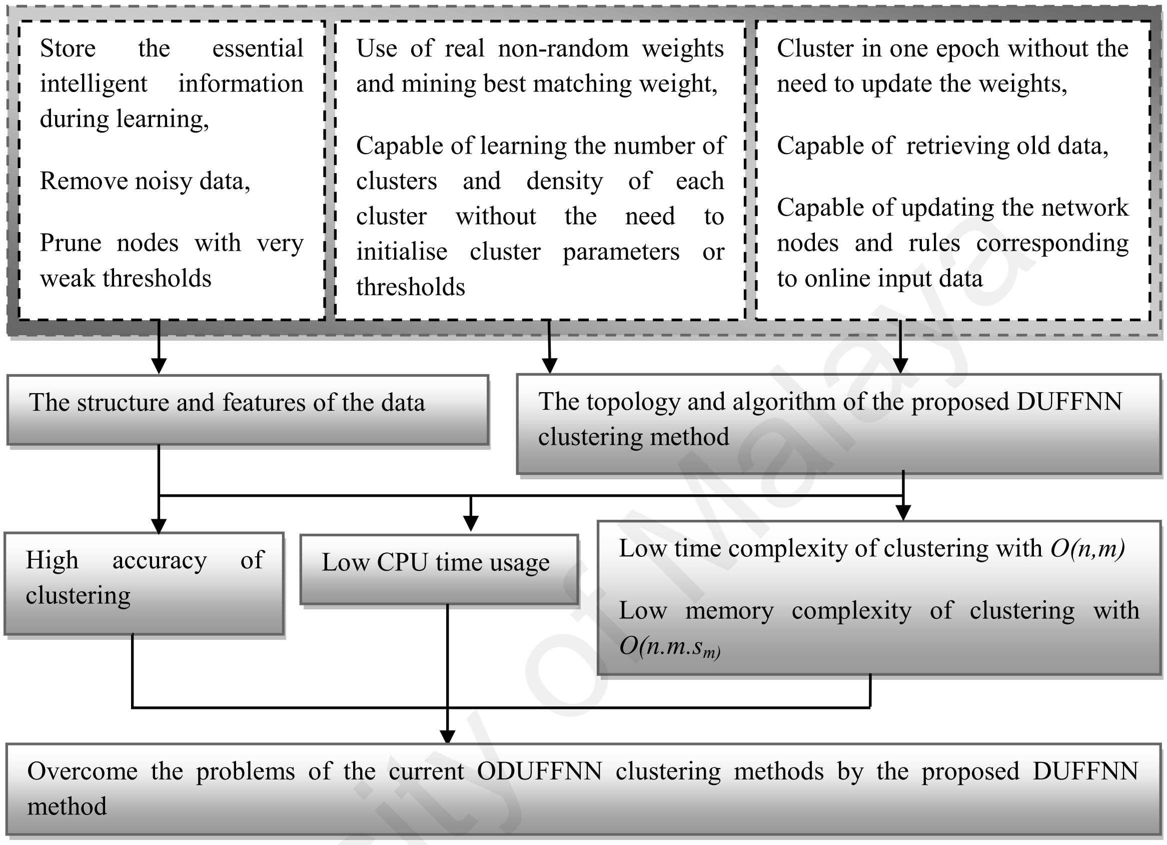
Cottrell, 1993; Furao, et al., 2007; Hebooul, et al., 2015; Liu, et al., 2013; Van der Maaten, et al., 2009). As mentioned in clustering, data is divided into meaningful groups with special goals, with related data classified as higher similarities within groups and unrelated data as dissimilarities between groups. Therefore, highly correlated data vector and noisy data cause difficulty in the clustering process, recognizing the special property of each attribute and finding its related cluster. However, using dimensional reduction techniques may cause the data loses some important its parts, which affect the accuracy of clustering results (DeMers & Cottrell, 1993; Furao, et al., 2007; Van der Maaten, et al., 2009). An example of data with a large number of attributes is the Arcene dataset with 10,000 attributes, from the UCI machine learning repository (Asuncion & Newman, 2007). This dataset is one of five datasets of the NIPS 2003 feature selection challenge. Most current existing papers try to select the best attributes in order to reduce the dimension of the arcane dataset with better accuracy, CPU time usage and memory usage (Guyon, 2003; Guyon & Elisseeff, 2003). Hence, ODUFFNN models should control and manage the distributions of data, the structure of the network of data nodes and their connections. As mentioned, the incremental ODUFFNN clustering methods should check all nodes as a neighbourhood or special cluster for inserting or updating the nodes of the network through the entrance of each online input data (Deng & Kasabov, 2003; Hinton & Salakhutdinov, 2006; Kasabov, 1998; Kohonen, 2000; Van der Maaten, et al., 2009), which affect CPU time usage, accuracy, time complexity and memory complexity of the ODUFFNN clustering methods (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). For example, the ESOM is unable to control the number of clusters and the size of the network (Furao & Hasegawa, 2006; Hebooul, et al., 2015; Liu, et al., 2013; Rougier & Boniface, 2011).

1.2.2 Some Reasons of the Problems Related to the Topology and Algorithm of the Current ODUFFNN Clustering Method

The ODUFFNN models have the ability to adjust the interconnection weights to learn and divide online continuous data into meaningful groups by calculating distance thresholds, and classify related data into similar groups without using any class label, and additionally control noisy data and learns types of input data values based on their weights, properties and distance thresholds (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Hegland, 2003; Jain, 2010; Rougier & Boniface, 2011). Therefore, current ODUFFNN clustering methods are sensitive to the initialization of the weights, distance thresholds and parameters for controlling the clustering tasks (Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013). Current ODUFFNN clustering methods generally use random weights, distance thresholds and parameters for controlling tasks during clustering, such as the, DSOM, that creates the code book of weights by selecting the input data randomly (Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013). Random initialization of weights results in the paradox of low accuracy and high training time (Han & Kamber, 2011; Jain, 2010; Rougier & Boniface, 2011). The clustering process is considerably slow because the weights have to be updated in each epoch during learning. Utilizing suitable weights and parameters is necessary because the neural network relies on the "garbage-in, garbage-out" principle. Therefore, the problem also affects memory usage (Andonie & Kovalerchuk, 2007; Demuth, et al., 2008; Jolliffe, 2002; Kantardzic, 2011; Kasabov, 1998). The parameter values are often selected by trial and error after several executions of the clustering model, and the clustering method often uses many parameters to manage clustering performance (Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013). The ESOM is an example which applies a parameter set in order to measure the distance threshold, control the spread of the neighbourhood, learning rate, steps of learning time and forgetting constant (Deng & Kasabov, 2003). Generally, the ODUFFNN clustering methods such as the ESOM initialize the parameters for training based on trial and error, and after several performances of the clustering model and checking the results, the best parameters are recognized. Therefore, based on our experience and reports of authors, the models are not suitable and have different results for each performance (Furao & Hasegawa, 2006; Hebooul, et al., 2015; Liu, et al., 2013; Rougier & Boniface, 2011).

Relearning is another critical issue in the ODUFFNN clustering method with lifelong and incremental learning. The ODUFFNN clustering methods should train online data quickly without relearning and cluster the continuous data in one pass, because there is no capacity to store all the details of the online data, previous data, and the connection of the data points in subsequent steps (Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Rougier & Boniface, 2011). The current ODUFFNN clustering methods are not able to accommodate and adjust the data and rules without destroying old data and old knowledge (Hebboul, et al., 2011; Jain, 2010; Jain, et al., 1999; Pavel, 2002). Relearning over several epochs takes time and clustering is considerably slow. Relearning affects the clustering accuracy, as well as time complexity and memory complexity for clustering. In the case of relearning, the topological structure of the incremental data cannot be represented well, the number of clusters and density of each cluster are not clear, and cannot be easily learnt. Therefore, relearning affects the clustering accuracy, time complexity and memory complexity of clustering (Bouchachia, et al., 2007; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Pavel, 2002).

In order to overcome the problems, we developed a dynamic UFFNN (DUFFNN) clustering model with some solutions and strategies such as one epoch training, as shown in Figure 1-2.



Store the essential intelligent information during learning,
Remove noisy data,
Prune nodes with very weak thresholds
Use of real non-random weights and mining best matching weight,
Capable of learning the number of clusters and density of each cluster without the need to initialise cluster parameters or thresholds
Cluster in one epoch without the need to update the weights,
Capable of retrieving old data,
Capable of updating the network nodes and rules corresponding to online input data
The structure and features of the data
The topology and algorithm of the proposed DUFFNN clustering method
High accuracy of clustering
Low CPU time usage
Low time complexity of clustering with O(n,m)
Low memory complexity of clustering with O(n.m.sm)
Overcome the problems of the current ODUFFNN clustering methods by the proposed DUFFNN method

Figure 1-2: The solutions for the mentioned problems

Dynamically after each entrance of the online data, the DUFFNN learns and stores important information about the current online data, such as the non-random weights and completes a code book of the weights. Then, a standard weight vector is extracted and updated from the code book. For example, in order to cluster the Iris dataset as a standard dataset from the UCI Repository with 4 attributes and 150 instances (Asuncion & Newman, 2007), the model randomly selects one instance (data record) from the dataset for simulation of the online continuous input data. Immediately, the ODUFFNN model calculates the non-random weight vector of the data and the final standard weight vector of the Iris data will be (0.16, 0.33, 0.23, 0.28), after learning the last input data. Consequently, a single layer DUFFNN calculates the exclusive distance threshold of each data based on the standard weight vector. The input data are clustered based on the exclusive distance thresholds. The calculated weight vector of the Iris shows that the weight of the first attribute is lower and weaker than the others and has least affect on the calculation of the related distance threshold. If the weight is very close to zero, the model is able to reduce the related attribute values. Also, the model is able to recognize the outlier input data as noisy data through its solitary distance threshold. The threshold of each data point is stored in the memory as important information too.

Based on the literature, after learning in order to improve the quality of the DUFFNN clustering result, the model assigns a class label to the input data through the training data (Deng & Kasabov, 2003; Hebooul, et al., 2015; Kamiya, et al., 2007; Prudent & Ennaji, 2005; Shen, et al., 2011). The class label of each initially unlabeled input data is predicted by considering a linear activation function and the exclusive distance threshold. Finally, the number of clusters and the density of each cluster are updated

1.3 Objectives of Research

In order to overcome the problems as mentioned in Section 1.2, the objectives of this research are as follows:

- To review current effective ODUFFNN clustering methods.
- To identify limitations and problems of current effective ODUFFNN clustering methods through the literature and practical investigations.
- To develop a dynamic unsupervised feedforward neural network (DUFFNN) clustering method that is able to:

1) Reduce the training time of clustering during one training epoch
2) Increase the accuracy of clustering
3) Reduce the time complexity of clustering

4) Reduce the memory complexity of clustering

- To evaluate the performance of the proposed DUFFNN method.
- To compare the results of the proposed DUFFNN method performance with rival methods within the scope of this research.

1.4 Preliminary Research Questions

The research essays the following preliminary questions corresponding to the research objectives identified in Section 1.3.

Objectives 1 and 2: To review current effective ODUFFNN clustering methods; to identify limitations and problems of current effective ODUFFNN clustering methods through the literature and practical investigations.

- **Q1:** What are the strategies, topologies and performances of current effective ODUFFNN clustering methods?
- **Q2:** What are the limitations and problems of the current ODUFFNN clustering methods?

Objective 3: To develop a dynamic unsupervised feedforward neural network (DUFFNN) clustering method that is able to:

1) Reduce the training time of clustering during one training epoch
2) Increase the accuracy of clustering
3) Reduce the time complexity of clustering
4) Reduce the memory complexity of clustering

- **Q3:** Is the proposed DUFFNN clustering method appropriate in order to cluster online continuous data dynamically and incrementally?

- **Q4:** How can we develop a DUFFNN clustering method and what is the associated algorithm that is able to reduce the training time of clustering, increase the accuracy of clustering, reduce the time complexity and reduce the memory complexity of clustering?

Objectives 4 and 5: To evaluate the performance of the proposed DUFFNN method; to compare the results of the proposed DUFFNN method performance with rival methods within the scope of this research.

- **Q5:** How is the performance of the developed DUFFNN clustering method in comparison of results with rival methods?

1.5 Scope of Research

This research work focuses on the clustering aspect of unsupervised neural network learning. The focus is on the unsupervised neural network clustering methods, which have a feedforward topology and cluster the data in only one forward interconnection from the input layer to the output layer without any backward loop. Therefore, the proposed method is able to cluster the data without relearning in a short training time.

In order to achieve the objectives of the thesis, we focus on the abilities of reducing the training time of clustering during one epoch, reducing the time complexity and reducing the memory complexity, and increasing the accuracy of clustering.

Also, the work is limited to unsupervised feedforward neural network clustering method with online incremental learning abilities. Since the data environment is an online non-stationary, the online dynamic UFFNN (ODUFFNN) clustering methods should have online incremental learning. In these environments, the data are often high volume, continuous and uninterrupted, the distributions of data is not known and the data distribution may change over time. Incremental learning refers to the ability of repeatedly training a network using new data or deleting unnecessary data, without destroying outdated prototype patterns (see Section 1.1).

For experimental results in this research, non-stationary continuous data is simulated by picking an instance data from the dataset randomly and uniformly at each training process of clustering (Rougier & Boniface, 2011) (see Section 4.2.6).

All practical experiments are carried out under the domain of nine datasets with low dimension such as Iris dataset and with a large number of dimensions such as Arcene dataset from the University of California at Irvine (UCI) machine learning repository (Asuncion & Newman, 2007), (see Section 4.2.6).

The selected datasets are remarkable, and most conventional methods do not satisfactorily cluster these datasets because of their features (see Section 1.2). The datasets from the UCI Repository of Machine Learning (Asuncion & Newman, 2007) are used for benchmarking purposes of unsupervised feedforward neural network clustering (see Section 4.2.6). This research work focuses on the ODUFFNN clustering methods for selected datasets with different size, number of attributes and number of classes, similar to the selected datasets in this thesis (see Section 4.2.6). However, the scope of the thesis is not on big and high dimensional datasets, and solving the problems associated with them. For example, this research work does not focus on mapping high dimensional data to low dimensional data, such as principal component analysis (PCA) (Jolliffe, 1986), which is considered as a preprocessing technique for dimensional reduction of data (see Section 3.2.6).

The accuracy of the methods is measured through the number of clusters, the number of correctly classified nodes and the *F-measure* function by 10 folds of the test set

(Andrew, 2014; Chaimontree, et al., 2010; Rendón, et al., 2011; Rendón, et al., 2011; Sung & Mukkamala, 2003).

The focus of this research is the DUFFNN clustering, however, based on the literature in order to improve the quality of the clustering results by assigning the class labels after learning (Deng & Kasabov, 2003; Hebooul, et al., 2015; Kamiya, et al., 2007; Prudent & Ennaji, 2005; Shen, et al., 2011), the dynamic semi-supervised feedforward neural network (DSFFNN) clustering is introduced. Also, in order to illustrate the efficiency of the DSFFNN clustering method, in addition to the UCI datasets, the breast cancer dataset from the University of Malaya medical centre (UMMC) is used to predict the survival times of breast cancer patients.

1.6 Research Methodology

This research focuses on a developed dynamic unsupervised feedforward neural network clustering method. In order to overcome the mentioned problems in Section 1.2, the methodology follows the steps of a framework, which is shown in Figure 1-3. The details of the research methodology are provided in Chapter 5. The research methodology has briefly three main processes:

- We review, investigate and analyse current ODUFFNN clustering methods and identify their limitations and problems through the literature, their history, and identify some reasons of these problems through a practical investigation.
- In order to overcome the mentioned problems in Section 1.2 and achieve to our objectives, we develop the DUFFNN clustering method with one epoch of training for clustering non-stationary continuous data with incremental learning ability. For this purpose, we firstly introduce a real unsupervised feedforward neural network (RUFFNN) clustering method suitable for stationary data environment, and improve the real semi-supervised FFNN (RSFFNN) clustering method, in order to further improve

the quality of the result of the RUFFNN method, we assign a class label to each of the unlabeled data by considering a linear activation function and the exclusive distance threshold. Then, we develop the DUFFNN clustering method by adapting the structure and the features of the RUFFNN method, and improve the dynamic semi-supervised FFNN (DSFFNN) clustering method, in order to improve the quality of the result of the DUFFNN method by applying class labels, as illustrated in Chapters 4 and 5 of the thesis.

- To evaluate the performance of the proposes methods and compare their results with similar methods, we use various datasets from the UCI machine learning repository and the breast cancer dataset from UMMC, as illustrated in Chapters 6 and 7 of the thesis.

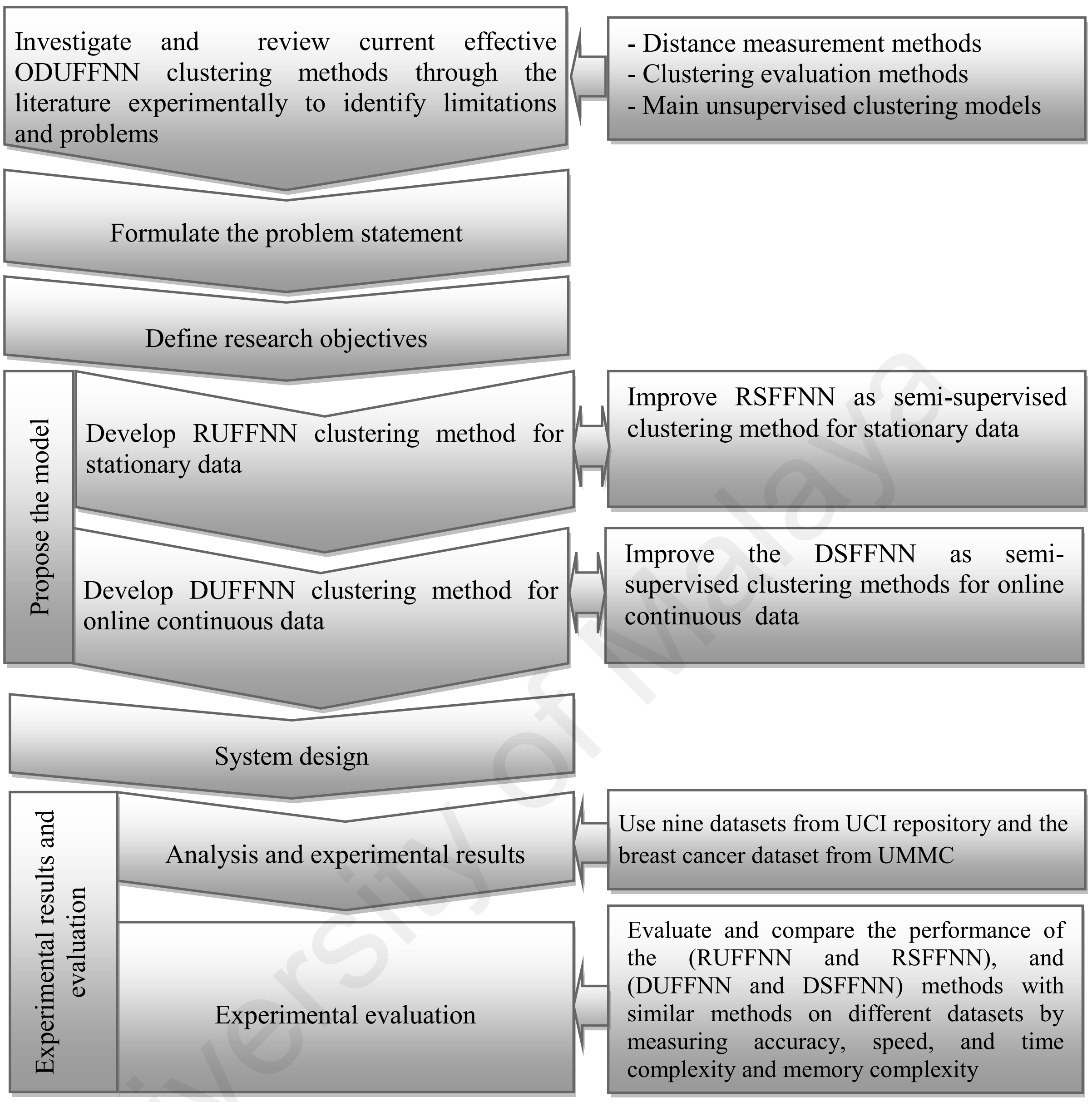

Investigate and review current effective ODUFFNN clustering methods through the literature experimentally to identify limitations and problems
- Distance measurement methods
- Clustering evaluation methods
- Main unsupervised clustering models
Formulate the problem statement
Define research objectives
Propose the model
Develop RUFFNN clustering method for stationary data
Improve RSFFNN as semi-supervised clustering method for stationary data
Develop DUFFNN clustering method for online continuous data
Improve the DSFFNN as semi-supervised clustering methods for online continuous data
System design
Experimental results and evaluation
Analysis and experimental results
Use nine datasets from UCI repository and the breast cancer dataset from UMMC
Experimental evaluation
Evaluate and compare the performance of the (RUFFNN and RSFFNN), and (DUFFNN and DSFFNN) methods with similar methods on different datasets by measuring accuracy, speed, and time complexity and memory complexity

Figure 1-3: The framework of the research methodology

1.7 Overview of the Contributions of Thesis

The contributions are discussed with details in Section 8.4, however, the overall original contributions in several viewpoints are as follows:

- A developed real unsupervised feedforward neural network (RUFFNN) model for single epoch clustering method, which does not require any training cycle, computation of error functions, and updating the weights. The model uses non random weights by using normalized input data. Also, the proposed RUFFNN method is able to control and delete attributes with weak weights to control the dimensions of data, and data with solitary distance thresholds in order to reduce noise. The number of clusters and the density of each cluster are predicted based on the thresholds by the model without any pre-initialization of parameters.
- An improved real semi-supervised feedforward neural network (RSFFNN) clustering method. The RUFFNN clustering is improved by applying class labels as partial supervision, which is entitled, " real semi-supervised FFNN (RSFFNN) clustering", to assign a class label to the each unlabeled data by considering a linear activation function and the exclusive threshold for more accurate results.
- A developed dynamic unsupervised feedforward neural network (DUFFNN) clustering method. The DUFFNN clustering method inherits the structure, features and capabilities of the RUFFNN clustering. The DUFFNN clustering with incremental lifelong or online learning property is developed for real non stationary environments. The DUFFNN clustering is a flexible method, and with each online continuous data, immediately updates all nodes, weights and distance thresholds.
- An improved dynamic semi-supervised feedforward neural network (DSFFNN) clustering method. The DUFFNN clustering is improved by applying class labels as partial supervision, which is entitled, " dynamic semi-supervised FFNN (DSFFNN)

clustering", to assign a class label to each unlabeled data by considering a linear activation function and the exclusive threshold for more accurate results.

The experimental results using various datasets prove that the methods, which have been developed based on the strong support of the theoretical background, are practical and efficient.

1.8 Organization of Thesis

The thesis is organized in accordance with the standard structure of thesis and dissertations at the University of Malaya. The thesis has eight chapters, including this introductory chapter. Chapter one covers the online dynamic unsupervised feedforward neural network (ODUFFNN) clustering that leads to the idea of exploring the details of the concepts of faster and more accurate ODUFFNN clustering with low time complexity and low memory complexity.

Chapter two discusses the basic concepts in unsupervised clustering, which includes the distance function of similarities, clustering evaluation, and the main unsupervised clustering algorithms.

Chapter three covers the literature review on the feedforward neural network, and online dynamic unsupervised feedforward neural network clustering. This chapter introduces the methods in general and the applications, which includes the limitations and problems are discussed in this research.

In Chapter four, the details of the research methodology, in order to achieve the research objectives, are provided based on the framework of the research methodology as shown in Figure 1-3. This chapter includes a brief overview of proposed methods and the evaluation plan. Also, the data requirements for running experiments of the proposed methods are discussed.

Chapter five is the main part of the thesis. In this chapter, the design and algorithms of the developed RUFFNN, RSFFNN, DUFFNN, and DSFFNN are discussed.

In Chapter six and seven, firstly the performances of the proposed methods are experimentally determined, and then experimental evaluation of the proposed methods and comparison of their results with similar methods are discussed by measuring speed, time complexity, memory complexity, and accuracy. Chapter six discusses on experimental results and evaluation on the RUFFNN and RSFFNN. Chapter seven discusses on experimental results and evaluation on the DUFFNN and DSFFNN.

Chapter eight is the last chapter, which concludes the research with the contributions and some recommendations for future work and development.

CHAPTER 2: CONCEPT IN UNSUPERVISED CLUSTERING

2.1 Introduction

Clustering has its roots in different domains, such as data mining, statistics, biology and machine learning. Clustering groups the data into clusters with high similarities between objects within each cluster, but high dissimilarities between objects within different clusters. Dissimilarities are measured based on the values of attributes of the objects. This chapter illustrates the overview of unsupervised clustering, the different types of data used in unsupervised clustering, and related distance measurements based on dissimilarity features between two objects, and the popular clustering validation methods. Then, we review the main unsupervised clustering methods in data mining, such as partitioning hierarchical, density-based, model-based, and grid based clustering.

2.2 Overviews of Unsupervised Clustering

Artificial intelligence (AI) includes learning in virtual environments and rule-based systems (Deconinck, 2010; Nilsson, 1998). AI has the capabilities of self-analysis, learning, adapting, improving through communication with its environment, and conducting intelligent tasks. Kuncheva in 2005 showed the relations of different areas, such as AI, machine learning, data mining, statistics, mathematics and pattern recognition in one graph (Kuncheva, 2005), as shown in Figure 2-1.

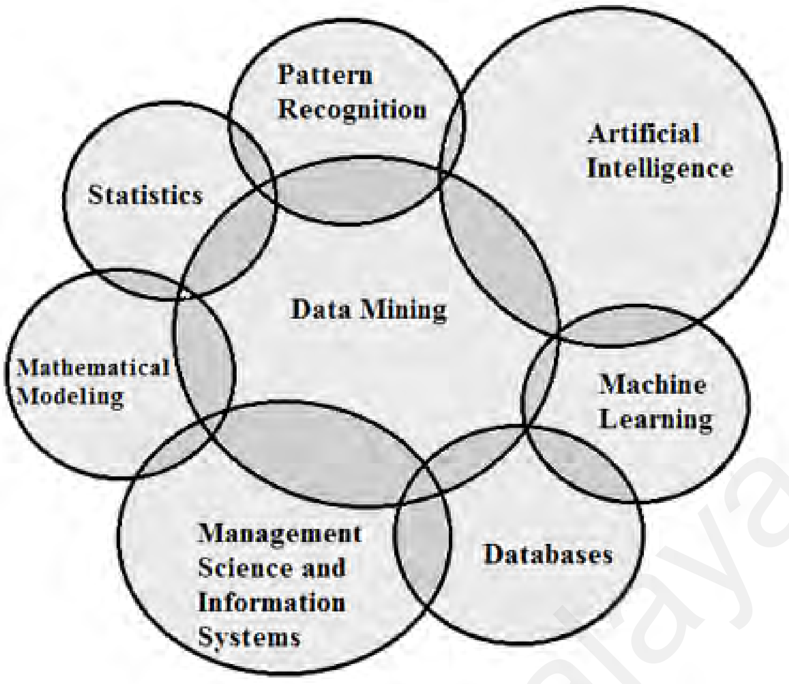


Figure 2-1: The relations of data mining and other fields

Machine learning (Mitchell, 1997) is an advanced approach of learning system in AI, and its main effort is making a machine with capability of learning and accommodating the new information. AI is a multi-interdisciplinary field that is connected to other fields such as computer science, statistics and mathematics. Machine learning is also concerned with the discovery of model, patterns, and regularities in data. In machine learning, the particular sample data and past knowledge are applied and learned to generate models. Learning rules are categorized broadly under supervised learning, unsupervised learning, and reinforcement learning (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Han & Kamber, 2006, 2011; Kantardzic, 2011). Supervised learning determines patterns in the data that illustrates the relation between data attributes and a constraint such as a class label. The supervised model, learns these patterns and predicts the value of the target attribute for future input data. However, to assess the performance of unsupervised learning, there is no target attribute, no error or reward signal. Approaches to unsupervised learning in machine learning are statistical modelling, compression, filtering, blind source separation, and clustering. Unsupervised learning or self-organized learning finds symmetries in the data represented by unlabeled input data. Unsupervised learning, discovers the data to extract some essential structures in the data (Han & Kamber, 2006, 2011; Hebboul, et al., 2011; Kantardzic, 2011). In this research, we consider the clustering aspect of unsupervised learning. Unsupervised clustering learns raw data and divides the data into meaningful groups with special goals and classifies highly similar data into one group without using any class labels. In reinforcement learning, the model is capable of generating certain effects and interactions with the dynamic environment for recognizing an unknown attribute value. At each point in time, the environment generates an observation for this unknown attribute value which is the reward or reinforcement for the model. Consequently, the model can select and use suitable rule and interacts with the environment for more rewards (Lin, Osan, & Tsien, 2006).

Existing methods with the capability of extracting valuable knowledge from large data stores, are necessary because of the growth of the database industry and the advanced resulting market. Data mining (DM) and knowledge discovery in databases (KDD) (Brachman & Anand, 1994; Liu & Ban, 2015) have emrged as a new scientific and engineering discipline. As mentioned earlier, machine learning is an advanced approach of learning system in AI with the capability of learning and accommodating new information in systems; while data mining is the process to solve the problems by analysing data that already exists in databases, and extracts the structured data stored in the data warehouse, in the form of rational data tables. Data mining is the core phase of the process of knowledge discovery in databases (KDD). Knowledge discovery is the process of data analysis and extraction of the knowledge and potentially useful information based on the relationships between data values in large rational databases, including discovery of the meaningful patterns from large quantities of data. The process of the KDD includes obtaining raw data from all sources of data values, preprocessing, cleaning, analysing, and transforming or formatting of data values, and applying data mining algorithms, and finally interpreting and evaluating the results (Brachman & Anand, 1994; Han & Kamber, 2011; Hegland, 2003; Hsu, 2003; Liu & Ban, 2015). Figure 2-2 shows the KDD process, as follows:

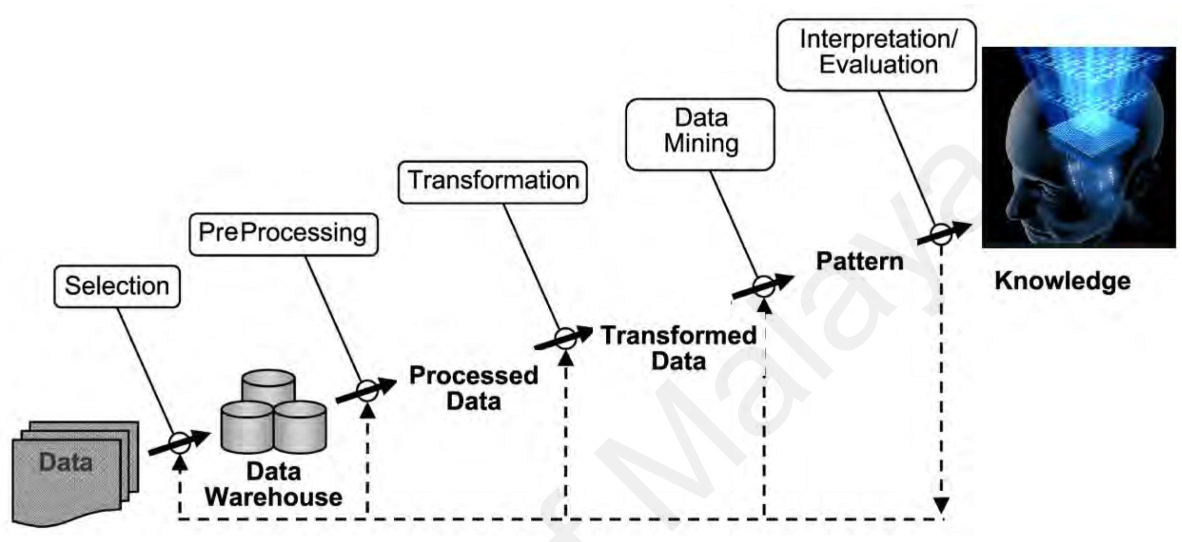


Figure 2-2: The KDD process

In the final step of the KDD process, using some visualization techniques are useful. Visualization of data is the transition of the data into a visual or a two-dimensional format to analyse the type of data, and the relationship between attributes of data (Goebel & Gruenwald, 1999; Granda, 2003). Visualization techniques are dependent on the types of the data mining model, structure, and attributes of the data. Visualization techniques in clustering methods are often the scatter plots, dendrograms, smoothed data histograms, self organizing maps, and proximity matrixes. For example, clusters can be represented by the centroid of each cluster, therefore, the radius of the cluster and standard deviation of the data points in ratio to the related centroid in each cluster shows the limited area of each cluster (Goebel & Gruenwald, 1999; Granda, 2003), as shown in Figure 2-3.

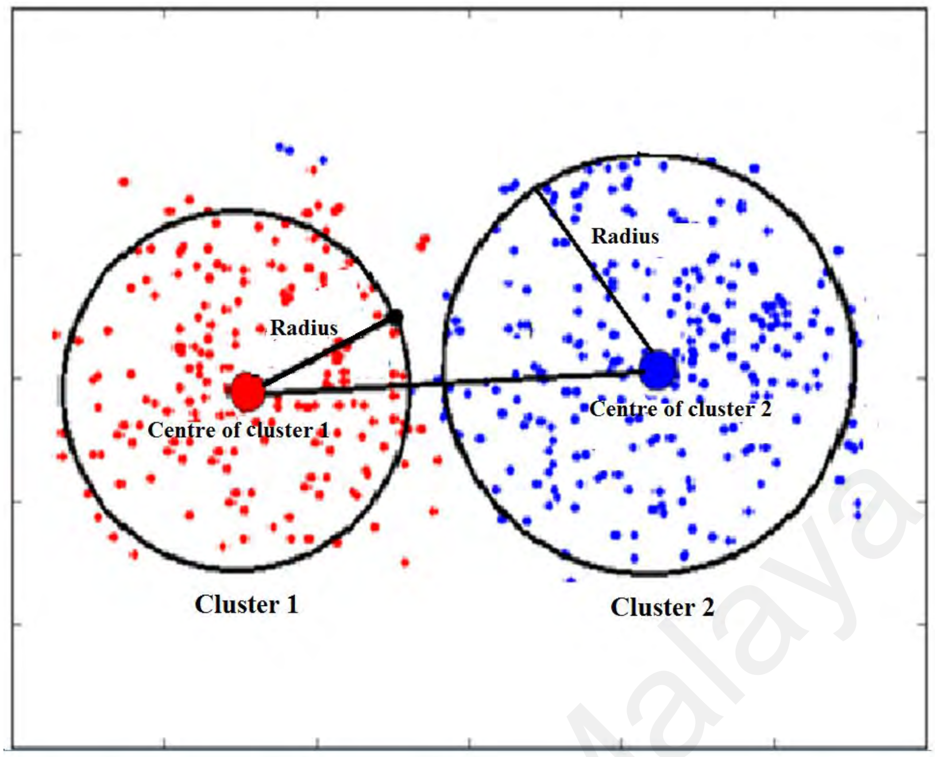


Figure 2-3: An example of two clusters

However, representing the irregular shape clusters is difficult, and using centroid representation is not suitable in general. Figure 2-4 shows an example of irregular shape clusters.

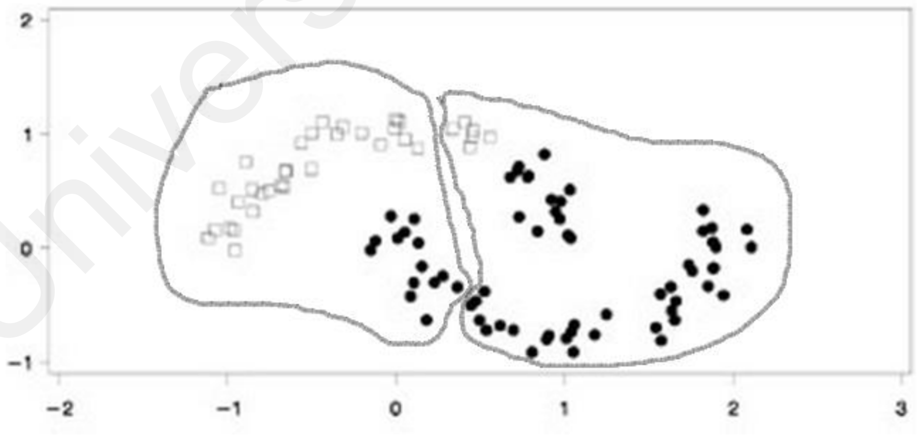


Figure 2-4: An example of irregular shape cluster

2.3 Types of Data in Unsupervised Clustering

Generally, the dataset which is considered for clustering contains n objects, and each object has p attributes. The memory based clustering methods usually consider two structures of the dataset (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011):

- Data matrix (objects by variable structure), has the structure of *n* objects by *p* variables of attributes (or measurements) in the form of relational table, such as *n* persons by *p* attributes of age, height, weight, gender and so on. The numbers of the rows and columns in the matrix are different, therefore, the data matrix is two mode matrix.
- Dissimilarity matrix (objects by objects structure), has the structure of *n* objects by *n* objects for storing a collection of proximities that are available for all pairs of *n* objects. The numbers of the rows and columns in the matrix are the same, therefore, the data matrix is a one mode matrix, where *d(i,j)* points to the measured dissimilarities between objects *i* and *j*. The dissimilarities *d(i,j)* are not negative. If the objects *i* and *j* are far, their *d(i,j)* has a high value and shows less similarity between objects.

Figure 2-5 shows the data matrix and the dissimilarity matrix.

$$\begin{pmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{pmatrix} \qquad \begin{pmatrix} 0 & & & \\ d(2,1) & 0 & & \\ d(3,1) & d(3,2) & & \\ \cdots & \cdots & 0 & \\ d(n,1) & X_{n2} & \cdots & 0 \end{pmatrix}$$

Figure 2-5: The data matrix and the dissimilarity matrix

During the clustering process, the clustering methods often convert the structure of raw data matrix to the dissimilarity matrix by measuring dissimilarities (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

2.4 Distance Functions For Measuring the Dissimilarity of Objects

The *d(i,j)* is a distance function for measuring the dissimilarity between two data objects, and is different for interval-scaled, boolean, categorical, ordinal and ratio variables.

2.4.1 Distance Measuring of Interval-Scaled Variable

Interval-scaled variables are defined as continuous variable quantities on the basis of an approximately linear scale, such as weight and height. There are several techniques to measure the distance of the interval-scaled objects such as the mean $Mean_f$ as shown in Equation (2.1), absolute deviation S_f as shown in Equation (2.2), and standard measurement (Z-score) Z_f as shown in Equation (2.3) (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

Mean, $Mean_f = {}^1\!/_n \left(X_{1f} + X_{2f} + \cdots + X_{nf}\right)$ (2.1)

Absolute deviation, $S_f = {}^1\!/_n \left(|X_{1f} - Mean_f| + |X_{2f} - Mean_f| + \ldots + |X_{nf} - Mean_f|\right)$

(2.2)

Standard measurement (Z-score), $Z_f = {}^{X_{1f} - Mean_f}\!/_{S_f}$ (2.3)

Minkowski distance as shown in Equation (2.4),

$$d(i,j) = {}^1\!/_q \left((X_{i1} - X_{j1})^q + (X_{i2} - X_{j2})^q + \ldots + (X_{ip} - X_{jp})^q\right) \quad (2.4)$$

Where $i = (X_{i1}, X_{i2}, \ldots, X_{ip})$, and $i = (X_{j1}, X_{j2}, \ldots, X_{jp})$ are two p-dimensional data objects, and q is a positive integer.

Manhatan distance, when *q=1* , as shown in Equation (2.5),

$$d(i,j) = |X_{i1} - X_{j1}| + |X_{i2} - X_{j2}| + \dots + |X_{ip} - X_{jp}| \quad (2.5)$$

Euclidean distance, when *q=2*, as shown in Equation (2.6),

$$d(i,j) = {}^{1}/_{2} \left((X_{i1} - X_{j1})^2 + (X_{i2} - X_{j2})^2 + \dots + (X_{ip} - X_{jp})^2 \right) \quad (2.6)$$

Moreover, weighted distances can be generated by using the mentioned formulas in the above Equations, such the weighted Euclidean distance in Equation (2.7),

$$d(i,j) = {}^{1}/_{2} \left(W_1|X_{i1} - X_{j1}|^2 + W_2|X_{i2} - X_{j2}|^2 + \dots + W_m|X_{in} - X_{jn}|^2 \right) \quad (2.7)$$

The mathematical requirements of the above distance functions are as follows:

$d(i,j) \geq 0$: Distance is a nonnegative number;

$d(i,j) = 0$: The distance of an object to itself is zero;

$d(i,j) = d(j,i)$: Distance is a symetric function;

$d(i,j) \leq d(i,h) + d(h,j)$: Triangular inequality;

2.4.2 Distance Measuring Binary Variables

A binary variable has just two states 0 and 1 such as absent and present for two cases of alive or dead. Table 2-1 shows a contingency table of the binary variables (Han & Kamber, 2006, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

Table 2-1: A contingency table of the binary variables

Object *j*

Object *i*

	1	0	*Sum*
1	*q*	*r*	*q+r*
0	*s*	*t*	*s+t*
Sum	*q+s*	*r+t*	*p*

Where q is the number of variables that equal 1 for both objects i and j, r is the number of variables that equal 1 for object i but equal 0 for object j, s is the number of variables that equal 0 for object i but equal 1 for object j, t the number of variables that equal 0 for both objects i and j. The total number of variables is p,where $p=q+r+s+t$. When both of the states of a binary variable have the same weight, the binary variable is symmetric, and it is called symmetric binary dissimilarity. For example, the attribute of gender can be male or female, and it is not important which one is assigned 0 or 1 and they have the same weights. Equation (2.8) is used to the symmetric binary dissimilarity between objects i and j. This is a simple matching coefficient (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

$$d(i,j) = {}^{r+s}/_{q+r+s+t} \tag{2.8}$$

A binary variable is asymmetric if the states of a binary variable have different weights, such as the positive and negative results of a disease test. Equation (2.9) is used to find the asymmetric binary dissimilarity between object i and object j (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

$$d(i,j) = {}^{r+s}/_{q+r+s} \tag{2.9}$$

The distance between two binary variables can be computed based on the similarity instead of dissimilarity. For example, the asymmetric binary dissimilarity between object i and object j, which is *sim(i,j)* can be measured as shown in Equation (2.10). *sim(i,j)* is called the Jaccard coefficient (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

$$sim(i,j) = {}^{q}/_{q+r+s} = 1 - d(i,j) \tag{2.10}$$

2.4.3 Distance Measuring Nominal Variables

A nominal variable is called categorical variable, if it can have more that two states, such as an attribute of colour with five states of blue, red, yellow, green and brown. The dissimilarity between two object i and object j can be measured based on the ratio of dissimilarities, as shown in Equation (2.11) (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

$$d(i,j) = {}^{p-m}/_{p} \tag{2.11}$$

Where m is the number of states of the categorical variable, and p is the total number of variables.

2.4.4 Distance Measuring Ordinal Variables and Interval-Scaled Variables

An ordinal variable is similar to a categorical variable, however, the states of the ordinal value are placed sequentially, such as rank variable. We can consider f as a variable from a set of ordinal variables describing n objects, which has m_f states. For example, professional ranks often enumerated in a sequential order of assistant, associate and full professors, therefore it has three states or $m_f = 3$. X_{if} is the value of the f for the i_{th} object. Each X_{if} can replace by its matching rank, $rank_{if} \epsilon \{1, 2, \dots, m_f\}$. In order to have equal weight, it is necessary to map the range of each variable onto [0.0,1.0] by replacing i_{th} object in the f_{th} variable as shown in Equation (2.12).

$$\mathrm{Z_{if}} = {}^{\mathrm{rank_{if}} - 1}/_{\mathrm{m_f} - 1} \tag{2.12}$$

This technique can be applied to compute the dissimilarity of the interval-scaled variables too (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

2.4.5 Distance Measuring Ratio-Scaled Variables

A ratio-scaled variable is a positive quantity on a nonlinear scale. An example for a nonlinear scale is an exponential scale that approximately follows the formula of Ae^{Bt} or Ae^{-Bt} , where A and B are positive constants, and t is time. The bacteria population growth and the radioactive element decay are examples for the ration-scaled variable. There are three methods to adjust the dissimilarity between objects (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011), as follow:

1) Treat ratio-scaled variables, similar interval-scaled variables, and follow Equation (2.12).

2) Apply logarithmic transformation to a ratio-scaled variable, as shown in Equation (2.13).

$$Y_{if} = log(X_{if}) \qquad (2.13)$$

3) Treat X_{if} as continuous ordinal data and treat their rank as interval-scaled, and follow Equation (2.12).

2.4.6 Distance Measuring of Vector Object

There are several techniques to measure a similarity function for *s(x,y)* , to compare two vector *X* and vector *Y*. One accepted technique is to measure similarity function as a cosine measure, as shown in Equation (2.14).

$$s(x,y) = X^t \times Y / \|X\|\|Y\| \qquad (2.14\ )$$

When X^t is a transposition of vector *X,* $\|X\|$ is the Euclidean norm of vector X^t , $\|Y\|$ is the Euclidean norm of vector *Y*, and s is fundamentally the cosine of the angle between vector *X* and the general linear transformation (Han & Kamber, 2011; Hebboul, et al., 2011; Kantardzic, 2011).

2.5 Clustering Evaluation

Evaluation is a critical issue to identify the clustering tendency of the dataset with non-random structure and the correct number of clusters. Clustering evaluation is also important in order to recognize, how well the clustering results are compatible and fit with the data, and to compare the results of different sets of cluster methods for determining the best one. Generally, there are three numerical criteria to evaluate clustering, namely external indexes, internal indexes, and relative indexes (Halkidi, et al. , 2001).

2.5.1 External Indexes

External indexes are used to measure the similarity of the processed cluster to the externally provided ground truth, such as cluster data points with class labels that are often built by human experts (Halkidi, et al., 2001). In the literature, this index is known as external validation, external criterion, or supervised methods through the class labels, such as entropy, purity, precision (Amigó, et al., 2009; Halkidi, et al., 2001).

Cluster *Purity* is a simple and easy evaluation method to measure the quality of a clustering (Zhao & Karypis, 2004). When $G=\{G_1 ,G_2 , ...,G_M\}$ is a set of the ground truth clusters, and $C=\{C_1 , C_2 , ...,C_M\}$ is a set of the processed clusters by a special clustering algorithm, the purity of cluster *C* is measured by considering cluster *G*.

The cluster *Purity* method assigns each cluster to the special class label, which is most frequent in the cluster, and then the clustering accuracy is measured by computing the number of correctly assigned data points and dividing by the number of data points in the cluster. The purity of the cluster C_j is computed in Equation (2.15):

$$Purity\ (C_j) = {}^{max(|C_j \cap G_i|)}\big/_{|C_j|} \tag{2.15}$$

here $|C_j|$ is the size of cluster C_j and $max(|C_j \cap G_i|)$ shows the number of data points of the class G_i which are assigned to cluster C_j. Also, the total purity of a clustering result is computed as shown in Equation (2.16), which can be measured as a weighted sum of individual cluster purities.

$$Purity(C, G) = \sum_{j=1}^{k} {|C_j|}/{|D|} \times Purity(C_j) \quad (2.16)$$

Where *K* is the number of clusters, $|D|$ is the size of the dataset D. The accuracy by cluster *Purity* will be between 0 and 1, and the *Purity* is 1 if each object gets its own cluster. The large number of clusters results in high purity, therefore, just computing the cluster *Purity* is not suitable to show the quality measurement (Zhao & Karypis, 2004).

Density of the *Correctly Classified Nodes (CCN)* (Bouchachia, et al., 2007; Camastra & Verri, 2005; Costa & Oliveira, 2007) is the simple method for external evaluation measurement of clustering. The accuracies of the methods were measured through the number of clusters and the number of the *CCN*. The *CCN* showed the total of nodes and their densities, with the correct class in the correct related cluster, in all created clusters by the method. The *CCN* is shown in Equation (2.17).

$$CCN = {|t_n| + |t_p|}/{|t_n| + |t_p| + |f_n| + |f_p|} \quad (2.17)$$

$|t_p|$ is true positives, which refers to the number of the positive objects that are laid in correct related cluster, $|t_n|$ is true negatives, which refers to the number of the negative objects that are laid in correct related cluster. The error clustering $|f_n|$ is false negative, which refers to the number of the negative objects that are laid in incorrect related cluster, and $|f_p|$ false positive which is which refers to the number of the positive objects that are laid in incorrect related cluster (Bouchachia, et al., 2007; Camastra & Verri, 2005; Costa & Oliveira, 2007).

Folkes and Mallow (FM) **Index** measures (Fowlkes & Mallows, 1983) is defined as shown in Equation (2.18):

$$FM(C,T) = \sqrt{{|t_p|}/{|t_p| + |f_n|} \times {|t_p|}/{|t_p| + |f_p|}} \tag{2.18}$$

In order to calculate the *FM*, we consider the same parameters as defined for *CCN*.

Jaccard **Score** (Fowlkes & Mallows, 1983) is one of the evaluation methods in an external index (Halkidi, et al., 2001; Jiang & Ren, 2011). The Equation (2.19) shows the *Jaccard* index:

$$Jaccard(C,T) = \sqrt{{|t_p|}/{|t_p| + |f_p| + |f_n|}} \tag{2.19}$$

Rand **Index** is popular measure the evaluation method in external index, that is, how the clustering results are close to the ground truth (Halkidi, et al., 2001; Jiang & Ren, 2011). The agreement between *C* and *G* can be estimated, as shown in Equation (2.20):

$$Rand(C,G) = \sqrt{{|t_p| + |t_n|}/{|t_p| + |t_n| + |f_p| + |f_n|}} \tag{2.20}$$

Adjusted Rand Index (ARI) (Hubert & Arabie, 1985) can only take a value between 0 and 1, and may can not take a constant value such as zero. Hence, the *ARI* can take a negative value if the index is less than the expected index. Therefore, the *ARI* is the corrected for chance version of the *RI*, which works better than *RI* and many other indexes (Milligan & Cooper, 1986; Steinley, 2004). As shown in Equation (2.21), the expected *RI* of random labelling *E[RI]*, is pushed aside:

$$ARI(C,G) = {RI - E[RI]}\Big/{\max(RI) - E[RI]} \tag{2.21}$$

F-measure (Van Rijsbergen, 1979) is a strong established measure for computing the quality of any given clustering result with respect to ground truth, and shows how closely each cluster matches a set of categories of ground truth. The *F-measure* is computed by considering $Precision(C,G)$ and $Recall(C,G)$. $Precision(C,G)$ is the probability that a randomly selected recovered instance is relevant, as shown in equation (2.22).

$$Precision(C,G) = {|t_p|}\Big/{|t_p| + |f_p|} \tag{2.22}$$

$Recall(C,G)$ is the probability that a randomly selected relevant instance is recovered in a search, as shown in Equation (2.23).

$$Recall(C,G) = {|t_p|}\Big/{|t_p| + |f_n|} \tag{2.23}$$

The *F-measure* is called balanced *F-Score* function accuracy, or *F1 measure* because recall and precision are weighted, as shown in Equations (2.24) and (2.25) (Andrew, 2014; Chaimontree, et al., 2010; Rendón, et al., 2011; Sung & Mukkamala, 2003).

$$F - measure(C,G) = {2.\big(Precision(C,G).Recall(C,G)\big)}\Big/{\big(Precision(C,G) + Recall(C,G)\big)} \tag{2.24}$$

$$Error(C,G) = {\big(|f_p| + |f_n|\big)}\Big/{\big(|t_n| + |t_p| + |f_n| + |f_p|\big)} \tag{2.25}$$

Table 2-2 illustrates the details of variables and formulas of the *F-measure* for dataset with two classes.

Table 2-2: Comparing characters to compute the *F-measure*

		Correct result / Clustering	
		E_1	E_2
Obtained results of Clustering	E_1	$\|t_p\|$	$\|f_p\|$
	E_2	$\|f_n\|$	$\|t_n\|$

Table 2-3 shows an example for a computing *F-measure* of clustering of a dataset with three classes based on the definitions of Table 2-3 (Andrew, 2014). Parameter *i* refers to the special class, that i ={1,2,3} , and *j* refers to the special cluster, that *j*={1,2,3}. t_{ci} is true objects, which refers to the number of the objects that are laid in correct related cluster. f_{ci} and f'_{ci} are false objects related to first and second other classes, which refer to the number of the objects that are laid in incorrect cluster. Parameter M_j is total members of cluster j. T_i total number of objects with class *i*.

Table 2-3: Computing *F-measure* of clustering of a dataset with three classes

a) Distribution the members of each cluster with their related classes

	Class1	Class2	Class3	Total members of each cluster
Cluster1	$\lvert t_{c1}\rvert$	$\lvert f_{c1}\rvert$	$\lvert f'_{c1}\rvert$	$M_1=\lvert t_{c1}\rvert+\lvert f_{c1}\rvert+\lvert f'_{c1}\rvert$
Cluster2	$\lvert f_{c2}\rvert$	$\lvert t_{c2}\rvert$	$\lvert f'_{c2}\rvert$	$M_2=\lvert t_{c2}\rvert+\lvert f_{c2}\rvert+\lvert f'_{c2}\rvert$
Cluster3	$\lvert f_{c3}\rvert$	$\lvert f'_{c3}\rvert$	$\lvert t_{c3}\rvert$	$M_3=\lvert t_{c3}\rvert+\lvert f_{c3}\rvert+\lvert f'_{c3}\rvert$
Total number of objects of each class	$T_1=\lvert t_{c1}\rvert+\lvert f_{c2}\rvert+\lvert f_{c3}\rvert$	$T_2=\lvert tc2\rvert+\lvert f_{c1}\rvert+\lvert f_{c3}\rvert$	$T_3=\lvert t_{c3}\rvert+\lvert f_{c1}\rvert+\lvert f_{c2}\rvert$	

b) Computing *Recall*

Recall			
	Class1	Class2	Class3
Cluster1	$Recall_{11}=\lvert t_{c1}\rvert/T_1$	$Recall_{21}=\lvert f_{c1}\rvert/T_2$	$Recall_{31}=\lvert f'_{c1}\rvert/T_3$
Cluster2	$Recall_{12}=\lvert fc_2\rvert/T_1$	$Recall_{22}=\lvert t_{c2}\rvert/T_2$	$Recall_{32}=\lvert f'_{c2}\rvert/T_3$
Cluster3	$Recall_{13}=\lvert f_{c3}\rvert/T_1$	$Recall_{23}=\lvert f_{c3}\rvert/T_2$	$Recall_{33}=\lvert t_{c3}\rvert/T_3$

c) Computing *Precision*

Precision			
	Class1	Class2	Class3
Cluster1	$Preision_{11}=\lvert t_{c1}\rvert/M_1$	$Preision_{21}=\lvert f_{c1}\rvert/M_1$	$Preision_{31}=\lvert f'_{c1}\rvert/M_1$
Cluster2	$Preision_{12}=\lvert f_{c2}\rvert/M_2$	$Preision_{22}=\lvert t_{c2}\rvert/M_2$	$Preision_{32}=\lvert f'_{c2}\rvert/M_2$
Cluster3	$Preision_{13}=\lvert f_{c3}\rvert/M_3$	$Preision_{23}=\lvert f_{c3}\rvert/M_3$	$Preision_{33}=\lvert t_{c3}\rvert/M_3$

d) Computing *F-measure*

F-measure			
	Class1	Class2	Class3
Cluster1	$F\text{-}measure_{11}=2*(Recall_{11}*Precision_{11})/(Recall_{11}*Precision_{11})$	$F\text{-}measure_{21}$	$F\text{-}measure_{31}$
Cluster2	$F\text{-}measure_{12}$	$F\text{-}measure_{22}$	$F\text{-}measure_{32}$
Cluster3	$F\text{-}measure_{13}$	$F\text{-}measure_{23}$	$F\text{-}measure_{33}$

Table 2-3 shows the phases of computing an *F-measure* of clustering for dataset with three classes. In phases *b*, *c* of Table 2-3 averages of the best *Recalls* and *Precision* are computed respectively. In phase *d* of Table 2-3 an *F-measure* of each element is computed and consequently the average of the best computed values of the *F-measure* is considered as *F-measure* of the cluster.

Normalized Mutual Information (NMI) (Studholme, et al., 1999) is a measure for computing the quality of clustering result, and improves the purity index in order to overcome the high number of clusters (Estévez, et al. , 2009; Fern & Brodley, 2004). The *NMI* as shown in Equation (2.26) is based on normalization, and can be used to compare clustering results with different numbers of clusters (Manning, et al. , 2008).

$$NMI(CG) = \sum_{i=1}^{k}\sum_{j=1}^{k}\left|C_j \cap G_i\right| \times log\left({|D| \times |C_j \cap G_i|}/{|C_j| \times |G_i|} \right) \Bigg/ \sqrt{\left(\sum_{i=1}^{k}|G_i| \times log\left({|G_i|}/{|D|}\right)\right) \times \left(\sum_{i=1}^{k}|C_j| \times log\left({|C_j|}/{|D|}\right)\right)} \tag{2.26}$$

Entropy (Rohlf, 1974; Xianguang, 1998) is an external measure of clustering, which is a function of the distribution of classes in the resulting clusters. For each cluster C_j , the class distribution of data is computed as the probabilities $Pr(G_i|C_j)$ that is a data point in C_j belongs to class G_i . The normalized entropy is computed based on the class distribution, as shown in Equation (2.27) and (2.28):

$$Entropy(C_j) = -{1}/{\log h} \sum_{i=1}^{h} Pr(G_i|C_j) \times log(C_i|C_j) \tag{2.27}$$

Where $Pr(G_i|C_j) = {|C_i \cap G_i|}/{|C_j|}$ (2.28)

The overall entropy between 0 and 1 is defined as the sum of the individual cluster entropies weighted by the size of each cluster, as shown in Equation (2.29):

$$Entropy(C,G) = {1}/{|D|} \sum_{j=1}^{k} |C_j| \times Entropy(C_j) \tag{2.29}$$

The low amount of the entropy measure shows a good clustering result.

2.5.2 Internal Index

In the case that the ground truth is not available, internal index is applied to measure validation (Han & Kamber, 2011; Marina, 2001). The internal index is called the internal criterion, internal validation, intrinsic and unsupervised evaluation method. Internal evaluation measures the amount of similarities between data points within the clusters and the dissimilarities between data points of different clusters. Therefore, the internal evaluation of clustering is based on only features and information inherent in a dataset. The internal index can only compare different clustering results by using the same model. Internal indexes are used to measure the fitness of a clustering structure without considering the external information. There are two internal measures, namely cohesion and separation.

Cluster Cohesion measures how closely related data objects are in a cluster. Cluster cohesion includes the *sum of squared error*. *Sum of squared error (SSE)* (Han & Kamber, 2006, 2011; Marina, 2001) is an objective function in order to describe the coherence of a cluster. The low amount of the *SSE* shows good result of clustering. The *SSE* measure, is shown in Equation (2.30):

$$SSE = \sum_{j=1}^{c} \sum_{f_i \in C_j} \left(disance\left(F_i, R_j\right)^2 \right) \tag{2.30}$$

Where, F_i is a data point in cluster C_i and R_i is the representative for cluster C_j. Square error cohesion is measured by within cluster sum of SSE, as shown in Equation (2.31):

$$WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2 \tag{2.31}$$

Where m_i is the mean value of the number of objects in cluster *i*.

Cluster Separation measures how distinct a cluster is from other clusters (Han & Kamber, 2011; Marina, 2001).

Separation is measured by the between cluster sum of squares, as shown Equation (2.32):

$$BSS = \sum_i |C_i| (m - m_i)^2 \tag{2.32}$$

where $|C_i|$ is the size of cluster i, and m is the mean value of the number of objects in all clusters.

Silhouette Coefficient (Han & Kamber, 2011) is an internal measure, that combines the ideas of both cohesion and separation for individual points and clusters. The silhouette coefficient for a data point i is computed, as shown in Equation (2.33), which is typically between 0 and 1, and closer 1 is the best result:

$$\textit{Silhouette coefficient} = \begin{cases} 1 - a/b & if\ a < b \\ b/a - 1 & if\ a \geq b\ , not\ the\ usual\ case \end{cases} \tag{2.33}$$

Where a is the average distance of i to the points in its cluster, and b is minimum average distance of i to points in another cluster. The average silhouette width can be considered for internal evaluation, as well.

2.5.3 Relative Index

Relative indexes are used to compare two different clustering methods or clusters, in which, internal or external indexes can be used for pre-specified measure, such as *SSE* or *Entropy* (Halkidi, et al., 2001; Han & Kamber, 2011; Marina, 2001).

2.6 The Overview of the Main Unsupervised Clustering Methods

During clustering, the data are divided into meaningful groups based on the special goals, with related data classified as higher similarities within groups and unrelated data as dissimilarities between groups (Hegland, 2003). For example, as shown in Figure 2-6 for one data point in cluster1, its threshold must be less than the distance between clusters and greater than the distance within each cluster.

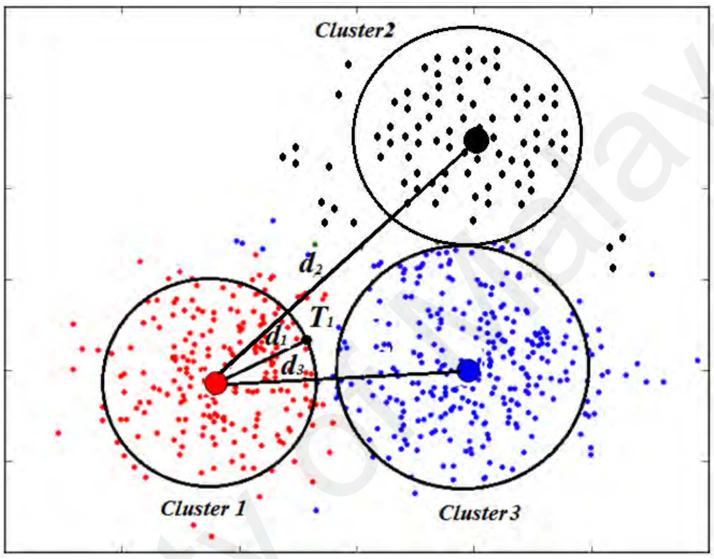


Figure 2-6: A sample of the distances between the clusters and within each cluster

Clustering is often applied for data reduction such as summarization like preprocessing of classification; compression like vector quantization; and finding the nearest neighbours (Han & Kamber, 2006; Hegland, 2003; Kantardzic, 2011). Different unsupervised clustering methods are proposed in the literature, which can be categorized based on: type of input data clustering criterion in order to define the similarities between data points, and fundamental concepts and theory of clustering process. In data mining, the main unsupervised clustering methods can be categorized as follows (Han & Kamber, 2006; Hegland, 2003; Kantardzic, 2011):

2.6.1 Partitioning Clustering

Partitioning clustering divides *n* data into *K* partitions and evaluates them by some criteria functions, such as a distance function based on dissimilarity. Objects within a cluster are similar, and the objects of different clusters are dissimilar in terms of the dataset attributes. Each partition represents a cluster. Also, each partition has at least one object and each object is just in one special partition. For example *K-means* (MacQueen, 1967) is one type of partitioning methods. *K-means* randomly selects *K* objects from the input data and considers them as centroids *of K* partitions, and assign each object to the cluster with the nearest centroid. Consequently, the model computes each centroid as the mean of the objects assigned to it. In order to cluster the data, the centroids are considered as the standard points in the dataset, and the model distributes each group of data points based on a concept of similarity to the features of the centroid or centre of gravity of the cluster. Mostly, *K*-means applies the Euclidean distance in order to compute the dissimilarities between the centroid of each cluster and other objects, and continues this process until the model reaches the minimum error value based on the square-error criterion function. Therefore, the model usually provides a natural tendency summarization of the given data and knowledge to lead to cluster. (Hegland, 2003; Kantardzic, 2011; Pavel, 2002). Figure 2-7 shows a sample diagram of the *K-means* clustering method:

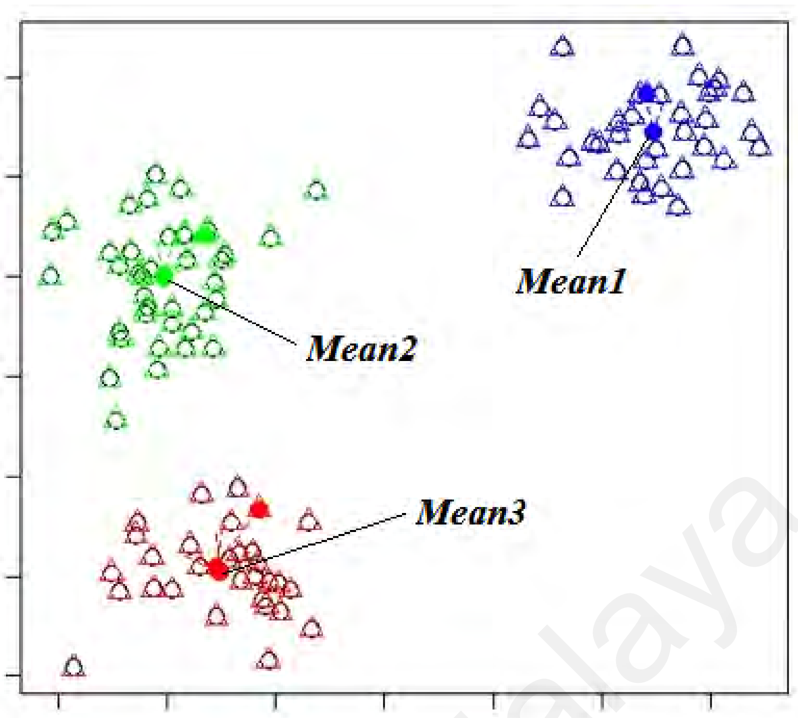


Figure 2-7: The *K*-means clustering method

The *K-means* is a simple and straightforward model for clustering the data, and is based on the firm foundation of analysis of variances. The *K-means* also has some disadvantages: the results strongly depend on the initial guess of centroids (or assignments), the computed local optimum is known to be a far from the global one, it is not obvious what is a good *K* to use, the process is sensitive to outliers, it lacks scalability, only numerical attributes are covered, and resulting clusters can be unbalanced.

Moreover, *K-modes* clustering method is another kind of *K-means* clustering for clustering of categorical data, which frequently updates the modes of clusters instead of the means. Another example of partitioning clustering method is of partitioning around medoids (PAM) (Kaufman & Rousseeuw, 1987). The PAM considers *K* partition of *n* objects. The PAM, firstly initializes random selection of *K* representative objects called medoids in each cluster. The PAM iteratively replaces one of the medoids by one of the non-medoids which improves the total distance of the clustering result for better representative objects. Using PAM is suitable for clustering small datasets. *K-medoid* clustering like *K-means* method pre-describes the number of clusters *K* (Han & Kamber, 2011).

2.6.2 Hierarchical Clustering

Hierarchical clustering has the structure of nested sub-clusters; each cluster has several sub-clusters. The method applies distance matrix as clustering criteria. These methods do not require the number of clusters but a termination condition is necessary (Han & Kamber, 2011; Hegland, 2003; Jain & Dubes, 1988; Kantardzic, 2011). Figure 2-8 shows a sample diagram of the hierarchical clustering method:

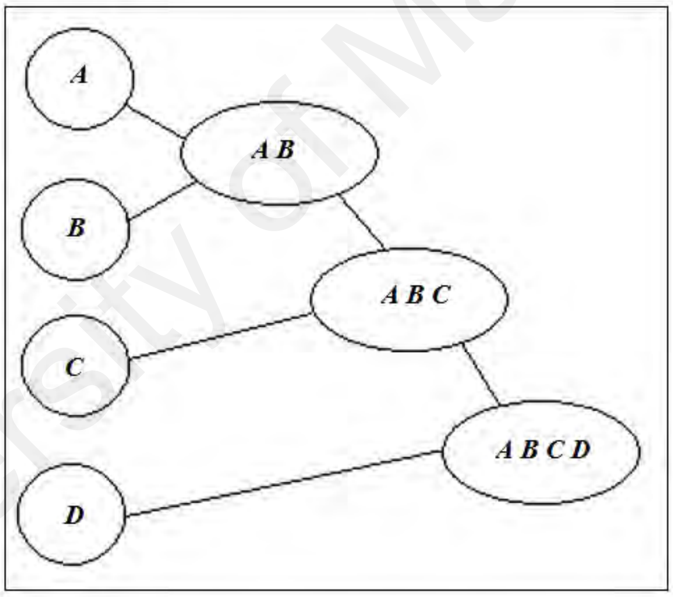


Figure 2-8: The hierarchical clustering method

Generally, there are two types of hierarchical clustering methods, agglomerative and divisive hierarchical clustering methods.

Agglomerative hierarchical clustering has a bottom-up strategy and starts by placing each object in its own cluster and then merges these atomic clusters into a larger cluster, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category. They differ only in their definition of intercluster similarity. Agglomerative nesting (AGNES) method is an example of this method and was introduced by Kaufmann and Rousseeuw (Kaufmann & Rousseeuw, 1990). Initially, AGNES places each object into a cluster of its own. The clusters are then merged step by step according to some criterion, such as Euclidean distance. AGNES uses the single link method and the dissimilarity matrix, and merges nodes that have the least dissimilarity.

The divisive hierarchical clustering method has a top-down strategy and starts with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the diameter of each cluster is within a certain threshold. The diameter is the square root of average mean squared distance between all pairs of points in the cluster. Divisive analysis (DIANA) is an example of this method , and introduced by Kaufmann and Rousseeuw (Kaufmann & Rousseeuw, 1990).

The advantages of hierarchical clustering include: embedded flexibility regarding the level of granularity, ease of handling of any forms of dissimilarity or distance, and consequently, applicability to any attribute types. It does not require the knowledge of the number of clusters. The disadvantages of hierarchical clustering include: vagueness of termination criteria, the fact that most hierarchical algorithms do not revisit once constricted (intermediate) clusters with the purpose of their improvement, and they are unable to undo what was done previously (Han & Kamber, 2011; Hegland, 2003; Jain & Dubes, 1988; Kantardzic, 2011).

2.6.3 Density Based Clustering

Density based clustering model clusters the data based on local cluster criteria, such as density-connected data points, with higher density in the special groups in the one scan. Also, the density based clustering models have the capabilities of controlling the noise and the border of each cluster space based on the property and density functions. The density method assumes that the data point of each cluster is based on a specific probability distribution function. The aim is to identify the clusters in the arbitrary shape, and their distribution and density parameters for a termination condition (Ester, et al., 1996; Ester, et al., 1995; Han & Kamber, 2011; Hegland, 2003; Kantardzic, 2011). Figure 2-9 shows a sample diagram of the density-based clustering method:

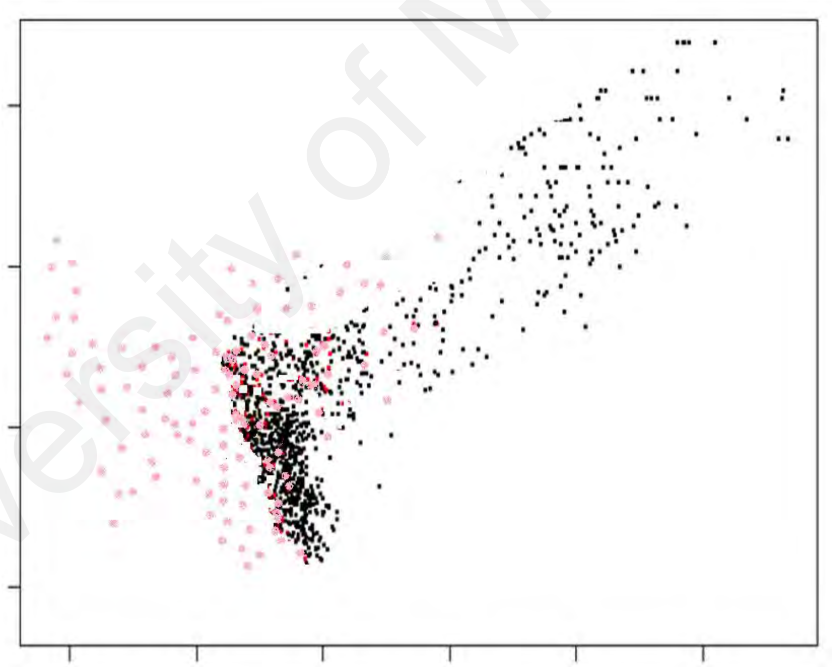


Figure 2-9: The density based clustering method

A clustering method based on connected regions with sufficiently high density (DBSCAN) is an example of the density based method, and was introduced by Esster et al. (Ester, et al., 1996). The density is the number of points within a specified radius *r (Eps)*, which is the maximum radius of the neighbourhood. The radius is the square root of the average distance from any point of the cluster to its centroid. A point is a core point if it has more than a specified number of points (*MinPts*), which is the minimum number of points in an *Eps-neighbourhood* of the points. These are points at the interior of a cluster, a border point has fewer than *MinPts* within *Eps,* but it is in the neibourhood of a core point. Furthermore, a noise point is any point that is not a core point or a border point. If the model finds a core point, a cluster is formed. The model continues the process until all of the points are checked. Figure 2-10 shows the DBSCAN clustering method.

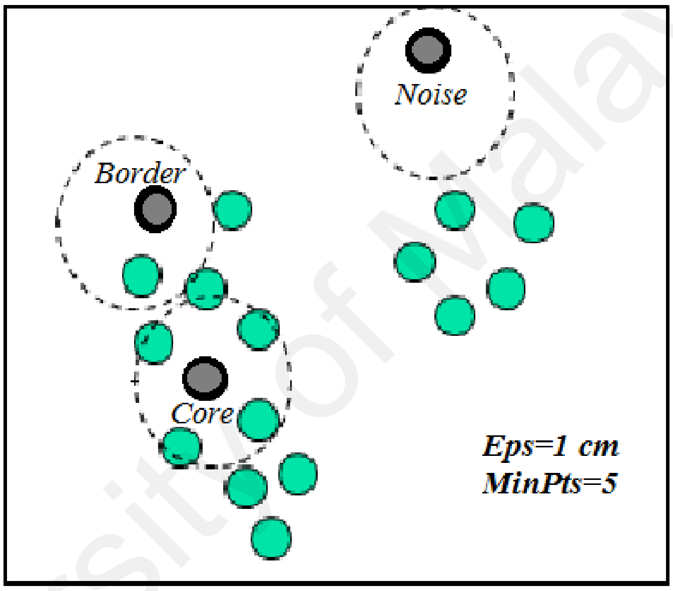


Figure 2-10: The DBSCAN clustering method

Ordering points to identify the clustering structure (OPTICS) (Ankerst, et al., 1999) is based on the DBSCAN. The model develops a special order of the set of values in order to determine parameters to represent the density-based clustering structure. This cluster-ordering includes information equivalent to the density-based clusterings earned from a wide range of parameter settings. The OPTICS is suitable for both automatic and interactive cluster analysis, including finding intrinsic clustering structure.

Some advantages of the density based clustering methods are discovery of arbitrary-shaped clusters with different sizes, immunity to noise and outliers. Some disadvantages of the density based clustering methods are high sensitivity to the input parameters set, disability to describe the clusters and clustering of high dimensional datasets.

2.6.4 Model-based Clustering

Model based clustering methods have their roots in some fields such as statistic and machine learning, which try to find a pattern based on the training data usually for prediction and description. This approach assumes a model for each cluster and finds the best fit of data to that model (Shavlik & Dietterich, 1990). The models may also apply a density function to reflect the special distribution of the data points. These methods can determine the number of clusters and their densities (Ester, et al., 1996; Ester, et al., 1995; Han & Kamber, 2011; Hegland, 2003; Kantardzic, 2011; McCloskey, 2000; Pavel, 2002). The popular methods in the model based clustering are decision tree and neural networks.

A decision tree represents the data by using a hierarchical tree, in which each leaf points to a concept and includes a probabilistic description of that concept. There are several algorithms in this area, such as COBWEB. The COBWEB (Fisher, 1987) considers all attributes of the data are independent, and try to achieve high predictability to nominal variable values in each cluster. The model starts with a tree while consists of empty nodes, and data instances are added one by one, and the structure of the tree is updated to best fit nodes. Each node denotes a concept includes a probabilistic description of that concept. The objects are presented sequentially items. The COBWEB is not suitable for clustering high dimensional and large data because of the time complexity and memory complexity. The model has a high cost of probability distributions and checks all attributes of the data instances (Fisher, 1987). Figure 2-11 shows an example of the COBWEB as a decision tree model.

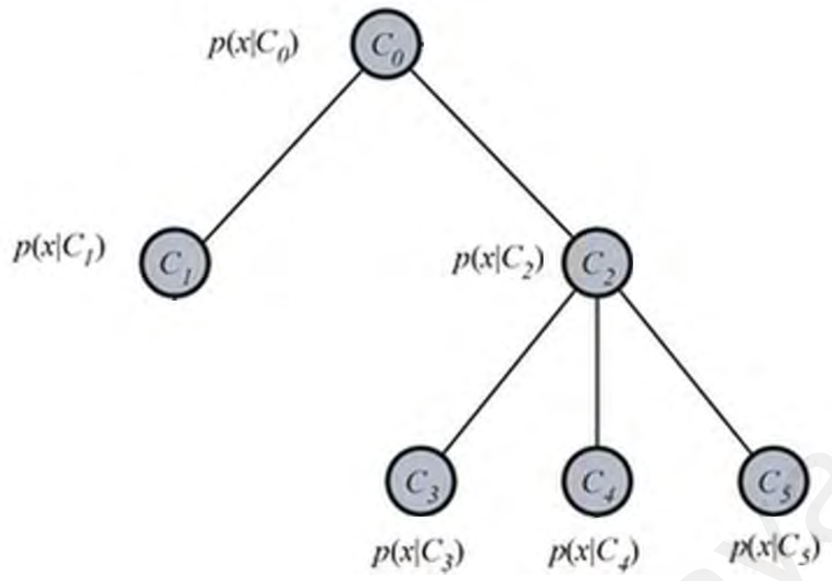


Figure 2-11: An example of the COBWEB in decision tree clustering method

Neural networks are computational models inspired by neurobiology for enhancing and testing computational analogues of neurons. Neural networks are adaptable algorithms that permit users to encode nonlinear relationships between the input and the desirable outputs (Dasarathy, 1990; Goebel & Gruenwald, 1999; Hegland, 2003; Kantardzic, 2011; Kemp, et al., 1997). Neural network clustering method such as Kohonen's self-organizing map (SOM) (Kohonen, 1982) is an example of the model based method. SOM maps multi-dimensional data onto lower dimensional subspaces, with the geometric relationships between points indicating their similarity. SOM generates subspaces with unsupervised learning neural network training through a competitive learning algorithm. The weights are adjusted based on their proximity to the "winning" nodes, that is, the nodes that most closely resembles a sample input (Germano, 1999; Honkela, 1998; Kohonen, 2000; Ultsch & Siemon, 1990). The SOM clustering will be discussed in Section 3.3.1.

Neural networks are particularly useful for solving problems, which cannot be expressed as a series of steps. The neural networks are suitable for extracting rules, quantitative evaluation of these rules, classification, clustering, and regression feature evaluation (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000). Pattern recognition which is a branch of machine learning, is mostly used in the neural networks. Neural networks extract a pattern of data which can be an image, a sound , or any other sources of data. The model based clustering has some critical problems, such as initialization of a set of parameters by user which are used during clustering and low processing time, especially on large and high dimensional datasets, that they affect the accuracy and speed of clustering.

2.6.5 Grid-based Clustering

Grid based clustering are based on multiple resolution grid data structure such as the quantized space. These methods approximate the instance space into a finite number of the cells in the grid structure. First the model predefines a set of grid-cells, then it assigns objects to the suitable grid cell and compute the density of each cell. The density should be below a certain threshold. The clustering is fast and independent of the number of objects, but it is dependent on the number of cells in each dimension in the quantized space (Han & Kamber, 2011; Kantardzic, 2011; Larson et al., 2013). Figure 2-12 shows a sample diagram of the grid-based clustering method:

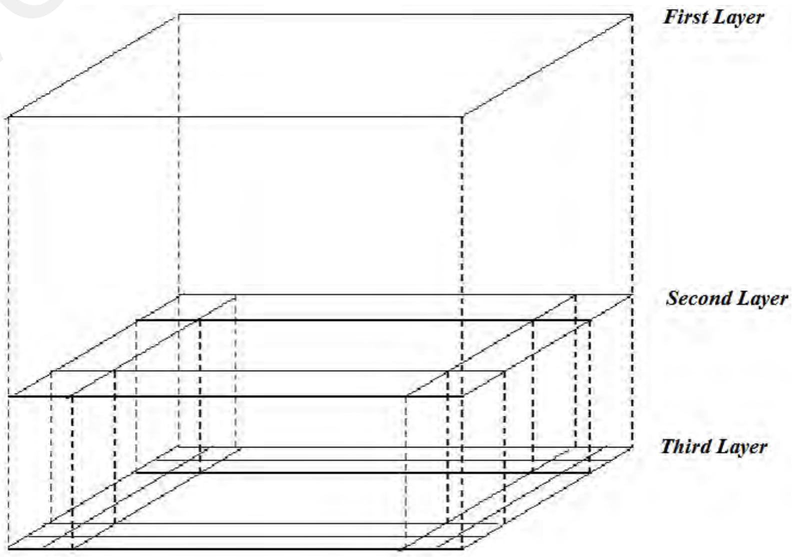


Figure 2-12: The grid-based based clustering method

Statistical information grid (STING) is an example of the grid based clustering method (Wang, et al., 1997). STING is a grid based multiresolution clustering technique and has several levels of rectangular cells matching with different levels of resolution. The cells have a hierarchical structure and each cell at the top level is partitioned to the cells at the lower related level. Each grid cell computes and stores the statistical parameters of information and the data attributes such as mean, maximum, and minimum values, which are used during the processing of clustering.

Grid based clustering can be used for clustering of data incrementally, which is suitable for lifelong data in real environments. Also, the model has the capability to discover data clusters with irregular formats. The disadvantages of the grid based clustering methods are their sensitivity to the order of the data, and the clustering task is dependent on pre-initialization of the parameters such as thresholds. Another example of the grid based clustering is a wave clustering (Sheikholeslami, et al., 1998), which is a multi-resolution clustering approach and applies the signal processing technique. The wave clustering applies the wavelet transform. A wavelet transform is a signal processing technique in order to decompose a signal into different frequency sub-band, with high frequency parts of a signal representing the boundaries, and low frequency high amplitude parts of a signal representing insides of the clusters. The wave cluster is based on the grid based clustering and the density based clustering methods. The wavelet clustering method summarizes data by enforcing a multi-dimensional grid structure onto data space, which are represented in a *n* dimensional feature space. Then, the model iteratively applies wavelet transforms on the feature space to find the filtered data dense parts, in the feature space.

2.6.6 Other Categories of Clustering

Constraint-based or user guided methods consider the feedback of users. These models have several constraints on individual objects, distances, parameters and user specified constraints (Hebooul, et al., 2015; Kamiya, et al., 2007; Prudent & Ennaji, 2005; Shen, et al., 2011). Semi-supervised evolving self-organizing map (Semi-ESOM) (Deng & Kasabov, 2003) clustering is an example that will be explained in Section 3.5.1.

2.7 The Time Complexity and Memory Complexity of the Main Unsupervised Clustering Methods

The time complexity and memory complexity of the main unsupervised clustering methods are summarized in Table 2-4.

Table 2-4: The time complexity and memory complexity of the main unsupervised clustering methods

Unsupervised clustering method	Example method	Time complexity	Memory complexity
Partitioning clustering (Halkidi, et al., 2001)	*K-means*	$O(c.n)$	$O((n+K).m.S_m)$
	K-medoids (PAM)	$O(c.K.(n-K)^2)$	$O(c.K.(n-K)^2.S_m)$
	K-modes	$O(c.n)$	$O(c.K.n.m.S_m)$
Hierarchical clustering (Niknam, et al., 2008)	AGNES	$O(n^2.\log(n))$	$O(n^2.\log(n).S_m)$
	DIANA	$O(n^2.\log(n))$	$O(n^2.\log(n).S_m)$
Density based clustering (Halkidi, et al., 2001)	DBSCAN	$O(nlog(n))$	$O(nlog(n).S_m)$
	OPTICS	$O(nlog(n))$	$O(n.\log(n).S_m)$
Model based clustering (Satyanarayana & Acquaviva, 2014; Maiorana, et al., 2008)	COBWEB	$O(m.\alpha.\beta^2.log(K))$	$O(\alpha.\beta^2.log(K).m.S_m)$
	SOM	$O(c.n.m^2)$	$O(c.n.m^2.S_m)$
Grid based clustering (Halkidi, et al., 2001)	STING	$O(\cup)$	$O(\cup.S_m)$
	WAVE	$O(n)$	$O(n.S_m)$

$*$ *note*:
n is the number of data points, *m* is the number of attributes, *K* the number of clusters, *c* is the number of iterations, S_m is size of each attribute. Also specially, the parameteres $\cup$ is the number of grill cells at low level, γ is mean of successful swaps of the data points with existing methods for the PAM clustering method, and α is the average number of values, as well, β is branching factor for the COBWEB clustering method.

2.8 Summary

This chapter basically discussed the concepts in unsupervised clustering methods in data mining, such as artificial neural network, which is one of the most popular algorithms used in machine learning and data mining. Furthermore, we generally discussed the different aspects of unsupervised clustering, types of data in this area, and related distance measurements and popular clustering validation methods. Understanding the issues of this chapter, is useful to understand the main idea of this study.

CHAPTER 3: CONCEPT IN ONLINE DYNAMIC UNSUPERVISED FEEDFORWARD NEURAL NETWORK CLUSTERING

3.1 Introduction

An artificial neural network (ANN) is one of the numerous algorithms used in machine learning and data mining. An ANN has its roots in mathematics, statistics, numerical analysis, biology and psychology, and is an artificial representation of the human brain that has the capability to simulate its learning process. One category of unsupervised learning is learning raw data and dividing data into meaningful groups with special goals and classifying higher similar data into one group without using any class label. Unsupervised classification is also known as data clustering. In feedforward neural network (FFNN), data processing has only one forward direction from the input layer to the output layer without any backward loop. Unsupervised feedforward neural network (UFFNN) clustering has several advantages such its inherent distributed parallel processing architectures and ability to adjust the interconnection weights, and cluster data by only one forward direction from the input layer to the output layer without any feedback loop. In the real environment while the data are non-stationary, the structure of the data changes frequently. Therefore, the online dynamic unsupervised feedforward neural network (ODUFFNN) clustering methods should involve to have lifelong (online) and the incremental learning ability and be compatible with the changes in the continuous data (Hebooul, et al., 2015; Kasabov, 1998; Liu & Ban, 2015; Liu, et al., 2013).

In this chapter, we study the history, some advantages and disadvantages of the ODUFFNN clustering methods, and review the related literature works.

3.2 Concepts in Feedforward Neural Networks

Neural networks are computational models inspired by neurobiology for enhancing and testing computational analogues of neurons. In feedforward neural networks, the data processing has merely one forward path from the input layer to the output layer without any cycles or backward loop (Andonie & Kovalerchuk, 2007; Bose & Liang, 1996; McCloskey, 2000).

Neural networks are flexible algorithms that let users encode nonlinear relationships between the input and the desirable outputs (Dasarathy, 1990; Goebel & Gruenwald, 1999; Hegland, 2003; Kantardzic, 2011; Kemp, et al., 1997). The neural networks are suitable for extracting rules, quantitative evaluation of these rules, classification, clustering, and regression feature evaluation (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000).

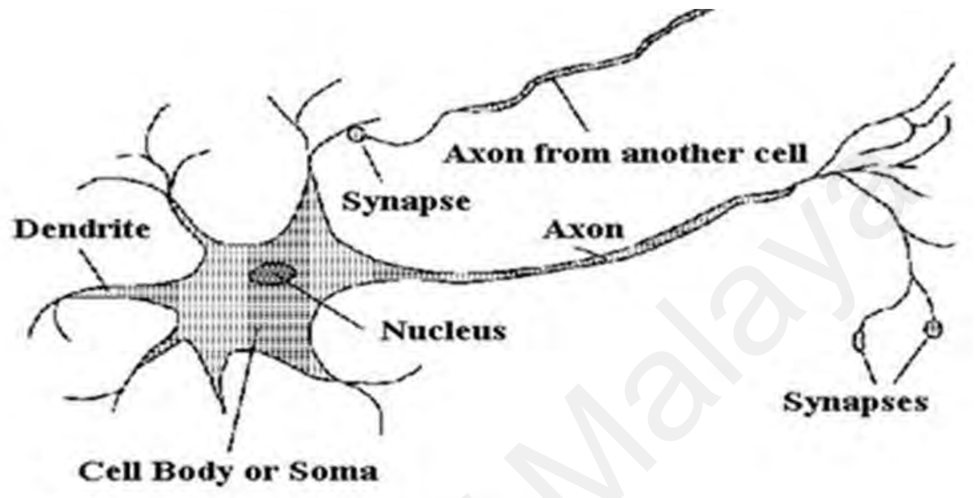
Learning is an important property of the neural network. Neural networks are able to dynamically learn types of input information based on their weights and properties. There are many types of learning rules used in neural networks, which fall under the broad category of supervised learning, unsupervised learning, and reinforcement learning. Learning from observations with unlabeled data in an unsupervised neural network classification is more advantageous than learning by examples in the supervised neural network classification, because preparing the training set is costly and time consuming (Andonie & Kovalerchuk, 2007; Han & Kamber, 2011; Hegland, 2003; Kantardzic, 2011).

In some cases collection of the data from dangerous environment is not possible (Han & Kamber, 2006; Kantardzic, 2011). However to evaluate the performance of unsupervised learning, there is no error or reward suggestion, and the learning process is performed without considering class labels (Demuth, et al., 2008; Kohonen, 1997; Van der Maaten, et al., 2009).

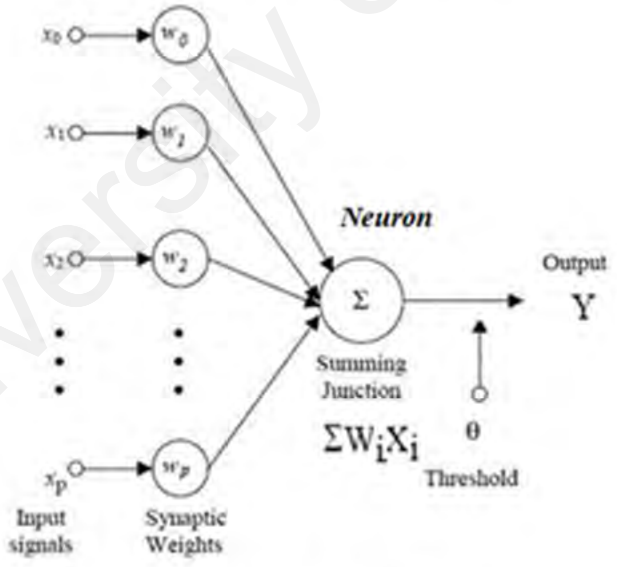
This chapter includes an introduction of neural networks, perspectives of the learning problems, the architectural structure of the neural networks, activation functions and the preprocessing techniques, while focusing on the feedforward neural network.

3.2.1 Neural Network Models

McCulloch and Pitts (McCulloch & Pitts, 1943) introduced an artificial neural network as a connective model and parallel distributed processing, resulting in computational power achieved through a combination of only a few simple processing units based on the idea of weights. The foundations of the artificial neural networks are in mathematics, statistics, numerical analysis, biology. The neural network is also associated with neurobiology for developing and testing computational analogues of neurons (Andonie & Kovalerchuk, 2007; Dasarathy, 1990; Goebel & Gruenwald, 1999; Hegland, 2003; Kantardzic, 2011; Kemp, et al., 1997). The neural network is one of the most popular algorithms used in machine learning and data mining. Hebb (Hebb, 1949) described a synaptic flexibility mechanism in which, if neuron i is close enough to stimulate neuron j at the same time and takes part in its activation repeatedly, the synaptic connection between these two neurons is strengthened and neuron j will be more sensitive to the action of the neuron i. The meaning of the synaptic flexibility is the ability in connection between two neurons to change the strength of responses. Haykin (1999) completed this idea with a parallel distributed processor (Haykin, 1999). As shown in Figure 3-1, the design of an artificial neural network is similar to a biological neural network. A neuron consists of a cell body (soma) with a nucleus, dendrites, and axon for connection with other neurons. Joints between neurons are called synapses. A signal is passed via a neuron to another neuron through electrochemical reactions. If the inputs exceed the specific threshold level, the soma will sum the input and send electrical impulse to the particular axon that will be fired. This signal is sent to another neuron in the synapse and the cycle continues. Figure 3-1a shows a biological neuron cell and Figure 3-1b shows a simple artificial neuron.



a. A biological neuron cell



b. An artificial neuron unit

Figure 3-1: The neuron

An artificial neural network consists of neurons in three kinds of layers; the input layer, hidden layer and the output layer (Haykin, 1999). Each unit has its weight, and the sum of the input data is computed for each unit individually. Next, every unit computes the activation function in order to obtain the desired output, before passing it down to the next layer in a feedforward topology. The number of layers, and the units of each layer should be selected. The training model works on the basis of the network weights and some thresholds. They also must be set to minimize the prediction error made by the network. Doszkocs et al. (1990) explained, “The components of neural networks consist of units similar to neurons and activation functions, which are used as input for local processing and obtained their weights from neighbouring nodes”. The learning rules are local procedures based on updating the weights and biases (Doszkocs, et al., 1990). Haykin (1999) recognized some useful capabilities for the neural networks:

Nonlinearity: A neural network is made up of interconnections of nonlinear neurons. The non-linear modelling power of neural networks lies in adjusting the network to lower its error between the output activations and the target activations in the neural network. Nonetheless, a low error rate does not necessarily mean that errors have been successfully minimized. Given a sufficiently large neural network with non-linear hidden units, it has been shown the model is able to approximate any continuous mapping from input to output, within the range of the desired degree of accuracy (Cybenko, 1989; Haykin, 1999).

Input and Output Mapping: Artificial neural network models have the capabilities of input and output mapping such as non-parametric statistical inferences which assumes the structure of the model is not fixed. The model arises in complexity to adjust the size of the data, and the parametric distributions and connections between individual variables. Neural network models often employ the weights of each connection in order to reach the desirable output (Haykin, 1999).

Adaptation: Neural networks have the ability to change the weights in response to their environment(Haykin, 1999).

Evidential Response: A neural network can be applied to classification problems and can be used in confidence for decision making (Haykin, 1999).

Contextual information: In a neural network, every neuron accepts the effects of the global activity of all neurons (Haykin, 1999).

Fault Tolerance: Neural networks have the inherent capability of robust computation, especially when the algorithm trains features and conditions of the network properly. For example, if the distribution or the connection of the data is damaged, the neural network can recall the pattern or reconnect the data (Haykin, 1999).

Very Large Scale Integrated (VLSI) Implements Ability: Neural networks are suitable for solving the problems in using VLSI technology, because of its parallel nature and the inherent distributed parallel processing architectures (Haykin, 1999).

Uniformity of the Analysis and Design: Neural networks have a universal feature of information processors in the neurons through the meaning of learning and theories, as well as integration of the modules. For example, each neuron represents the common feature of all neurons in the neural network. Learning algorithms can be applied in different environments (Haykin, 1999).

Neurobiological Analogy: Originally, a neural network is a computational model inspired by psychology and neurobiology with the capabilty of developing and testing computational analogues of neurons (Haykin, 1999).

Figure 3-2 shows a sample neural network with details:

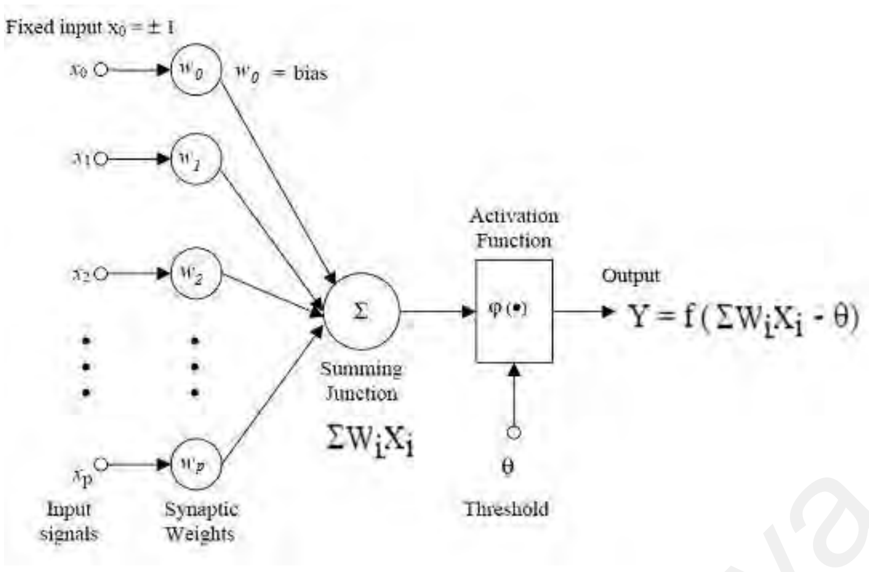


Figure 3-2: A sample neural network

Figure 3-2 is an example of the neural network following three basic elements: 1) A set of synapses or connections. Each connection has a special weight W_i that is adjusted to train the knowledge. 2) A sum function for summing the input data X_i and related weights W_i as $\sum X_i.W_i$. 3) An activation function for transforming the output of a neuron. The output Y is computed by considering the output of sum function and predetermined threshold θ as shown in Equation (3.1).

$$Y = f(\sum X_i.W_i - \theta) \tag{3.1}$$

Neural networks are commonly known as black boxes, however, they can provide simple and accurate sets of logical rules and knowledge from datasets (Han & Kamber, 2011; Kantardzic, 2011; Kemp, et al., 1997). How can we understand what a neural network model has learned? One of the critical issues in the neural network is that the knowledge of the neural network cannot be represented properly, and the human is not able to interpret a network of units and their weight interconnections. Therefore, extracting the knowledge embedded in trained neural networks, and representing the knowledge symbolically are domains of research. There are different methods such as extracting rules from neural networks and sensitivity analysis, for analysing and understanding the process of training in a neural network model:

Extracting Rules from Neural Networks: The neural network methods are limited to the training process, topology and discretization of input data. The first step of extracting rules from the neural network is network pruning by removing weak weight connections that have the least effect on the trained network results. Therefore, the set of data and activation values of the hidden and output units of the neural networks should be studied to describe their connections and knowledge. The algorithms can derive rules in different forms, such as "If-Then" in the boolean rules with the rule ancestor sequentially similar decision tree. In this method, if there are some conditions, then there is a special output (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Han & Kamber, 2011; Kantardzic, 2011; Kemp, et al., 1997).

Sensitivity Analysis: Sensitivity analysis is used to evaluate the effective input data on the output of the neural network. For this purpose, an input variable is changed while the remaining input variables are fixed at some value. In the meanwhile, the changes of the neural network output are monitored (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Han & Kamber, 2011; Kantardzic, 2011; Kemp, et al., 1997).

3.2.2 Perspectives of Learning Problems

Learning is an imperative feature of the artificial neural network in machine learning. There are numerous types of learning rules that are categorized broadly under supervised learning, unsupervised learning, and reinforcement learning (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Han & Kamber, 2011; Kantardzic, 2011). Supervised learning is similar to unsupervised learning in the sense that a training set is provided. However, in supervised learning the desired output is provided and the weight matrix is adjusted based on the difference between the predicted output and the actual output of the neural network. One of the popular supervised feedforward neural network models is the back propagation network (BPN) (Werbos, 1974). The BPN uses gradient-based optimization methods in two basic steps: to calculate the gradient of the error function, and to employ the gradient. The optimization procedure includes a high number of small steps, causing the learning to be considerably slow. Optimization problem in supervised learning can be shown as the sum of squared errors between the output activations and the target activations in the neural network as well as the minimum weights (Andonie & Kovalerchuk, 2007; Bose & Liang, 1996; Craven & Shavlik, 1997). Approaches to unsupervised learning in machine learning are statistical modelling, compression, filtering, blind source separation, and clustering. Unsupervised learning or self-organized learning finds symmetries in the data represented by unlabeled input data. However, to assess the performance of unsupervised learning, there is no error or reward signal (Han & Kamber, 2011; Hegland, 2003; Kantardzic, 2011).

In this thesis, the clustering aspect of unsupervised neural network classification is considered as an unsupervised neural network clustering. In reinforcement learning, the model is capable of generating certain effects and interactions with dynamic environment for recognizing an unknown attribute value. At each point in time, the environment generates an observation for this unknown attribute value which is the reward or reinforcement for the model. Consequently, the model can select and use suitable rule and interacts with the environment for more rewards. The environment is dynamic, hence the long-run cost is unknown, but it can be estimated. Neural networks are frequently used in reinforcement learning as part of the algorithm. The tasks include control problems and sequential decision making tasks (Lin, et al., 2006). The main advantages of the neural networks are the abilities of scaling and learning as in the machine learning algorithms. Neural networks are useful, especially for analysing unknown data (McCloskey, 2000).

3.2.3 Architecture of Neural Networks

Neural Networks can be classified into feedforward neural network (FFNN) and recurrent networks (RN) (also called feedback). In the FFNN model, connections of units are in one direction, forward from the input nodes, through the hidden nodes and to the output nodes. In the RN models, connections between units form a directed cycle. Neural Networks can also be classified as the following (Aleksander & Morton, 1990; Haykin, 1994; Lippmann, 1987):

Single-layer perceptrons (SLP) are the feedforward networks with only two layers which has a single layer of output nodes. The input layer is fixed with weight connections to the output layer for learning as shown in Figure 3-3. The weighted sum function is calculated in each node.

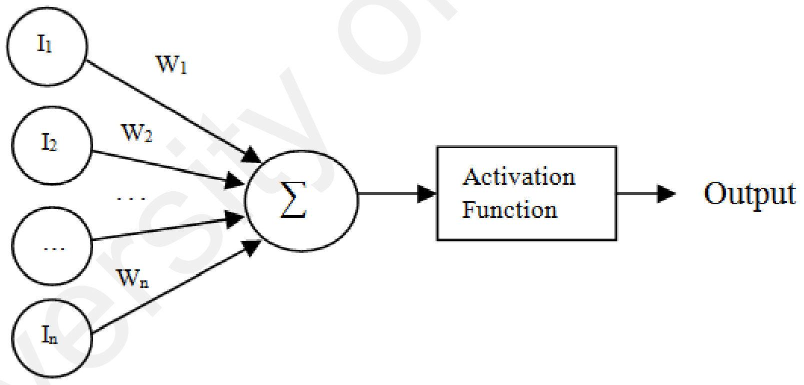


Figure 3-3: Single-layer perceptrons feedforward network

Multilayer perceptrons (MLP) feedforward network, which consists of an input layer, at least one hidden layer, and an output layer. Each layer consists of several computational neurons as shown in Figure 3-4. The MLP is used together with nonlinear activation functions with a high degree of connection and computation.

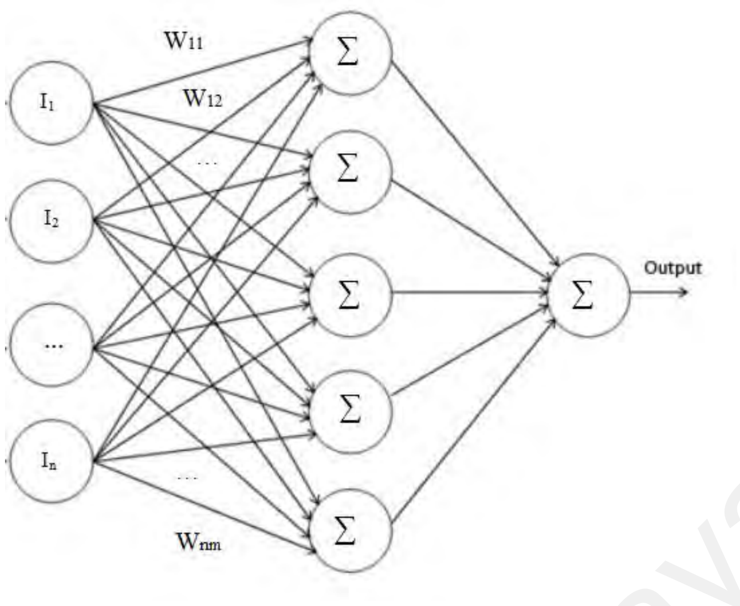


Figure 3-4: Multilayer perceptrons feedforward network

Recurrent networks (RN) are the neural networks with one or several feedback connections or closed cycles. This topology gives the different capabilities of non-linear mapping, and special internal memory usage for processing arbitrary sequences of inputs to the RN models. The activation functions and errors are computed at sequence time *t* based on stored learnt information about input data through involvement of the initial and last states. Therefore, the RN has the capability of temporal processing and sequential learning. The RN can have different forms, such as the MLP plus added loop, and can be fully connected (Haykin, 1999). There is a local global feedback for the whole network as shown in Figure 3-5:

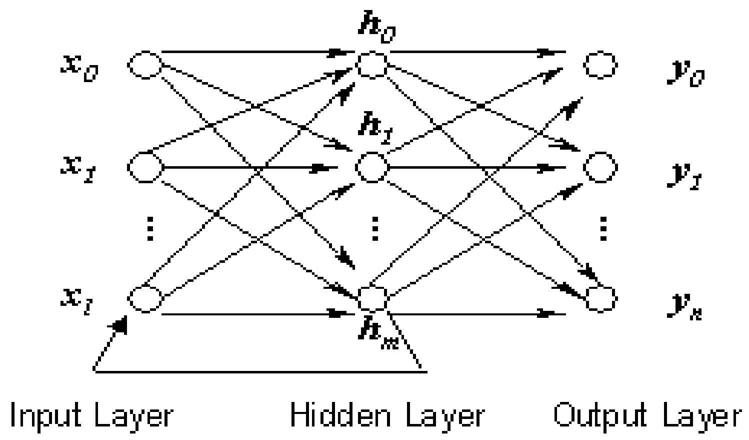


Figure 3-5: Recurrent networks

The connection weights in the RN models can be symmetric or asymmetric based on the measuring time of application of activation functions. In the symmetric RN, there is $(W_{ij} = W_{ji})$, and the network converges to stable point, however, the RN can not adjust temporal sequences of pattern. In the asymmetric RN, there is $(W_{ij} \neq W_{ji})$, and the dynamics of the network can show limit cycles and with a selection of suitable weights, temporal spatial patterns can be generated and stored in the network (Haykin, 1999).

3.2.4 History of Early Artificial Feedforward Neural Network Models

McCulloch and Walter Pitts (1943) described biological neuron as a threshold logic unit (TLU) with binary input sample neurons and one binary output neuron (McCulloch & Pitts, 1943). In Equation (3.2) the output *Y* of the TLU is:

$$Y = \sum_j X_j W_j + bias \qquad (3.2)$$

Where *Y* is the output and is equal to the weighted sum function and the bias, as shown in Equation (3.2). Each neuron has a special weight and bias.

McCulloch and Walter Pitts illustrated that universal computational capability is possible by using different networks with combination of units which create a special construction of logical function (McCulloch & Pitts, 1943). However, they did not introduce any learning rule. Hebb (Hebb, 1949) developed the meaning of the first learning rule known as the Hebbian learning. The simplest topology of the single layer unsupervised feedforward neural network with a linear unit and the Hebbian learning is shown in Figure 3-6 (Kornel & Dave, 2006).

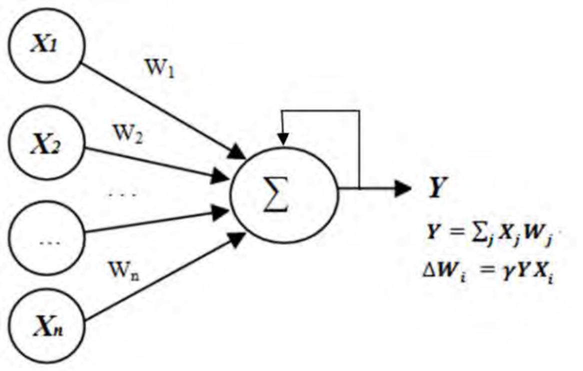


Figure 3-6: Single layer unsupervised feedforward neural network with Hebbian learning

Hebb described a synaptic flexibility mechanism in which, if neuron i is close enough to stimulate neuron j at the same time and takes part in its activation repeatedly, the synaptic connection between these two neurons is strengthened and neuron j will be more sensitive to the action of neuron i. Hebbian rule is shown in Equation (3.3):

$$\Delta W_i = \gamma Y X_i \quad (3.3)$$

X is the input vector, Y is the output vector and γ is learning rate where $\gamma > 0$ is used for controlling the size of each iteration of training.

Frank Rosenblatt (Rosenblatt, 1958) proposed the perceptron model that can be used as a pattern association or a pattern classifier. The single layer perceptron (SLP) as a linear separability model includes one input layer of binary units, and one output layer of binary units. Therefore, there is just one layer without any hidden layer for adjusting the weights. The output layer of the SLP contains a hard-limiting threshold output function. The single layer perceptron is shown in Figure 3-7:

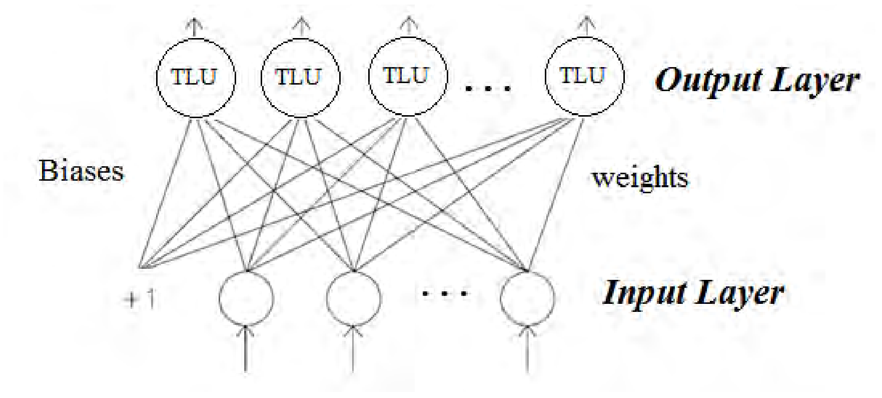


Figure 3-7: The single layer perceptron sample

Widrow and Hoff (Widrow & Hoff, 1960) proposed the ADAptive linear neuron (ADALIN) which is a TLU with a bipolar output and bipolar inputs (-1,1) and weighted sum function plus a bias. If the sum is bigger than 0 then the output is equal to 1 , and if the sum is smaller than 0 or equal to 0 then the output is equal to -1. A multiple ADALINE (MADALINE) is basically the single layer perceptron with bipolar inputs and outputs. In 1960, Widrow and Hoff proposed the Delta rule to use in the ADALINE/MADALINE training, that applied by the two layer feedforward neural network model includes the input and the output layers without any hidden layer. In addition, the Delta rule is known as the least mean square (LMS) algorithm with supervised learning and an error correction rule (Widrow & Hoff, 1960). Also, the single layer perceptron applies a gradient descent learning rule to update the weights (Bose & Liang, 1996; Demuth, et al., 2008; Han & Kamber, 2011).

Minsky and Papert (Minsky & Seymour, 1969) expressed the limitations of the SLP. In order to develop linear separability, the multiLayer perceptron (MLP) was considered with a hidden layer of the units for adjusting weights. Figure 3-8 shows an example of the MLP:

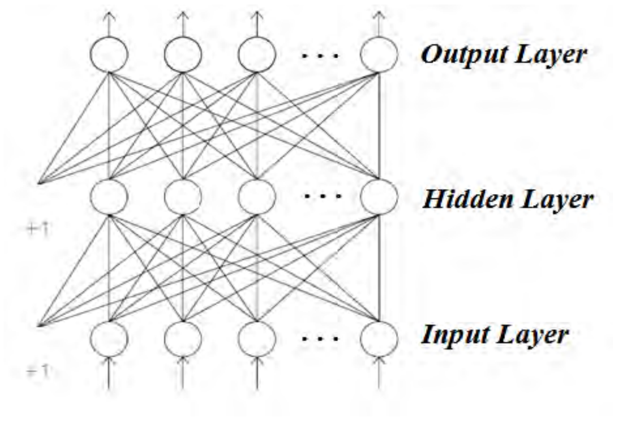


Figure 3-8: The multiLayer perceptron sample

The topology of the MLP must have at least one hidden layer with nonlinear activation function. The output can have linear or nonlinear activation functions.

3.2.5 Activation Functions

Neural networks often transform the input samples to the output through an activation function. This activation function includes the identity function, a *K-step* function such as binary step function, sigmoid function, and bipolar sigmoid function as follows (Demuth, et al., 2008):

Identity function: The identity function is a linear activation function that transforms the input data for computing output with an unknown range. The function *f(X)* in Equation (3.4) is an activation function for transforming *X* and the output will be *X* as shown in Figure 3-9 (Demuth, et al., 2008).

$$f(X) = X \tag{3.4}$$

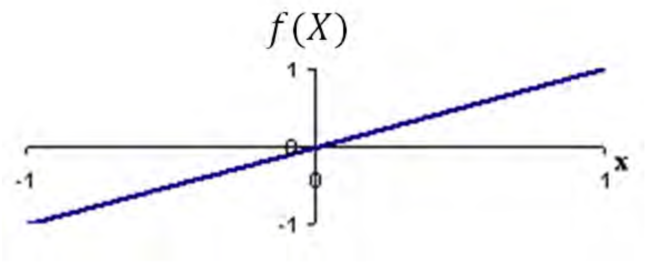


Figure 3-9: Identity function

***K-step* function:** The threshold function is a linear activation function for transforming the input data. This kind of function is limited with *K* values based on the number of classes of the dataset; and each limited domain of thresholds refers to the special output value of the *K-step* function. The *Binary-step* function is a branch of the *K-step* function for two data classes 0, and 1 as shown in Equation (3.5) which is often used in single layer networks. The output will be 0 or 1 based on the threshold θ as shown in Figure 3-10 (Demuth, et al., 2008).

$$g(X) = \begin{cases} 1 & If\ (X \geq \theta) \\ 0 & If\ (X < \theta) \end{cases} \tag{3.5}$$

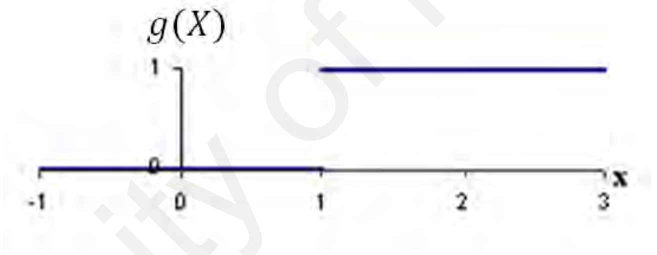


Figure 3-10: Binary step function

Sigmoid function: The sigmoid function as shown in Equation (3.6) is a nonlinear differentiable function that maps the perceptron's inputs to a range of values such as 0 to 1. This function is especially used in the back-propagation neural network to reduce computation of the weights in training. It is often used for the desired output between 0 and 1, as shown in Figure 3-11:

$$v(X) = \frac{1}{1+e^{-\alpha x}} \tag{3.6}$$

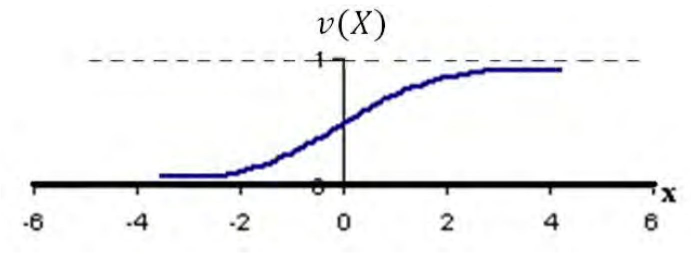


Figure 3-11: Sigmoid function

The sigmoid function *v(X)* is equivalent to $\frac{1}{1+e^{-\alpha x}}$ since *0<v(X)< 1*. In the neural network applications, the coefficient α is a real number constant. Usually α is chosen between 0.5 and 2 experimentally. As a starting point α can be considered equal to 1 and by fine tuning the network, the best value for this coefficient can be found. The sigmoid function converts values that are less than 0.5 to 0, and values greater than 0.5 to 1 (Demuth, et al., 2008).

Bipolar sigmoid function: The bipolar sigmoid function is provided in Equation (3.7); $z(X)$ is a nonlinear function that is used to convert the desired output into between -1 and 1 as shown in Figure 3-12. The performance of the multilayer network is increased significantly through the use of the bipolar sigmoid function (Demuth, et al., 2008).

$$z(X) = \frac{1-e^{-x}}{1+e^{-x}} \tag{3.7}$$

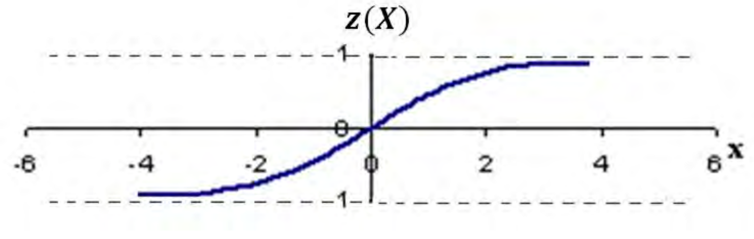


Figure 3-12: Bipolar sigmoid function

3.2.6 Preprocessing Techniques of Feedforward Neural Networks

The data preprocessing and pre-training in the FFNN helps to choose the right input attributes and to generate the desirable output with better results in terms of both speed and accuracy. Changing the input data or initial conditions can immediately affect the supervised and unsupervised classification accuracy in the FFNN models. Also, without preprocessing, the classification process is comparatively slow and it may even converge leading to an incomplete classification (Han & Kamber, 2011; Hinton & Salakhutdinov, 2006; Ziarko & Shan, 1994). Nonetheless, the main problems in data preprocessing and pre-training include identifying suitable input values and attributes, suitable weights, and considerable processing time (Andonie & Kovalerchuk, 2007; Demuth, et al., 2008; Han & Kamber, 2011; Jolliffe, 1986). Existing pre-training techniques generate suitable weights for reducing the training process, but with the application of random values for initial weights (DeMers & Cottrell, 1993; Van der Maaten, et al., 2009). In this section, we review the powerful functions of data preprocessing,pre-training and their advantages and disadvantages.

- **Data Preprocessing Techniques of Feedforward Neural Network**

Neural network learning will be faster and the performance will be better when the input values are pre-processed (Erhan et al., 2010; Gabrys, 2012; Larochelle, et al., 2009). Several powerful data preprocessing functions will be discussed in this study on the basis of improving efficiency of the feedforward neural network models (Demuth, et al., 2008; Erhan, et al., 2010; Gabrys, 2012; Han & Kamber, 2011; Larochelle, et al., 2009). These are mathematical and statistical functions to scale, filter, and pre-process the data.

Data Cleansing: In the real world, data may be incomplete, in attribute values and certain noisy with outliers, and or in codes or names (Maletic & Marcus, 2000; Marcus, et al., 2001). Outliers are data objects, that are considerably different than most of the other data objects in the dataset (Bose & Liang, 1996; Demuth, et al., 2008; Han & Kamber, 2011; Hegland, 2003). Data cleaning has different techniques to solve these problems, such as filling in missing values, identifying outliers and smooth out noisy data, and correcting inconsistent data. .There are some techniques to handle missing values, such as the statistical functions for data cleaning are the mean, the standard deviation, and the range (Barnett, 1994). The technique for recognizing outliers and data cleaning, is a pattern-based technique, that uses the combined strategies of partitioning, classification and clustering (Knorr, et al., 2000; Murtagh, 1983). Therefore, clustering method has the capability to recognize outliers, when similar values are organized into groups, or clusters. The noisy data that fall outside the set of clusters, are considered as outliers (Han & Kamber, 2011). Also, some important issues in data mining are duplication and redundancy of data (Galhardas, et al., 2001).

Data Integration: The merging of data from multiple data stores causes the problems of data duplication, redundancy and conflicts of data values, therefore an integrated access to multiple data sources is necessary. Data integration utilizes two main techniques: schema integration and data integration (Christen, 2005; Han & Kamber, 2011). In the real world scenario, attribute values from different sources may differ because of different representation, scaling, or encoding such as different metric units. The functions of the mean, standard deviation and correlations between attributes of data can solve this problem (Christen, 2005; Ganesh, et al., 1996; Han & Kamber, 2011).

Data Transformation: Data transformation involves smoothing, aggregation, normalization and data reduction. Noise is a random error or variance in a measured variable. Smoothing is used to remove noise from the data, such as clustering.

Aggregation is used for summarizing the data based on dimensions. Therefore, the result is smaller in volume, without loss of information necessary for the analysis task. For example, the aggregation technique is useful to prepare the annual report sales of a shopping centre that is based on the monthly information on the sales. Normalization is used for scaling data attributes to fall within a small specified range, such as the *MinMax* normalization, which is often used in standard back propagation network (BPN) as a supervised clasiffication (Werbos, 1974). The *MinMax* normalization technique is used to transform an input value of each attribute to fit in specific range such as [0,1]. The following Equation, Equation (3.8) shows the formula to normalize the input values (Han & Kamber, 2011; Kantardzic, 2011):

$$Normalized(X_{ij}) = \left({X_{ij} - Min(X_{ij})} \Big/ {Max(X_{ij}) - Min(X_{ij})} \right) \times (newMax - newMin) + newMin \qquad (3.8)$$

Where X_{ij} is j_{th} attribute value of the input data X_i , and has a special range and domain, with $Min(X_{ij})$ as the minimum value in this domain, and $Max(X_{ij})$ is the maximum value in this domain. $newMin_i$ is the minimum value for the specific domain of [0,1] which is 0, and $newMax_i$ is the maximum value of the domain which is 1.

Brown et al. (Brown, et al., 1993) used *tanh* on the hidden layer for speedup convergence, and they showed that when the input values of a two-layer network with linear output function is between –α to α, the learning is faster than when it is in the range of 0 to α, especially when using *tanh* as the activation function in three hidden layers. The main disadvantage of the *MinMax* technique is the lack of a special and unique class for each data (LeCun, et al., 1998). The function of *tanh* or hyperbolic tangent is defined based on the ration between the hyperbolic sine and the hyperbolic cosine functions, as shown in Equation (3.9).

$$tanh(X_{ij}) = {sinh(X_{ij})}/{cosh(X_{ij})} = {e^{X_{ij}} - e^{-X_{ij}}}/{e^{X_{ij}} + e^{-X_{ij}}} \quad (3.9)$$

Where, $sin(X_{ij}) = {e^{iX_{ij}} - e^{-iX_{ij}}}/{2i}$ and $cos(X_{ij}) = {e^{iX_{ij}} + e^{-iX_{ij}}}/{2i}$ are the Equations of the hyperbole of sine and the hyperbole of cosine, based on e exponent value.

Data reduction is used for reducing the volume of data representation and transform the data to the same analytical results. Data cube aggregation, data compression, numerosity reduction, discretization and concept hierarchy generation, and dimensional data reduction are some techniques of the data reduction.

Data Cube Aggregation: Aggregation of operations and applying them to the data in the construction of a data cube. A data cube is a three or higher dimensional matrix of values.

Data compression: Encoding the data and reducing the dataset size. If the original data can be reconstructed from the compressed data without any loss of information, we can reconstruct an approximation of the original data (Demuth, et al., 2008; Erhan, et al., 2010; Gabrys, 2012; Han & Kamber, 2011; Larochelle, et al., 2009).

Numerosity Reduction: To replace the data with alternative, smaller data representations such as parametric models or non parametric models such as clustering, histograms, and sampling (Demuth, et al., 2008; Erhan, et al., 2010; Gabrys, 2012; Han & Kamber, 2011; Larochelle, et al., 2009).

Discretization and Concept Hierarchy Generation: To divide the range of attributes from continuous to intervals. Some classification algorithms work on categorical (non-numerical) attributes. Data discretization is useful for automatic generation of concept hierarchies based on the number of distinct values of each attribute. The attributes with the most distinct values is placed at the lowest level of the hierarchy (Demuth, et al., 2008; Erhan, et al., 2010; Gabrys, 2012; Han & Kamber, 2011; Larochelle, et al., 2009).

Dimension Data Reduction: If the dimension of the input vectors is large, the components of the vectors are highly correlated (redundant). In this situation, it would be useful to reduce the dimension of the input vectors. The assumption is that most information on classification of higher dimensional matrix has a large variety. The main disadvantage of dimensionality reduction techniques is missing input values (DeMers & Cottrell, 1993; Furao, et al., 2007; Van der Maaten, et al., 2009). Dimension data reduction method projects high dimensional data matrix to lower dimensional sub-matrix for effective data processing at high speed (Han & Kamber, 2011; Van der Maaten, et al., 2009). There are two types of supervised and unsupervised dimension reduction methods. The type of reduction is based on the relationship of dimensionality reduction to the dataset itself or on an integrated known feature. In supervised dimension reduction, a suitable sub-matrix is selected based on their scores, prediction accuracy, selection of the number of the necessary attributes, and computation of weights with a supervised classification method. Unsupervised dimension reduction maps a high dimension matrix to a new lower dimensional matrix of data points. The dimension reduction techniques are also divided into the linear and nonlinear methods based on the consideration of the various relations between parameters. In the real world, data is non-linear; hence only nonlinear techniques are able to handle them. Linear techniques consider a linear subset of higher dimensional space, while nonlinear techniques assume more complex subset of higher dimensional space (DeMers & Cottrell, 1993; Furao, et al., 2007; Han & Kamber, 2011; Van der Maaten, et al., 2009).

Principal component analysis (PCA) (Jolliffe, 1986) is a classical multivariate data analysis method that is useful in linear feature extraction and data compression. The PCA technique has three effects (Lindsay et al., 2002; Özbay, et al., 2006): it orthogonalizes the components of input vectors so they are uncorrelated with each other; it orders the resulting orthogonal components (principal components) so that those with larger variation come first; it eliminates the components that contribute least to the variation in the dataset. Next, the input vectors are normalized and the zero mean and unity variance are computed before the mean and standard deviation method is employed (Demuth, et al., 2008). The basic assumption is that most information in classification of high dimensional matrix has a large variation. The PCA computation maximizes the variance in a process environment for the standardized linear process. p-dimensional dataset X is given and the m principal axes $T_1, T_2, \dots, T_m$, where $1 \leq m \leq p$. The axes within variance are maximized in the process environment. $T_1, T_2, \dots, T_m$ lead to the eigenvectors of the sample covariance matrix $S = {}^1\!/_N \sum_{i=1}^{n} (X_i - \mu)^T (X_i - \mu)$ where $X_i \epsilon X$, μ is the sample mean and N is the number of samples, so that: $ST_i = \gamma_i T_i$; $i = 1, \dots, m$. γ_i is the i_{th} largest eigenvalue of S . The m principal components of the given observation vector X are given by $y = [y_1, y_2, \dots, y_m] = [T_1^T, T_2^T, \dots, T_m^T] = T_X^T$. The m principal components of X are uncorrelated during the processing. In multi-class problems, variations of data are determined on a global basis. This means that the principal axes are derived from a global covariance matrix: $\bar{S} = {}^1\!/_N \sum_{i=1}^{k} \sum_{i=1}^{Nj} (X_i - \bar{\mu})^T (X_i - \bar{\mu})$ where $\bar{\mu}$ is the global mean of all the samples, K is the number of classes, N_j is the number of samples in class j; $N = \sum_{j=1}^{k} N_j$ where X_{ij} represents the i_{th} observation from class j. The principal axes $T_1, T_2, \dots, T_m$ are therefore the m leading eigenvectors of $\bar{S}T = \gamma_i T_i$; $i = 1, \dots, m$ where $\overline{\gamma_i}$ is the i_{th} largest eigenvalue of $\bar{S}$. Each input data vector can be represented by its principal component vector with dimensionality m. The algorithm of the PCA is shown in APPENDIX A

Figure 3-1A**.** The advantages of PCA are in three-fold. First, it is an optimal linear scheme for compressing a set of high dimensional vectors into a set of lower dimensional vectors and reconstructing the original set. PCA is often used for reducing the dimensional input values into two or three dimensions. Second, dimensionality reduction is possible through computation of the highest variance in the components of the input feature vector, without performing any transformation on the input environment. The input values are analysed within their own input environment, and the transformation results are deterministic as well as independent of the initial conditions. Third, compression and decompression are easy to perform (Labib & Vemuri, 2006) because the calculation is based on matrix multiplication. The time complexity of the PCA is $O(p^2n)+(p^3)$, where p is the number of data samples and n is the number of attributes (Lindsay, et al., 2002; Özbay, et al., 2006).

- **Pre-training Techniques of Feedforward Neural Network**

Initialization of weights is the first critical step in the training process of the FFNN models and it is often done at random. Training can be accelerated through a good initialization of weights during pre-training (DeMers & Cottrell, 1993; Kamyshanska & Memisevic, 2013; Van der Maaten, et al., 2009). The number of epochs in the training process depends on the initial weights. The FFNN may not obtain an acceptable range of results and may halt during training. The processing time depends on the initial values of the weights and biases, the learning rate, as well as the network topology (Bengio, 2013; Galhardas, et al., 2001; Kamyshanska & Memisevic, 2013; Larochelle, et al., 2012; Zhang, et al., 2004). In the following sections, the latest main methods of pre-training are discussed.

Min and Max: Zhang et al. (Zhang, et al., 2004) and Fernández-Redondo and Hernández-Espinosa (Fernández-Redondo & Hernandez-Espinosa, 2001) discussed several initial weight initialization techniques. In the Min and Max pre-training technique, weight is initialized based on a random value from a special range, which is a critical issue to research.

Kim and Ra (Kim & Ra, 1991) introduced a minimum bound for initialization of weights by using the Equation (3.10):

$$\left({}^{L}\!/_{N\ input}\right)^{1/2} = |W_i| \tag{3.10}$$

L is the learning step. Fernández-Redondo and Hernández-Espinosa [13] proposed an upper bound of 0.1 plus a lower bound.

Keeni et al. (Keeni, et al., 1999) introduced the idea for initializing the weight range within the domain of [−0.77; 0.77] with fixed variance of 0.2. The experiment achieved the best mean performance for multi-layer perceptrons with only one hidden layer.

Currently, the Min and Max pre-training technique using a BPN has initial random weights in the range of [-0.05, 0.05] (Fernández-Redondo & Hernandez-Espinosa, 2001). Also, Fernández-Redondo and Hernández-Espinosa presented a BPN with the weights initialized using the technique of Yoon and his cooperators (Yoon, Bae, & Min, 1995) in two different ranges of [0, 0.5] and [0.5, 1] (Fernández-Redondo & Hernandez-Espinosa, 2001). Moreover, Fernández-Redondo and Hernández-Espinosa presented a BPN with the weights initialized using the technique of Kim and Ra (Kim & Ra, 1991) with an upper bound of 0.1 plus a lower bound (Fernández-Redondo & Hernandez-Espinosa, 2001). The disadvantage of the Min and Max pr-training technique is in the need to initialize random values which create critical problems during training.

Statistically Controlled Activation Weight Initialization: Drago and Ridella (Drago & Ridella, 1992) introduced a method called statistically controlled activation weight initialization (SCAWI). The formula in Equation (3.11) was designed for initializing the weights W, whereby V is the mean square value of the input and r_{ij} is a random number uniformly distributed in the range [-1, +1].

$$W_{ij}^{input} = {}^{1.3}\Big/_{(1 + N\ input. V^2)^{1/2}} . r_{ij} \tag{3.11}$$

Fernandez-Redondo and Hernandez-Espinosa (Fernandez-Redondo & Hernandez-Espinosa, 2000) and Funahash (Funahashi, 1989) improved this method in Equation (3.12) seeking better result:

$$W_{ij}^{hidden} = {}^{1.3}\Big/_{(1 + 0.3\ .N\ hidden)^{1/2}} . r_{ij} \tag{3.12}$$

However, one of the disadvantages of the SCAWI is that it uses random numbers to feed the formula similar to the Min and Max pre-training technique, which is its disadvantage.

Delta Pre-training (DPT): Li et al., (Li, et al., 1993) described the delta pre-training (DPT) technique as a different weights initialization technique. The core of the DPT is in using the Delta rule instead of using random numbers, after this phase, the FFNN model training process is carried out to complete the network training. First, the multi-layer model is partitioned at the hidden layer into two simple perceptrons models. The weights are initialized with zero values by using the Delta rule in two perceptrons. The disadvantage of this technique is the initialization of the zero value that is not based on computing real weights.

Shimodaira Technique: Shimodaira (Shimodaira, 1994) introduced onepre-training technique based on geometrical considerations, as shown in Equations (3.13), (3.14), (3.15). W_i is the weights from n units in the lower layer to the unit number i.

$$b = |f^{-1}(1-e) - f^{-1}(-e)| \tag{3.13}$$

Where f is the transfer function and e is a parameter which was appropriated the value of 0.1 in the reference (Shimodaira, 1994).

$$W^* = {}^{b}/_{2^{1/2}}.k.n \tag{3.14}$$

Where k is the parameter. γ is a parameter and a_i is generated as a uniform random number in the range of $\gamma \leq a_i \leq \gamma$.

$$W_i = W^*.(a_i + 1) \tag{3.15}$$

Where, the W_0 is zero. Eqations (3.14) and (3.15) repeat n times to calculate the n weights.The disadvantage of this technique is in the initialization weights to zero, that is not based on the computation of the real weights.

Multilayer Auto-encoder Networks: Multilayer encoder is the latest preprocessing technique in FFNN model, which trains an odd number of hidden layers (DeMers & Cottrell, 1993; Hinton & Salakhutdinov, 2006; Kamyshanska & Memisevic, 2013; Van der Maaten, et al., 2009). Generally the BPN is used in auto encoders. The FFNN trains to minimize the mean squared error between the input and the output by using the sigmoid function. A high-dimensional matrix may be reduced into a low-dimensional matrix through the extraction of node values in the middle hidden layer. In addition, the auto-encoder/auto-Associative neural networks are neural networks that are trained to recall their inputs. When the neural network uses the linear neuron and activation functions, the auto-encoder processes are similar to the PCA. The Sigmoid activation function allows the auto-encoder network to train a nonlinear mapping between the high-dimensional and low-dimensional data matrix. After the pre-training phase, the model is “unfolded” to encode and decode the initial weights (Bengio, 2013; Goroshin & LeCun, 2013; Kamyshanska & Memisevic, 2013). The BPN advances the global fine-tuning phase through the auto-encoder to fine-tune the weights for optimization. In a high number of multi-layer auto-encoders connections, the BPN is considerably slower. The restricted boltzmann machines (RBM) (Smolensky, 1986) are able to train efficiently using an unsupervised learning procedure. The RBM is a two-layer network with visual and hidden nodes, and is suitable for an ensemble of binary vectors (i.e. images). The single layer of hidden units in the RBM is not connected to each other and have undirected, symmetrical connections to a layer of visible units. All visible and hidden unit configurations have energy (Bengio, 2013; Larochelle, et al., 2012). The auto-encoder network is fine-tuned by a supervised model of the BPN in a standard way. The main disadvantages of this method are the use of random numbers for weight initialization and the high number of multi-layer auto-encoders connections in the training process, resulting in slow performance.

Weight Linear Analysis (WLA): The WLA (Asadi, et al., 2009) technique in the FFNN is incorporated as a combination of data preprocessing and a specific pre-training technique for simplifying the training process. The WLA is a technique for reducing the training time and increasing the accuracy of the feedforward neural network (FFNN). The WLA has two phases. During the first phase, the WLA considers data preprocessing through vertical evaluation of the input data matrix for generating normalized input values. The output of the first phase includes normalized input values, and is used to compute the weights of attributes. The WLA recognizes high deviations of data values from the global mean of the matrix similar to the PCA. The high deviation causes more score for the data value. The FFNN, uses the output of the WLA which includes normalized values and weights to classify the dataset. The time complexity of the WLA is *O(pn)*, where p is the number of attributes and *n* is the number of instances.

3.3 Unsupervised Feedforward Neural Network Clustering

As mentioned in the last Section, feedforward neural network processes data by only one forward direction from the input layer to the output layer without any feedback (Demuth, et al., 2008; Kohonen, 1997; Van der Maaten, et al., 2009). Generally, UFFNN clustering has several advantages such as its inherent distributed parallel processing architectures, the abilities to adjust the interconnection weights to learn and describe suitable clusters, process vector quantization prototypes and distribute similar data without class labels to describe the clusters. It can also control noisy data and cluster data for which they have not been trained, while learning the types of input values based on their weights and properties (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Hegland, 2003; Jain, 2010; Rougier & Boniface, 2011).

UFFNN clustering methods currently often use Hebbian learning (Hebb, 1949), competitive learning, or the competitive Hebbian learning (Chakraborty, 2010; Hebooul, et al., 2015; Liu, et al., 2013; Martinetz, 1993), as discussed in Section 3.5. Figure 3-13 is the topology of an unsupervised competitive learning neural network (Du, 2010; S. Haykin & Network, 2004).

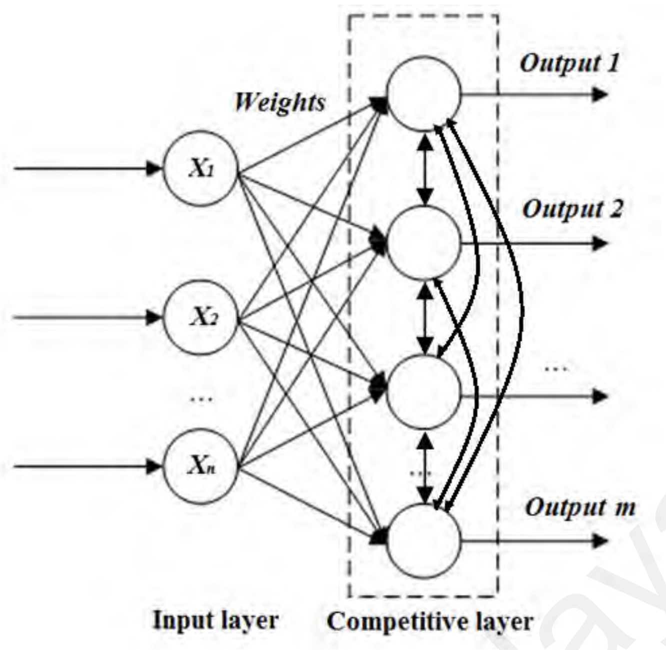


Figure 3-13: A sample topology of the competitive clustering

The UFFNN clustering methods with competitive learning train the nearest weight vector to the input vector as the winner node by computing a distance such as the Euclidean. The similarities of Hebbian learning and competitive learning include unsupervised learning without an error signal, and is strongly related to biological systems. However, in competitive learning, only one output must be active; that is, only the weights of the winner, which is very close to the input vector, are updated at each epoch, and for weights updating, it is necessary to consider the learning rate and input data from the input layer. Conversely, in Hebbian learning, no constraint is enforced by neighbouring nodes, all weights are updated at each epoch and for updating weights, it is necessary to consider the learning rate of input data from the input layer and output data. In the case of the competitive Hebbian learning, the neural network method shares some properties of both competitive learning and Hebbian learning (Fritzke, 1997; Hebooul, et al., 2015; Liu, et al., 2013; Martinetz, et al., 1993; McClelland, et al., 1999). The growing neural gas (GNG) (Fritzke, 1995) method and the evolving self-organizing map (ESOM) (Deng & Kasabov, 2003) are examples which use the competitive Hebbian learning where in each cycle of training the connection between the winner node and the second nearest node is created or updated. Competitive learning can apply vector quantization (VQ) (Linde, et al., 1980) during clustering. Typically VQ, *K-means* (MacQueen, 1967) and some unsupervised feedforward neural network clustering methods such as Kohonen's self-organizing map (SOM) (Kohonen, 1982), neural gas (NG) (Martinetz, et al., 1993) and growing neural gas (GNG) (Fritzke, 1995) are considered as the fundamental patterns in the current ODUFFNN clustering methods in stationary and non-stationary environments. Linde et al. (Linde, et al., 1980) introduced an algorithm for the VQ design to gain a suitable code book of weights for data clustering. The VQ is based on probability density functions through the distribution of vectors of the weights, and it is often used for data compression. The VQ divides a huge set of the data (vectors) into clusters, in which, each cluster is represented by its centroid node, as in the *K-means* and some other clustering algorithms. The VQ is powerful for using in large and high-dimensional data. Since the data points are represented by the index of their closest centroid, commonly, clustered data are more accurate. *K-means* (MacQueen, 1967) is a partitioning clustering method by using centroid-based technique similar to the VQ. The main problem of partitioning methods is the definition of a special number of clusters and the initialization of the steps before the clustering tasks (Andrews & Fox, 2007; Mercer, 2003). Although, *K-means* clustering is not efficient in large datasets, it can be efficient if the initialization of the steps is well defined (Jain, 2010; Mercer, 2003).

In the following sections, we will illustrate some unsupervised feedforward neural network clustering models such the SOM and GNG as foundation patterns which are used by current online dynamic unsupervised feedforward neural network clustering and are related to the proposed method.

3.3.1 Self-organization Map

The self-organization map (SOM) (Kohonen, 1982) maps multidimensional data onto lower dimensional subspaces where the geometric relationships between the data points indicate their similarity. Figure 3-14 shows a simple topology of the SOM.

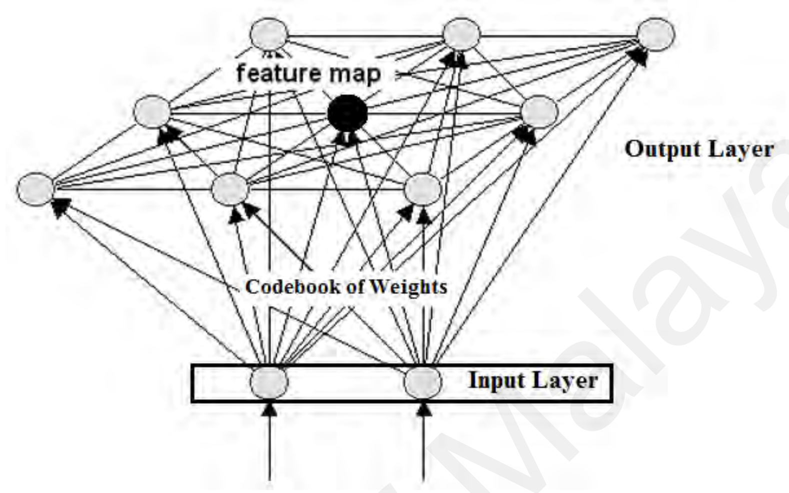


Figure 3-14: A simple topology of the SOM

The SOM is an unsupervised competitive learning neural network and generates subspaces. The algorithm of the SOM is shown in APPENDIX A, Figure 3-2A. The algorithm of the SOM contains several stages. The code book of weights is often initialized by using small random values. There are other techniques to initialize the code book of weights in the SOM, such as initializing by using the values from two largest principal components eigenvector subsets of the input data, but this will incur a long CPU time usage and high memory capacity. Then, one input data vector is chosen from the training set. The most similar weight to the input vector in the code book is distinguished as the winner node and the best matching unit (*BMU*). Two other neighbours from the code book are appointed as second and third winners. Figure 3-15 shows one stage of processing in the SOM clustering. In order to increase the similarity of the winning node and neighbour nodes to the input data, their weights are updated. These iterations are repeated until the final *BMU* is obtained. Finally, the data nodes are clustered based on the *BMU* (Jolliffe, 1986; Kaski, 2009; Kohonen, 1997).

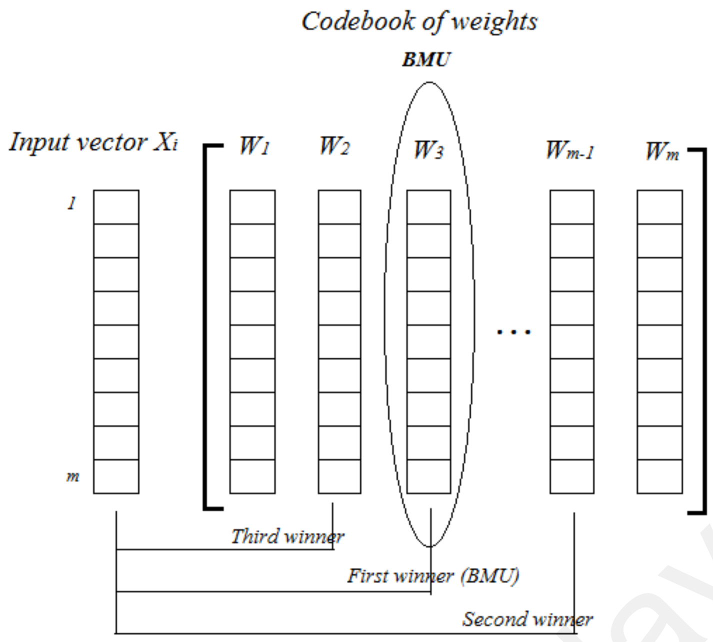


Figure 3-15: One stage of the SOM process as an example

The fundamental focal point of utilizing the SOM can be simply interpreted. The SOM can cluster large datasets in a short duration of time. The model is not parameter free to control clustering tasks and commonly these parameters are selected as random numbers between [0,1]. The time complexity and memory complexity of the SOM are $O(c.n.m^2)$ and $O(c.n.m^2.s_m)$ where n, m and s_m are the number of nodes, attributes and size of the attribute, respectively (Jolliffe, 1986; Kaski, 2009; Kohonen, 1997). Incomplete and noisy data as unclean data affect the accuracy of clustering (Germano, 1999; Honkela, 1998; Kohonen, 2000). The weight vectors are dependent on the data that can cluster and recognize inputs, but the initialization of the weights is often random (Kohonen, 1997; Rougier & Boniface, 2011). The SOM creates fix code book of the weights. Therefore, the SOM clustering methods is not suitable for lifelong incremental learning (Kulkarni & Mulay, 2013; T. Wang, et al., 2013).

Some literatures, devoted to improving the clustering methods by the technique of using constraints such as class labels (Hebooul, et al., 2015; Prudent & Ennaji, 2005). The constraints of class labels are based on the knowledge of experts and the user guide as partial supervision for better controlling the tasks of clustering and desired results

(Hebooul, et al., 2015; Kamiya, et al., 2007; Prudent & Ennaji, 2005; Shen, et al., 2011). The semi-SOM (Herrmann & Ultsch, 2007) which was improved based on the SOM clustering method, is an example in this area. In order to improve the SOM clustering method, the users manage and correct the number of clusters and density of each cluster by inserting and deleting the data nodes and clusters. After clustering, the models assigned class label to the winning node and consequently assigned the same class labels to its neighbour nodes in its cluster. Each cluster must have a unique class label, if the data nodes of a cluster have different class labels, the cluster can be divided into different sub-clusters. However, assigning the class labels to the data nodes between the clusters can be somewhat vague. The judgment of users can be wrong or they may make mistakes during the insertion, deletion or finding the link between nodes and assigning the class label to each disjoint sub-cluster (Hebooul, et al., 2015; Kamiya, et al., 2007; Prudent & Ennaji, 2005; Shen, et al., 2011).

3.3.2 Growing Neural Gas

Neural gas (NG) (Martinetz, et al., 1993) and growing neural gas (GNG) (Fritzke, 1995) methods are based on the competitive Hebbian learning where in each cycle of training the connection between winner node and the second nearest node is created or updated. Neural gas (NG) (Martinetz, et al., 1993) is based on VQ and data compression. NG dynamically partitions the network of the data nodes like gas and initializes the number of clusters. The vectors of weights are initialized randomly. NG clustering is faster and more accurate, but it cannot control the network of data nodes by deleting or adding a node dynamically during clustering. The GNG clustering method is developed based on the structure of the NG method and it is able to dynamically create and delete nodes with limited utility parameters. The algorithm of the GNG contains several stages too. Firstly, two random nodes from the input data are selected and the network competition starts for the highest similarity to the input pattern. Then, one input data vector is chosen from the training set. First and second nodes closest to the input data vector are found. Consequently, local errors are computed during the learning to determine where to add new nodes. A new node close to the node with the highest accumulated error is added. During learning, the related data nodes are classified based on the highest similarities within clusters. This process is repeated by chosen a new input vector from the training set, until the end of learning (Jolliffe, 1986; Kaski, 2009; Kohonen, 1997).

The time complexity and memory complexity of the NG and the GNG are *O(c.n^2.m)* and *O(c.n^2.m.s$_m$)* based on *n, m* and *s$_m$* the number of nodes, attributes and size of the attribute, respectively. Figure 3-3A in APPENDIX A shows the algorithm of the GNG. The disadvantages of the GNG are, that the model should determine the thresholds of clusters and maximum size of the network of clusters in order to get the maximum number of the clusters and the density of each cluster (Furao, et al., 2007; Hamker, 2001; Hebooul, et al., 2015; Liu, et al., 2013). Therefore, the NG and GNG clustering methods are not suitable for lifelong incremental learning. Figure 3-16 shows clustering of Iris dataset by using the GNG clustering method (Costa & Oliveira, 2007).

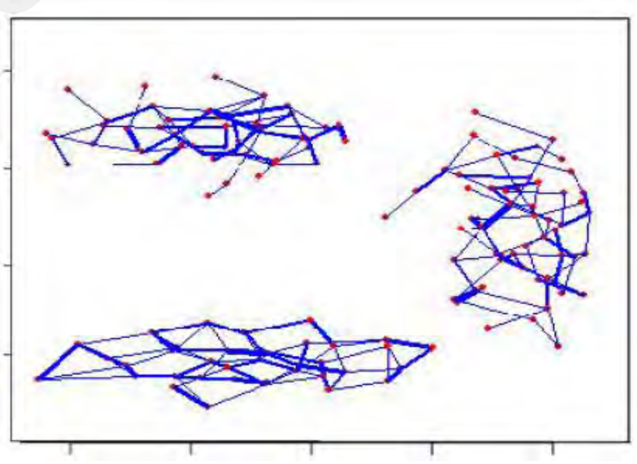


Figure 3-16: Classification of Iris dataset by using the GNG (Costa & Oliveira, 2007)

3.4 The Time Complexity and Memory Complexity of the Unsupervised Feedforward Neural Network Clustering and Some Related Clustering Methods

In this section, the time complexities and memory complexities of the mentioned UFFNN clustering methods and some related clustering methods of this chapter are discussed, as shown in Table 3-1:

Table 3-1: The time complexities and memory complexities of some UFFNN clustering and related clustering methods

Methods	Time complexity	Memory complexity
K-means	$O(c.k.n.m)$	$O((n+k).m.s_m)$
NG	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
GNG	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
SOM	$O(c.n.m^2)$	$O(c.n.m^2.s_m)$
$*$ *note*: n is the number of data points, m is the number of attributes, K the number of clusters, c is the number of iterations, S_m is size of each attribute.		

3.5 Online Dynamic Unsupervised Feedforward Neural Network Clustering

Many real world data processing environments, such as credit card transactions, intelligent multi-agent systems, and medical informatics, demand intelligent computational models to learn online continuous data that are updated frequently. The problem of such environments are collection, storage, search, transfer, visualization and analysis of massive sample size and high dimensional data (Bouchachia, et al., 2007; Hebboul, et al., 2011; Hsu, 2003; Kasabov, 1998; Rougier & Boniface, 2011). Clustering method is applied to recognize the pattern and discover the knowledge of the data, in order to group them (Bouchachia, et al., 2007). As the data environment is non-stationary, online dynamic UFFNN (ODUFFNN) clustering methods should have lifelong (online) and incremental learning. The ODUFFNN clustering method should be able to control noisy data, adapt its algorithm and adjust itself in a flexible way to new conditions of the environment over time dynamically for processing of both data and knowledge. The ODUFFNN should accommodate and prune the data and rules without destroying old knowledge, should learn a number of clusters and density of each cluster without predetermining the rules. In addition, the ODUFFNN method must control time, memory space and accuracy efficiently (Bouchachia, et al., 2007; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Schaal & Atkeson, 1998). Incremental learning refers to the ability of training in repetition by adding or deleting the data nodes in lifelong learning without destroying outdated prototype patterns (Furao, et al., 2007; Rougier & Boniface, 2011; Schaal & Atkeson, 1998). In this section, we consider some efficient ODUFFNN clustering methods such as evolving self-organizing map (ESOM) (Deng & Kasabov, 2003), self-organizing incremental neural network (SOINN) (Furao & Hasegawa, 2006) and its enhanced version (ESOINN) (Furao, et al., 2007), dynamic self-organizing map (DSOM) (Rougier & Boniface, 2011) and incremental growing with neural gas utility parameter (IGNGU) (Hebboul, et al., 2011). Current ODUFFNN clustering methods often use competitive Hebbian learning as used in the ESOM, or competitive learning as used in the DSOM through online learning.

In order to recognize and find the limitations and problems of existing ODUFFNN clustering methods, and their reasons, we select some strong related methods in the scope of this research, and then, we review and analyse the details of their topologies, algorithms and behaviours.

3.5.1 Evolving Self-organizing Map

Evolving self-organizing map (ESOM) (Deng & Kasabov, 2003) begins without the data nodes in its cluster network; and during training, the network earns non stationary data and updates itself with online entry, and if it is necessary, it creates new nodes. Each node has a special weight vector. The strong neighbourhood relation is determined by the distance between connected nodes. If the distance between connected data nodes is too big, it creates a weak threshold and the connection can be pruned. Also, the ESOM is an incremental network quite similar to the GNG that dynamically measures the distance of the winner to the data. The algorithm of the ESOM is shown in APPENDIX A, Figure 3-4A. During the ESOM clustering, one input data vector is chosen from the training set. If the network of clusters is empty or none of the existing matching nodes with the input vector within a distance threshold, the model creates a new node in the network representing the input, and connects it to the first two best-matching nodes. Consequently, the model searches for a winner node among the existing nodes. If the minimum distance is smaller than the distance threshold, the ESOM updates the winner node and its neighbours and their connections, otherwise, inserts it as a new node in the network. After several steps of learning time, the model prunes the weakest connection. The model continues to learn the new nodes until the end of training. The ESOM clustering process starts without any data node, and with the initialization of the number of clusters and the parameter set, such as a parameter of a distance threshold that controls the creation of the node, a parameter of controlling the spread of the neighbourhood, a parameter of a small constant learning rate which is usually 0.05, and a parameter of the steps of learning time. The ESOM is a model based clustering and is able to create the sub-clusters of the data points based on the normal distribution and the VQ. The ESOM is sensitive to noise nodes and prunes weak connections and isolated nodes based on Hebbian learning (Hebb, 1949), by computing and considering the strength of connections between the winner (or the newly created node) and its neighbours. After the entrance of each online input value, the ESOM checks all nodes and their weights of neighbour clusters for inserting or updating the nodes of the network. This may take a long training time and large memory usage; however, the clustering is carried out in just one epoch. The ESOM is unable to control the growth of the number of clusters and the size of the network. The ESOM is sensitive to a first entrance of the input data, and creates the structure of the cluster network based on the first input data, which shows poor adaptability to the input data vector. Initialization of parameters for training is based on trial and error and after several performances of the clustering model and checking the results, the best parameters are recognized. However, the model is not scalable and has different results for each performance (Furao & Hasegawa, 2006; Hebooul, et al., 2015; Liu, et al., 2013; Rougier & Boniface, 2011). The time complexity of the ESOM model is $O(n^2.m)$ and the memory complexity of the ESOM is $O(n^2.m.\ s_m)$. The parameters n, m, s_m are the number of nodes, attributes and size of each attribute respectively. The number of epochs is not considered in computing the time complexity and memory complexity because the clustering process is carried out just in one epoch.

As discussed in Section 3.3.1, some literature are devoted to improve the clustering methods by using class labels (Prudent & Ennaji, 2005). Assigning the class labels are based on the knowledge of experts and the user guide as partial supervision for better controlling the tasks of clustering and desired results (Hebooul, et al., 2015; Kamiya, et al., 2007; Prudent & Ennaji, 2005; Shen, et al., 2011). The semi-ESOM (Deng & Kasabov, 2003) which was improved based on the ESOM clustering method, is another example in this area. After clustering, the semi-ESOM model assigned class label to the winning node and consequently assigned the same class labels to its neighbour nodes in its cluster. Each cluster must have a unique class label, if the data nodes of a cluster have different class labels, the cluster can be divided into different sub-clusters.

3.5.2 Enhanced Self-organizing Incremental Neural Network for Online Unsupervised Learning

Self-organizing incremental neural network (SOINN) (Furao & Hasegawa, 2006) and its enhanced SOINN (ESOINN) (Furao, et al., 2007) are also based on an incremental structure where the first version uses a two layer network while the enhanced version uses a single layer network. The SOINN is useful to process online non-stationary data, report a suitable number of the classes, and represent the topological structure of the input probability density. The SOINN learns a necessary number of the data nodes, uses fewer nodes than the GNG, and obtains better results than the GNG. The SOINN has a two layer network that makes it unsuitable for using as an ODUFFNN clustering method. This is because it is unable to store all details of the data and loses them from one layer to another layer during training. Another disadvantage of the SOINN is that, it is able to separate clusters with very low density overlap. If there is a high density overlap between clusters, it cannot work appropriately, and the clusters will link together to form one cluster. Semi-SOINN (Shen, et al., 2011) was adapted from the SOINN (Furao & Hasegawa, 2006) in order to improve the accuracy of the clustering result. However, semi-SOINN inherits the problems and limitations of the SOINN. Furao & Hasegawa improved the structure of the SOINN to one layer and applied class labels for semi supervised feedforward neural network clustering. After clustering, the semi-SIONN assigns a class label to the winning node and consequently assigns the same class labels to its neighbour nodes in its cluster. Each cluster must have a unique class label. If the data nodes of a cluster have different class labels, the cluster could be divided into different sub-clusters. However, assigning the class labels to the data nodes between the clusters could be somewhat vague. The semi-SOINN model inherits some disadvantages from the SOINN method, such as, very outdated learning information is forgotten; new learned patterns are lost and only the old input pattern is represented and the topological structure of the incremental online data cannot be well represented; initialization of the parameters for training is based on trial and error; and there is relearning in several epochs (Han & Kamber, 2011; Kamiya, et al., 2007; Kantardzic, 2011; Shen, et al., 2011).

As a solution to the problems of the SOINN, Furao et al. (2007) proposed the ESOINN for online unsupervised learning. In the ESOINN method, it is necessary that very out of date learning information be forgotten because the method is unable to store, retrieve and manage very old data, and hence, this is a disadvantage of the ESOINN method (Hebboul, et al., 2011; Rougier & Boniface, 2011). The algorithm of the ESOINN is shown in APPENDIX A, Figure 3-5A. The ESOINN clustering algorithm contains several stages (Furao, et al., 2007). One input data vector is chosen from the training set. Then, the model finds the winner and the second winner close to the input vector node. If the distance between the new input vector and the winner or second winner is less than the similar distance threshold, the new node is added, and if it does not have the connection, the model creates a connection between the winner and second winner nodes. The model continues to learn other new nodes. During learningn, the density, the weight of the winner and neighbour, and the subclass label of the data nodes are updated depended on input data; and old connections, overlap, and noise nodes are deleted.

In the ESOINN, the input data vectors are not stored during learning. If the distance between the new data and the winner or second winner is greater than the similar distance threshold, the network will grow. If the distance between the new input vector and the winner or second winner is less than the similar threshold, the new data will be learned without changing the network. Therefore, learning of the new input does not destroy the last learned knowledge. The disadvantages of the ESOINN method are: out of date learning information is forgotten and new learned patterns are lost. Therefore, the topological structure of the incremental online data cannot be well represented. Furthermore, the initialization of the parameters for training is based on trial and error; and there is relearning in several epochs (Hebboul, et al., 2011; Hebooul, et al., 2015; Liu, et al., 2013). The time complexity of the ESOINN is *$O(c.n_2.m)$*. The memory complexity is *$O(c.n_2.m.s_m)$*. The parameters of *n, m* and *s_m* are the number of nodes, attributes and size of each attribute respectively, and the parameter *c* is the number of epochs.

3.5.3 Dynamic Self-organizing Map

Dynamic self-organizing map (DSOM) (Rougier & Boniface, 2011) is based on the SOM but is suitable for incremental learning since it is not dependent on the time, it has a special formula for updating the weights and the flexibility property. In order to update weights of the neighbourhood nodes as shown in Equation (3.16), the time dependency is removed, and a special parameter of flexibility is considered:

$$W_{new} = W_i + \varepsilon \, \| X - W_i \|_R \, \alpha(\gamma) . (X - W_i) \tag{3.16}$$

Where ε is learning rate, vector X is a subset of R which are sequentially presented in the map with respect to the probability density function. In Equation (3.17) $\alpha(\gamma)$ is the neighbourhood function in a different form:

$$\alpha(\gamma) = e^{-\left({}^{1}/_{\gamma^2}\right)\|p_i - p_{winner}\|^2 / \|X - W_{winner}\|^2 R} \tag{33.17}$$

Where position p is used for the vector-projection purposes and p_i refers to the node *i* in the position or the weight codeword of node *i*. The distance of $\|p_i - p_{winner}\|$ represents the distance between the weight of node *i* and the winner. γ is the flexibility parameter and the optimal γ must be learned by using trial and error. If γ is too high, the DSOM does not converge and if it is too low, it is not sensitive to the relation between neighbour nodes. If $X= W_{winner}$, then $\alpha(\gamma)=0$. In the DSOM, if the weight vector is similar to the input data, other neighbour nodes do not need to learn more and the winner can map the data (Rougier & Boniface, 2011). The DSOM method is sensitive to parameters and weights which are initialized randomly, the initialization of some parameters for controlling tasks is done by trial and error, and relearning occurs over several epochs (Hebboul, et al., 2011; Hebooul, et al., 2015; Liu, et al., 2013). The time complexity of the DSOM is $O(c.n.m^2)$ and the memory complexity is $O(c.n.m^2.s_m)$ based on n, m and s_m the number of nodes, attributes and size of the attribute.

3.5.4 Incremental Growing with Neural Gas Utility Parameter

Prudent and Abdel Ennaji (Prudent & Ennaji, 2000) introduced an incremental growing with neural gas (IGNG), based on the GNG with predefined thresholds, and it was not suitable for online dynamic clustering. Moreover, the IGNG could not control the noise and density overlapping (Hebboul, et al., 2011). Therefore, Hebboul and Hacini et al. (2011) proposed incremental growing with neural gas utility parameter (IGNGU) (Hebboul, et al., 2011) as the online incremental unsupervised clustering based on the structure of the GNG model and Hebbian (Hebb, 1949) learning. The IGNGU was proposed without any restraint and control on the network structure associated with competitive learning method of the Hebbian. The algorithm of the IGNGU is shown in APPENDIX A, Figure 3-6A. The structure of the IGNGU contains two layers of learning (Hebboul, et al., 2011): The first layer of the IGNGU creates a suitable structure of the clusters of input data nodes with lower noise data, and computes the distance threshold. During training, the first layer considers a few data at a time and then moves to the second layer to process active nodes and disable all nodes and their parameters in the first layer; and then immediately trains some other data. The second layer uses and trains the output of the first layer in parallel and creates the final structure of clusters. The model learns the disable data nodes by the first layer again.

Some disadvantages of the IGNGU are: parameters are determined experimentally by trial and error ; and some data and their connections are lost for high speed clustering; memory usage is high for maintenance and allocation of the information in the first layer and the second layer (Heboоul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013; Wang, et al., 2013). The time complexity of the IGNGU is $O(c.n^2.m)$ and the memory complexity is $O(c.n^2.m.s_m)$ based on n, m and s_m the number of the nodes, the attributes and size of each attribute.

3.5.5 An Enhancing Dynamic Self-organizing Map

Wang et al. (Wang, et al., 2013) introduced an enhancing dynamic self-organizing map (EDSOM) for data clustering incrementally, based on the SOM and GNG. The EDSOM starts with four connected data nodes, and initialization of random weights. The algorithm of the EDSOM is shown in APPENDIX A, Figure 3-7A. The EDSOM starts with four connected data nodes, and initializes the weight vectors of the four data nodes with random values. Then, one input data vector is chosen from the training set, and subsequently, finds the winner node closer to the input vector node, using Euclidean distance. If the distance between the new input vector and the winner is less than the similar distance threshold, inserts it, and updates the winner and its neighbours nodes. If else, the EDSOM creates a new node, and connects it to the winner node. The EDSOM continues to learn other new nodes, and during learning prunes weak connections and inactive nodes (Wang, et al., 2013).

The EDSOM has two bold advantages. It deletes the weak connections which represent large distances between connected data nodes, during pruning. It does not update all connections and weights, during learning. However, some disadvantages of the EDSOM are: selection four connected data nodes which affect the clustering accuracy; and losing some data nodes and their connections during clustering (Heboоul, et al., 2015; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). Initialization of parameters for training is based on trial and error, and after several performances of the clustering model and checking the results, the best parameters are recognized (Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). However, the model has different results for each performance. The time complexity of the EDSOM is $O(c.n^2.m)$ and the memory complexity is $O(c.n^2.m.s_m)$, based on n, m and s_m the number of the nodes, the attributes and size of each attribute.

3.5.6 Enhanced Incremental Growing Neural Gas

Lio et al. (Liu, et al., 2013) introduced an incremental self-organizing neural network based on enhanced competitive Hebbian learning, which is suitable for clustering of non-stationary data. The algorithm of enhanced incremental growing neural gas (HI-GNG) is shown in APPENDIX A, Figure 3-8A. The HI-GNG starts without any data node. One input data vector is chosen from the training set. If the network is empty, or the Euclidean distance between the input data node and first or second nearest data nodes are larger than the threshold, inserts two new nodes. Weight of the first nearest node (W_{n1}) is set to input data node, and weight of second nearest node (W_{n2}) is randomly generated, such as $\|W_{n1} - W_{n2}\| < \frac{\varepsilon}{10}$ where ε is distance threshold. Then, the model creates a connection between two new nodes, increases their connection-strength value, and updates the weight vectors of the first nearest node and its neighbours. During clustering, the HI-GNG model removes the weak connections and isolated data. If there is not any input data for learning, the model finishes learning and clusters all nodes (Liu, et al., 2013).

Some disadvantages of the HI-GNG are: the parameters are determined by trial and error, and some data and their connections are lost for high speed clustering (Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). The time complexity of the HI-GNG is $O(c.n^2.m)$ and the memory complexity is $O(c.n^2.m.s_m)$ based on n, m and s_m the number of the nodes, the attributes and size of each attribute.

3.6 Comparison and Discussion

In this chapter, we reviewed the related literature works and history of online dynamic unsupervised feedforward neural network. Also, we showed that the ODUFFNN were developed based on the vector quantization and *K*-means methods, and some static unsupervised feedforward neural network clustering methods as the fundamental methods, in order to have lifelong (online) and incremental learning abilities and be compatible with the changes in the continuous data. Table 3-2 shows some bold advantages of the ESOM, ESOINN, DSOM, IGNGU, EDSOM, and HI-GNG as current ODUFFNN clustering methods, according to the literature.

Table 3-2: Some bold advantages of current online dynamic unsupervised feedforward neural network clustering models

	ESOM	ESOINN	DSOM	IGNGU	EDSOM	HI-GNG
Base patterns	SOM and GNG, Hebbian	GNG	SOM	GNG and Hebbian	SOM and GNG	GNG and Hebbian
Some bold features (Advantages)	Begin without any node	Control the number and density of each cluster	Improve the formula of updating the weights	Train by two layers in parallel	Begin with four connected nodes	Begin without any node
	Update itself with online input data	Initialize code book	Elasticity or flexibility property	Control density of each cluster and size of the network	Initialize code book [m×4]	Control density of each cluster and size of the network
	The nodes with weak threshold can be pruned	Prune for controlling noise and weak thresholds		Control noise	The nodes with weak threshold can be pruned	Prune nodes with weak thresholds
		Input vectors are not stored during learning	Earning *best matching unit (BMU)*	Fast training by pruning	Earning *best matching unit (BMU)*	Use the enhanced Hebbian learning which makes the method robust to noisy data nodes
	Clustering during one epoch	New input does not destroy last learned knowledge				
Time Complexity	$O(n^2.m)$	$O(c.n^2.m)$	$O(c.n.m^2)$	$O(c.n^2.m)$	$O(c.n^2.m)$	$O(c.n^2.m)$
Memory Complexity	$O(n^2.m.s_m)$	$O(c.n^2.m.s_m)$	$O(c.n.m^2.s_m)$	$O(c.n^2.m.s_m)$	$O(c.n^2.m.s_m)$	$O(c.n^2.m.s_m)$

$*$ *note*:
n is the number of data points, *m* is the number of attributes, *K* the number of clusters, *c* is the number of iterations, S_m is size of each attribute.

For example, the DSOM is similar to the SOM and is based on competitive learning, thus it earns the property of elasticity (flexibility) by improving the formula of updating weights of the SOM. The DSOM can control the size of the network, the number of clusters and their densities through the elasticity property (Rougier & Boniface, 2011). The ESOM is based on the SOM, GNG and Hebbian. The bold properties of the ESOM includes: starting without any node, updating the clusters by online input values, and pruning weak connection. The ESOM trains during one epoch and has better CPU time usage for clustering (Deng & Kasabov, 2003). The ESOM is sensitive to noise nodes and prunes weak connections and isolated nodes based on Hebbian learning (Hebb, 1949); it resets the strength of connections between the winner (or the newly created node) and its neighbours. The ESOINN is based on the GNG with some properties such as initializing a code book, controlling noise, but it also has the disadvantages losing accuracy (Furao, et al., 2007). The IGNGU is based on the GNG and the Hebbian learning rule (Hebb, 1949) with some abilities such as parallel training in two layers; controlling noise and densities of clusters (Prudent & Ennaji, 2000). The EDSOM is based on the SOM and GNG, and its bold advantage includes pruning the weak connections (Wang, et al., 2013). The Hi-GNG is based on GNG and Hebbian, and applies the enhanced Hebbian learning, which makes the method robust to noisy data nodes (Liu, et al., 2013).

However, the ODUFFNN clustering methods also inherit some disadvantages of the UFFNN clustering methods, such as relearning, using random weights and parameters (Bouchachia, et al., 2007; Hebboul, et al., 2011; Kasabov, 1998; Schaal & Atkeson, 1998). The ODUFFNN clustering methods should train online data fast without relearning and cluster the continuous data in one pass, because there is no capacity to store complete online data, previous data and the connection of the data points in consequent steps (Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015). As shown in literature, current ODUFFNN clustering methods are not able to accommodate and adjust the data and rules without destroying old data and old knowledge (Hebooul, et al., 2015; Jain, et al., 1999; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Pavel, 2002). There is a trade-off between training time, clustering accuracy, time complexity and memory complexity of the algorithm in the ODUFFNN clustering methods, and there is surprisingly and comparatively very little works dealing with them together in one ODUFFNN clustering model (Deng & Kasabov, 2003; Furao & Hasegawa, 2006; Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Rougier & Boniface, 2011; Shen, et al., 2011).

Generally, the critical issues in the current ODUFFNN clustering methods are high training time and low accuracy of clustering, also high time complexity and high memory complexity of clustering (Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013; Rougier & Boniface, 2011). As mentioned in the details of the current ODUFFNN clustering methods in this chapter, massive data in size and the number of attributes, using random weights, distance thresholds and parameters based on trial and error for controlling clustering tasks, and relearning during several epochs are some of the reasons that create the problems.

3.7 Summary

This chapter focused on the online dynamic unsupervised feedforward neural network (ODUFFNN) clustering. We reviewed, investigated and analysed the details of some effective current methods in this scope, such as the DSOM and ESOM. The study of structures, topologies and behaviours of the existing ODUFFNN methods showed their major problems, such as high training time and low accuracy of clustering, which are caused through some common reasons, such as using random weights, distance thresholds and parameters for controlling clustering tasks, and relearning during several epochs.

CHAPTER 4: RESEARCH METHODOLOGY

4.1 Introduction

This chapter illustrates the research methodology based on Section 1.6, as shown in the framework of Figure 1-3. Therefore, this chapter briefly follows the sub-topics include an overview of the proposed methods, especially the DUFFNN clustering method, and the motivation for the design of the proposed models. It also includes the description of the methods and techniques in this study, in order to achieve the research objectives and overcome the problems, as discussed in Chapter 1. In addition, the methods used for evaluating the performance of the proposed DUFFNN clustering method are also described.

4.2 Approaches to Research

In this section, each stage of the research methodology framework of this thesis as shown in Figure 1-2 of Section 1.6, is explained.

4.2.1 Reviewing Related Literature

This research investigates and reviews the existing effective online dynamic unsupervised feedforward clustering methods through the literature experimentally. The literature review has clearly examined the features of the related methods, and identifies their limitations, some advantages and disadvantages. Furthermore, the distance measurement methods, clustering evaluation methods, and main unsupervised clustering models are discussed.

4.2.2 Problem Formulation

Based on the literature, we identified that the current ODUFFNN clustering methods generally suffer from high training time and low accuracy of clustering, also high time complexity and high memory complexity of clustering. Hence, the reasons of the mentioned problems are investigated and we recognized the reasons of the problems of the current ODUFFNN clustering methods are related to the structure and features of the data, and the topology and algorithm of the current ODUFFNN clustering method.

4.2.3 Definition of the Research Objectives

The objectives of this research are illustrated based on the formulation of the problems as mentioned in Section 1.2, as follows:

- To review current effective ODUFFNN clustering methods.
- To identify limitations and problems of current effective ODUFFNN clustering methods through the literature and practical investigations.
- To develop a dynamic unsupervised feedforward neural network (DUFFNN) clustering method that is able to:

1) Reduce the training time of clustering during one training epoch
2) Increase the accuracy of clustering
3) Reduce the time complexity of clustering
4) Reduce the memory complexity of clustering

- To evaluate the performance of the proposed DUFFNN method.
- To compare the results of the proposed DUFFNN method performance with rival methods within the scope of this research.

In order to achieve these objectives, the DUFFNN clustering method is developed and evaluated extensively in this study, as discussed in the next section.

4.2.4 Proposed Model

In order to accomplish the first objective, we reviewed, investigated and examined the current effective ODUFFNN clustering methods through the literature, and identified their limitations and problems, and formulated the problems. Then, we identified some reasons of these problems through the practical probe.

In order to accomplish the third objective and overcome the problems, we firstly introduce a real unsupervised FFNN (RUFFNN) clustering method suitable for stationary data environment with one epoch training to overcome the problems of high CPU time usage during training, low accuracy, high time complexity and high memory complexity of clustering. The RUFFNN considers a matrix of dataset as input data for clustering. During training, a non-random weights code book is learned through the input data matrix directly. A standard weight vector is extracted from the code book, and exclusive total threshold of each input instance (record of the data matrix) is calculated based on a standard weight vector. The input instances are clustered on the basis of their exclusive total thresholds.

In order to improve the results of clustering based on the literature, the RUFFNN model is improved to real semi-supervised FFNN (RSFFNN) method, which assigns a class label as a partial supervision to each input instance by considering the training set. The class label of each unlabeled input instance, is predicted, by utilizing a linear activation function and the exclusive total threshold. Finally, the RSFFNN model updates the number of clusters and density of each cluster.

Then, a dynamic unsupervised FFNN (DUFFNN) clustering method is developed by adapting the structure and the features of the RUFFNN method. The DUFFNN clustering method with incremental learning ability is suitable for clustering non-stationary continuous data model, and starts without any random parameters or coefficient value which needs predefinition. The proposed DUFFNN clustering method is able to control and delete attributes with weak weights to reduce the data dimensions, and data with solitary thresholds in order to reduce noise. Moreover, the DUFFNN method has the capabilities of increasing clustering accuracy and improving training time in a single epoch clustering without weight updating, and improving time complexity and memory complexity of clustering.

The DUFFNN clustering is improved to dynamic semi-supervised FFNN (DSFFNN) clustering by assigning a class label to each unlabeled data by considering a linear activation function and the exclusive threshold for more accurate results.

In order to accomplish the fourth and fifth objectives of the research, the performances of the proposed models are evaluated and compared with other related models by using the various datasets from the UCI machine learning repository, and the breast cancer dataset from the University of Malaya medical centre (UMMC) to predict the survival time of patients.

4.2.5 System Design

Based on the proposed model, the designs and algorithms of following methods are described, which are considered as contributions of this study.

- A developed RUFFNN method as a single epoch clustering method, which uses non random weights, and controls the number of dimensions of data and noise.
- An improved RSFFNN clustering method based on the RUFFNN model, by assigning a class label to the each unlabeled data through using a linear activation function and the exclusive threshold for more accurate results.
- A developed DUFFNN clustering method, which inherits the structure, features and capabilities of the RUFFNN clustering. The DUFFNN clustering with incremental lifelong or online learning property is developed for real non stationary environments.

- An improved DSFFNN clustering method based on the DUFFNN method, by assigning a class label to the each unlabeled data through using a linear activation function and the exclusive threshold for more accurate results.

The details of the system design are explained in Chapter 5.

4.2.6 System and Data Requirements for Running Experiment

This section provides a summary of both the hardware and software requirements to run the proposed system. Furthermore, this section provides a summary of datasets used in this study for experimental evaluation.

All the steps of the proposed methods are implemented in Visual C#.Net and Visual Basic under Microsoft windows 7 professional operating system (OS) by 4 GHz Pentium processor.

Visual *C#*.net is a powerful programming language for building real applications in the infrastructure and low level machine language. It properly manages the memory in front of data garbages, unpointed free cells, and wasted cells during execution of the program. Also, it is able to handle the errors of the program by using a strong compiler, and has cross-language capabilities to interoperate with any other language on the .NET platform. Table 4-1 shows the system characteristics.

Table 4-1: The system information of model implementation

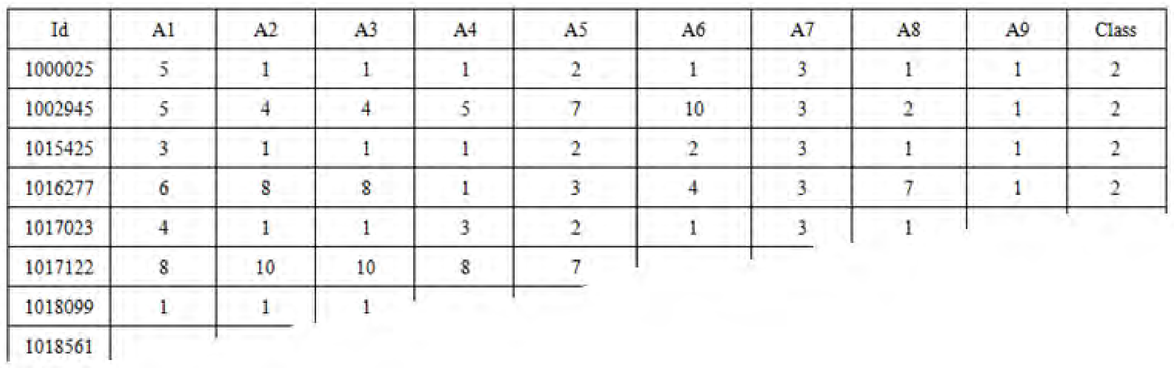
System Information	
Operating System	Microsoft Windows 7 Professional
Processor	Pentium 2 GHz
Physical Memory	4 GB
Programming Language	Visual C#.Net and Visual Basic 6.0

As mentioned in Section 4.2.3, the proposed model is experimentally evaluated in order to achieve the fourth and fifth objectives of the research. Validation experiments are performed on nine datasets selected from different domains from the UCI Irvine Machine Learning Database Repository (Asuncion & Newman, 2007) and the breast cancer dataset from the University of Malaya medical centre (UMMC). As previously mentioned, most conventional methods do not satisfactorily cluster these datasets. The details of the features of datasets are provided in APPENDIX B.

1) Breast Cancer Wisconsin (original) dataset was selected from the UCI Repository. The collected dataset was from the University of Wisconsin Hospitals, Madison, as reported by Dr. William H. Wolberg through his clinical cases (Asuncion & Newman, 2007; Wolberg & Mangasarian, 1990). As mentioned in the UCI Repository, the dataset characteristic is multivariable, the attribute characteristic is an integer, the number of cases is 699, and the number of attributes is 10 from the life area. Breast cancer can be classified into benign and malignant. The Breast Cancer Wisconsin dataset has 683 data points after cleaning. Therefore, in each iteration, we considered 614 data points as the training set and 69 data points as the test set. Table 4-2 and Figure 4-1 illustrate the breast cancer Wisconsin (original) dataset features and its sample.

Table 4-2: The features of the breast cancer Wisconsin dataset

Data Set Characteristics:	Multivariate	Number of Data:	699	Area:	Life
Attribute Characteristics:	Integer	Number of Attributes:	10	Date Donated	1992-07-15
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	77542



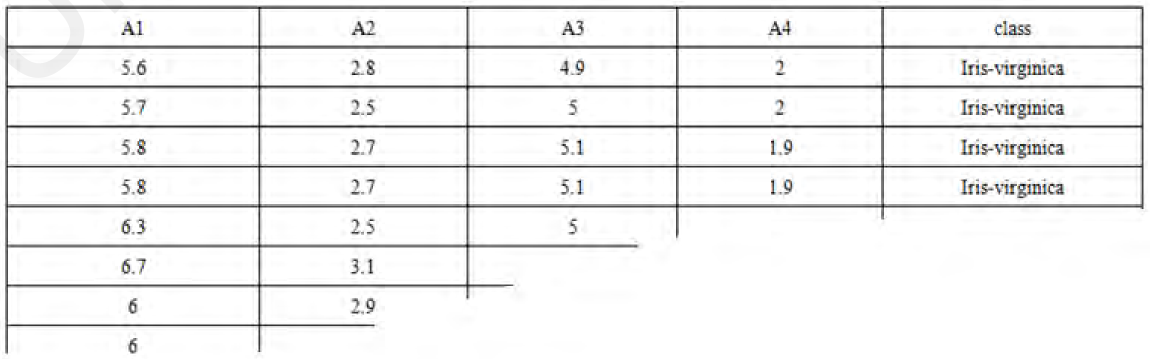
Id	A1	A2	A3	A4	A5	A6	A7	A8	A9	Class
1000025	5	1	1	1	2	1	3	1	1	2
1002945	5	4	4	5	7	10	3	2	1	2
1015425	3	1	1	1	2	2	3	1	1	2
1016277	6	8	8	1	3	4	3	7	1	2
1017023	4	1	1	3	2	1	3	1		
1017122	8	10	10	8	7					
1018099	1	1	1							
1018561										

Figure 4-1: The sample of breast cancer Wisconsin dataset

2) Iris dataset was selected from the UCI Repository. The Iris plants dataset was created by Fisher R.A. (Asuncion & Newman, 2007; R. Fisher, 1950). As reported in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is real, the number of cases is 150, the number of attributes is 4 from the life area. The Iris dataset can be classified into Iris Setosa, Iris Versicolour and Iris Virginica. Table 4-3 and Figure 4-2 illustrate the Iris dataset features and its sample.

Table 4-3: The features of the Iris dataset

Data Set Characteristics:	Multivariate	Number of Data:	150	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	4	Date Donated	1988-07-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	390757



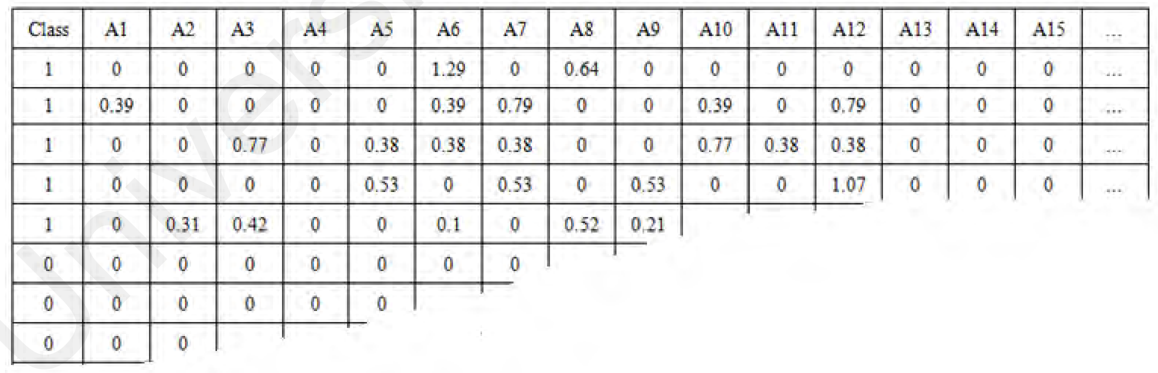
A1	A2	A3	A4	class
5.6	2.8	4.9	2	Iris-virginica
5.7	2.5	5	2	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
6.3	2.5	5		
6.7	3.1			
6	2.9			
6				

Figure 4-2: The sample of the Iris dataset

3) Spambase dataset was selected from the UCI Repository. The Spam E-mail dataset was created by Mark Hpkins, Erik Reeber, George Forman, Jaap Suermondt (Asuncion & Newman, 2007). As reported in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristics are integer-real, the number of data is 4601 and the number of attributes is 57 from the computer area. The Spambase dataset can be classified into Spam and Non-Spam. Table 4-4 and Figure 4-3 illustrate the Spambase dataset features.

Table 4-4: The features of the Spambase dataset

Data Set Characteristics:	Multivariate	Number of Data:	4601	Area:	Computer
Attribute Characteristics:	Integer, Real	Number of Attributes:	57	Date Donated	1999-07-01
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	65066



Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	...
1	0	0	0	0	0	1.29	0	0.64	0	0	0	0	0	0	0	...
1	0.39	0	0	0	0	0.39	0.79	0	0	0.39	0	0.79	0	0	0	...
1	0	0	0.77	0	0.38	0.38	0.38	0	0	0.77	0.38	0.38	0	0	0	...
1	0	0	0	0	0.53	0	0.53	0	0.53	0	0	1.07	0	0	0	...
1	0	0.31	0.42	0	0	0.1	0	0.52	0.21							
0	0	0	0	0	0	0	0									
0	0	0	0	0	0											
0	0	0														

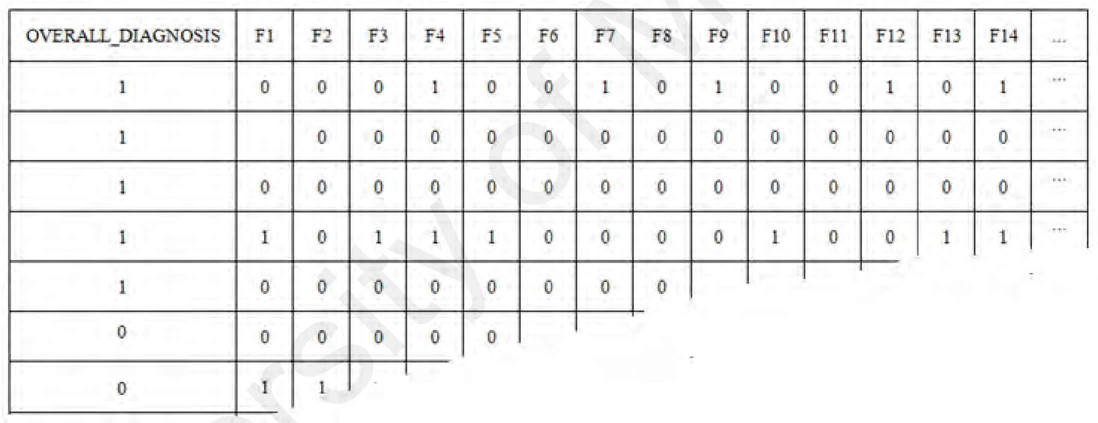
Figure 4-3: The sample of the Spambase dataset

4) SPECT heart dataset was selected from the UCI Repository. Kurgan et al. created the dataset that described diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images (Asuncion & Newman, 2007; Kurgan, et al., 2001). As reported by the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is categorical, the number of cases is 267 and the number of

attributes is 22 from the life area. The SPECT Heart dataset can be classified into normal and abnormal. Tables 4-5 and Figure 4-4 illustrate SPECT Heart dataset features and its sample.

Table 4-5: The features of SPECT Heart dataset

Data Set Characteristics:	Multivariate	Number of Data:	267	Area:	Life
Attribute Characteristics:	Categorical	Number of Attributes:	22	Date Donated	2001-10-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	85546



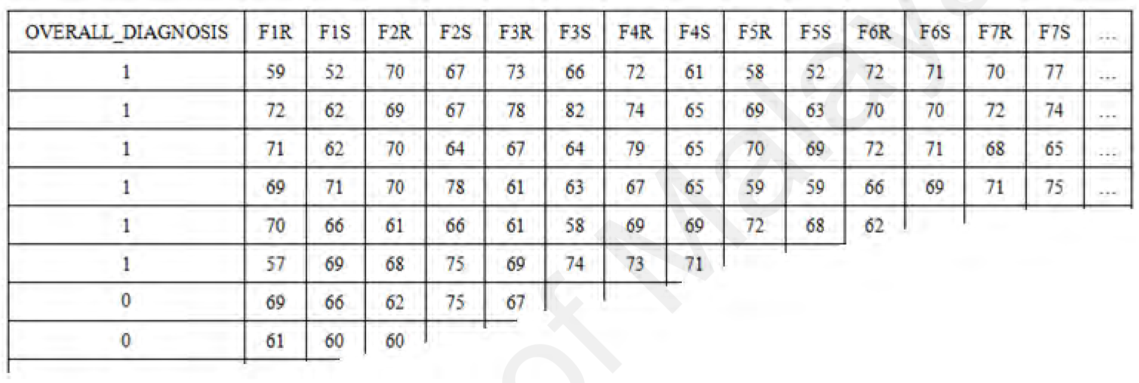
OVERALL_DIAGNOSIS	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	...
1	0	0	0	1	0	0	1	0	1	0	0	1	0	1	...
1		0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	...
1	0	0	0	0	0	0	0	0							
0	0	0	0	0	0										
0	1	1													

Figure 4-4: The sample of SPECT Heart dataset

5) SPECTF Heart dataset was selected from the UCI Repository. Kurgan et al. created the dataset that described diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images in different type in SPECTF Heart dataset (Asuncion & Newman, 2007; Kurgan, et al., 2001). As reported in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is integer, the number of cases is 267 and the number of the attributes is 44 from the life area. The SPECTF Heart dataset can be classified into normal and abnormal. Tables 4-6 and Figure 4-5 illustrate SPECTF Heart dataset features and its sample.

Table 4-6: The features of SPECTF Heart dataset

Data Set Characteristics:	Multivariate	Number of Data:	267	Area:	Life
Attribute Characteristics:	Integer	Number of Attributes:	44	Date Donated:	2001-10-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	21457



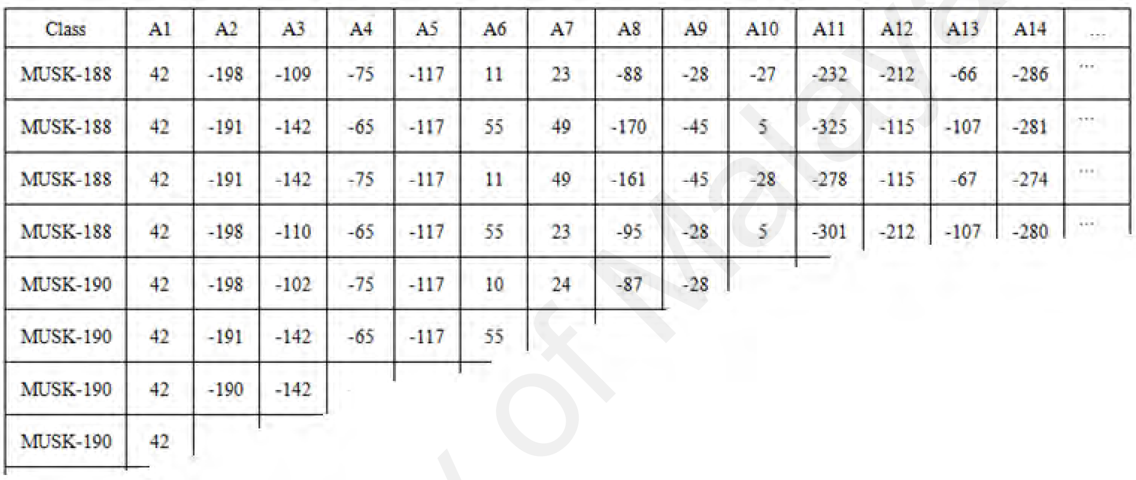
OVERALL_DIAGNOSIS	F1R	F1S	F2R	F2S	F3R	F3S	F4R	F4S	F5R	F5S	F6R	F6S	F7R	F7S	...
1	59	52	70	67	73	66	72	61	58	52	72	71	70	77	...
1	72	62	69	67	78	82	74	65	69	63	70	70	72	74	...
1	71	62	70	64	67	64	79	65	70	69	72	71	68	65	...
1	69	71	70	78	61	63	67	65	59	59	66	69	71	75	...
1	70	66	61	66	61	58	69	69	72	68	62				
1	57	69	68	75	69	74	73	71							
0	69	66	62	75	67										
0	61	60	60												

Figure 4-5: The sample of SPECTF Heart dataset

6) Musk1 dataset (version 1 of the musk dataset) was selected from the UCI Repository. The dataset was created by the "Artificial Intelligence" group at Arris Pharmaceutical Corporation, and describes a set of Musk or Non-Musk molecules (Asuncion & Newman, 2007). The goal is to train the dataset in order to predict whether new molecules will be Musk or Non-Musk based on their features. As mentioned in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is integer, the number of instance is 476, and the number of the attributes is 168 from the physical area. The Musk1 dataset can be classified into Musks or Non-Musks. Table 4-7 and Figure 4-6 illustrate the Musk1 dataset features and its sample.

Table 4-7: The features of the Musk1 dataset

Data Set Characteristics:	Multivariate	Number of Data:	476	Area:	Physical
Attribute Characteristics:	Integer	Number of Attributes:	168	Date Donated	1994-09-12
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	16271



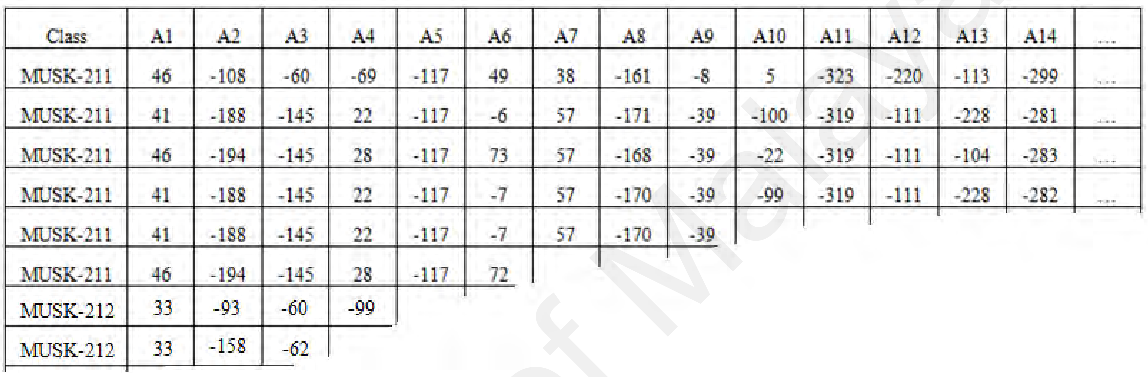
Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	...
MUSK-188	42	-198	-109	-75	-117	11	23	-88	-28	-27	-232	-212	-66	-286	...
MUSK-188	42	-191	-142	-65	-117	55	49	-170	-45	5	-325	-115	-107	-281	...
MUSK-188	42	-191	-142	-75	-117	11	49	-161	-45	-28	-278	-115	-67	-274	...
MUSK-188	42	-198	-110	-65	-117	55	23	-95	-28	5	-301	-212	-107	-280	...
MUSK-190	42	-198	-102	-75	-117	10	24	-87	-28						
MUSK-190	42	-191	-142	-65	-117	55									
MUSK-190	42	-190	-142												
MUSK-190	42														

Figure 4-6: The sample of the Musk1 dataset

7) Musk2 dataset (version 2 of the musk dataset) was selected from the UCI Repository. The dataset was created by the “Artificial Intelligence” group at Arris Pharmaceutical Corporation, and describes a set of Musk or Non-Musk molecules (Asuncion & Newman, 2007). The goal is to train the dataset in order to predict whether new molecules will be Musk or Non-Musk based on their features. As mentioned in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is integer, the number of data is 6598, and the number of attributes is 168 from the physical area. The Musk2 dataset can be classified into Musks or Non-Musks. Table 4-8 and Figure 4-7 illustrate the Musk2 dataset features.

Table 4-8: The features of the Musk2 dataset

Data Set Characteristics:	Multivariate	Number of Data:	6598	Area:	Physical
Attribute Characteristics:	Integer	Number of Attributes:	168	Date Donated	1994-09-12
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	16028



Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	…
MUSK-211	46	-108	-60	-69	-117	49	38	-161	-8	5	-323	-220	-113	-299	…
MUSK-211	41	-188	-145	22	-117	-6	57	-171	-39	-100	-319	-111	-228	-281	…
MUSK-211	46	-194	-145	28	-117	73	57	-168	-39	-22	-319	-111	-104	-283	…
MUSK-211	41	-188	-145	22	-117	-7	57	-170	-39	-99	-319	-111	-228	-282	…
MUSK-211	41	-188	-145	22	-117	-7	57	-170	-39						
MUSK-211	46	-194	-145	28	-117	72									
MUSK-212	33	-93	-60	-99											
MUSK-212	33	-158	-62												

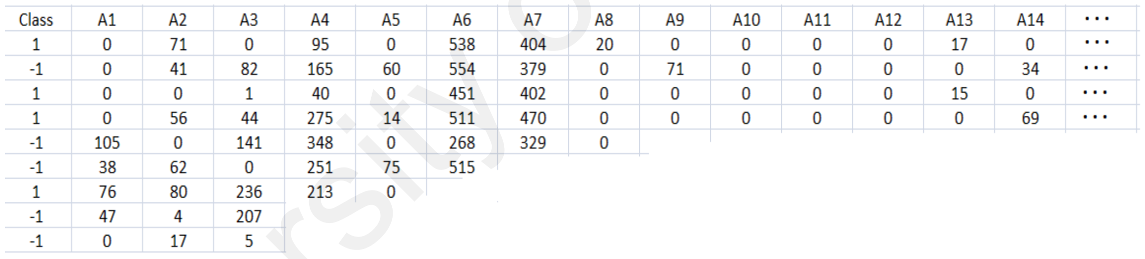
Figure 4-7: The sample of the Musk2 dataset

8) The Arcene dataset was collected from two different sources: the national cancer institute (NCI) and the eastern Virginia medical school (EVMS) (Asuncion & Newman, 2007). As mentioned in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is real, the number of cases is 900, and the number of attributes is 10000 from the life area. All data were obtained by merging three mass-spectrometry datasets to create training and test data as a benchmark. The training and validation instances include patients with cancer (ovarian, prostate cancer), and healthy patients. Each dataset of training and validation contains 44 positive samples and 56 negative instances with 10,000 attributes. We considered the training dataset and validation dataset with 200 total instances together as one set. The Arcene dataset can be classified into cancer patients and healthy patients. Arcene's task is to distinguish cancer versus normal patterns from mass-spectrometric data (Asuncion & Newman, 2007). This dataset is one of 5 datasets of the NIPS 2003 feature selection challenge

(Guyon, 2003; Guyon & Elisseeff, 2003). As such, most current publications in this area are on the selection of the best attributes so as to reduce the dimension of the arcene dataset to get better clustering accuracy, to reduce the central processing unit (CPU) time usage and memory usage (Guyon, 2003; Guyon & Elisseeff, 2003). Table 4-9 and Figure 4-8 illustrate the Arcene dataset features.

Table 4-9: The features of the Arcene dataset

Data Set Characteristics:	Multivariate	Number of Data:	900	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	10000	Date Donated	2008-02-29
Associated Tasks:	Classification	Missing Values?	N/A	Number of Web Hits:	55435



Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	...
1	0	71	0	95	0	538	404	20	0	0	0	0	17	0	...
-1	0	41	82	165	60	554	379	0	71	0	0	0	0	34	...
1	0	0	1	40	0	451	402	0	0	0	0	0	15	0	...
1	0	56	44	275	14	511	470	0	0	0	0	0	0	69	...
-1	105	0	141	348	0	268	329	0							
-1	38	62	0	251	75	515									
1	76	80	236	213	0										
-1	47	4	207												
-1	0	17	5												

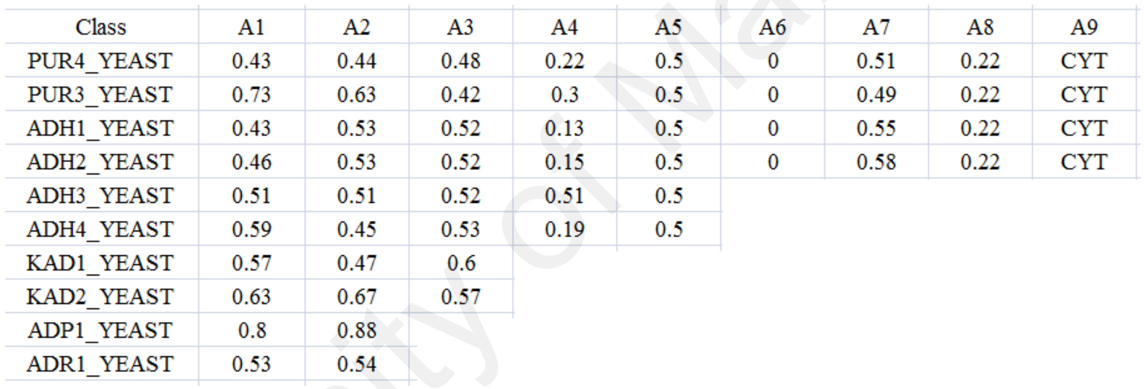
Figure 4-8: The sample of the Arcene dataset

9) The Yeast dataset was obtained from the UCI Repository. The collected dataset was reported by Kentai Nakai from Institute of Molecular and Cellular Biology, university of Osaka (Asuncion & Newman, 2007), in order to predict the cellular localization sites of proteins. As mentioned in the UCI Repository, the dataset characteristic is multivariable, the attributes characteristic is real, the number of cases is 1484, and the number of attributes is 8 from the life area. The classes are Cytosolic, Nuclear, Mitochondrial, Membrane protein: no *N*-terminal signal, Membrane protein: uncleaved signal and Membrane protein: cleaved signal. Extracellular, Vacuolar,

Peroxisomal and Endoplasmic reticulum lumen (Asuncion & Newman, 2007). Table 4-10 and Figure 4-9 illustrate the Arcene dataset features.

Table 4-10: The features of the Yeast dataset

Data Set Characteristics:	Multivariate	Number of Data:	1484	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	8	Date Donated	1996-09-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	167248



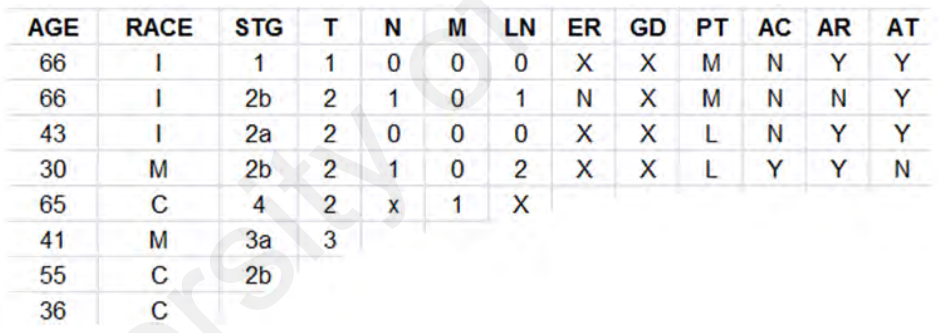
Class	A1	A2	A3	A4	A5	A6	A7	A8	A9
PUR4_YEAST	0.43	0.44	0.48	0.22	0.5	0	0.51	0.22	CYT
PUR3_YEAST	0.73	0.63	0.42	0.3	0.5	0	0.49	0.22	CYT
ADH1_YEAST	0.43	0.53	0.52	0.13	0.5	0	0.55	0.22	CYT
ADH2_YEAST	0.46	0.53	0.52	0.15	0.5	0	0.58	0.22	CYT
ADH3_YEAST	0.51	0.51	0.52	0.51	0.5				
ADH4_YEAST	0.59	0.45	0.53	0.19	0.5				
KAD1_YEAST	0.57	0.47	0.6						
KAD2_YEAST	0.63	0.67	0.57						
ADP1_YEAST	0.8	0.88							
ADR1_YEAST	0.53	0.54							

Figure 4-9: The sample of the Yeast dataset

10) The breast cancer dataset consists of the patients data from the University of Malaya Medical centre (UMMC), Kuala Lumpur from 1992 until 2002 (Hazlina, et al., 2004). As mentioned in Table 4-11, the dataset is divided into 9 sub sets based on the interval of survival time: 1_{st} year, 2_{nd} year, … , 9_{th} year. Also, Figure 4-10 shows the sample of breast cancer dataset from the UMMC.

Table 4-11: The observed data of breast cancer based on the interval of survival time

Year of treatment	1_{st} year	2_{nd} year	3_{rd} year	…	8_{th} year	9_{th} year
1993	Data from 1993 to 1994	Data from 1993 to 1995	Data from 1993 to 1996	…	Data from 1993 to 2001	Data from 1993 to 2002
1994	Data from 1994 to 1995	Data from 1994 to 1996	Data from 1994 to 1997	…	Data from 1994 to 2002	
1995	Data from 1995 to 1996	Data from 1995 to 1997	Data from 1995 to 1998	…		
…	…	…	…			
2000	Data from 2000 to 2001	Data from 2000 to 2002				
2001	Data from 2001 to 2002					



AGE	RACE	STG	T	N	M	LN	ER	GD	PT	AC	AR	AT
66	I	1	1	0	0	0	X	X	M	N	Y	Y
66	I	2b	2	1	0	1	N	X	M	N	N	Y
43	I	2a	2	0	0	0	X	X	L	N	Y	Y
30	M	2b	2	1	0	2	X	X	L	Y	Y	N
65	C	4	2	x	1	X						
41	M	3a	3									
55	C	2b										
36	C											

Figure 4-10: The sample of breast cancer from UMMC dataset

The number of cases in the dataset was 827, the number of attributes was 13 continuous and one attribute for showing the binary class in two cases of alive or dead. The used breast cancer dataset from UMMC has class labels of '0' for alive and '1' for dead as constraints.

4.2.7 Experimental Evaluation

As mentioned in Section 4.2.3, in order to achieve the fourth and fifth objectives of the research and evaluate the performance of the proposed system, two main experiments are considered. Chapter 6 is the first experiment, that the developed RUFFNN and RSFFNN methods are compared against some related static clustering methods. Chapter 7 is the second experiment, that the results of the developed DUFFNN and DSFFNN methods are compared against the result of the first experiment, and some related dynamic clustering methods. Furthermore, the RSFFNN and DSFFNN as semi-supervised clustering methods are applied to the prognosis of the breast cancer in survival breast cancer analysis by comparison with some related methods. The details of experiments and results are discussed in Chapters 6 and 7.

The performances of the proposed methods are compared with several current effective related methods in the scope of this research. In addition, the accuracy and speed of clustering as well as time complexity and memory complexity are assessed.

- **Quality metrics:** For experimentation, the speed of processing is measured by the number of epochs and CPU time usage, which is measured by using a timer module in Visual *C* or Visual Basic environment for implementation of the proposed framework. Also, the time usage and memory usage are estimated by time complexity and memory complexity. The complexity of an algorithm demonstrates the efficiency of the algorithm in terms of the amount of the processed data by the algorithm, and often has natural units for the domain and range. There are two main complexity measurements. Time complexity describes the dimension of time for processing of the input data by an algorithm. Time efficiency depends on many factors such as the speed of the processor or compiler of the machine for processing the algorithm. However, some factors such as the number of multiplications of the process is considered in time complexity. In addition, memory complexity or space utilization measures the memory space, which is necessary to process the input data by the program algorithm (Papadimitriou, 2003). In this research, the time complexity is computed by using the number of input data, attributes, training iterations and clusters. Also, the memory complexity is computed by using the number of input data, attributes, training iterations and clusters; and the densities of the clusters and the size of the attributes. There are different methods in order to measure and evaluate the clustering accuracy such as external evaluation. In this method the results of clustering are evaluated based on unused data for clustering such as a class label. An external evaluation measures how close the clustering results are to the known benchmark results, for example externally given class label. In the scope of the thesis, there are limited published articles in the literature that can be used for comparison of proposed methods. Therefore, we follow the method of computing accuracy based on existing published papers. The *correctly classified nodes* (*CCN*) (Bouchachia, et al., 2007; Camastra & Verri, 2005; Costa & Oliveira, 2007) and *F-measure* or balanced *F-Score* function (Van Rijsbergen, 1979), are two methods for external evaluation measurement of clustering accuracy, as discussed in Section 2.5.1. The choice of these measuring methods is based on the most common-used measures in the scope of this research (Andrew, 2014; Chaimontree, et al., 2010; Rendón, et al., 2011; Rendón, et al., 2011; Sung & Mukkamala, 2003). Therefore, the accuracy of the methods is measured through the number of clusters and the quantity of the *correctly classified nodes* (*CCN*). The *CCN* shows the total of nodes and their densities, with the correct class in the correct related cluster, in all created clusters of the method. The *CCN* is equal to the true positive and true negative nodes. The accuracy of the proposed method is measured by the *CCN* in comparison with other methods; furthermore, the accuracy of clustering is also measured by using the *F-measure* or balanced *F-Score* function with 10 folds of the test set (Andrew, 2014;

Chaimontree, et al., 2010; Rendón, et al., 2011; Rendón, et al., 2011; Sung & Mukkamala, 2003). The variables of true positive, false positive, true negative, false negative, are used to merit clustering of a data with the desired correct cluster. The *F-measure* is called *F1 measure*, because recall and precision are weighted. *Recall* is the probability that a randomly selected relevant instance is recovered in a search. *Precision* is the probability that a randomly selected recovered instance is relevant.

As was earlier mentioned, the accuracy of the proposed method is measured with 10 folds of the test set, based on 10 fold cross validation. In order to evaluate each method, the dataset is divided into 10 subsets through random selection of the dataset, however, the test sets cover all data points in the dataset. Nine subsets are considered for training, and one subset is considered for testing. This process is repeated 10 times, and the average of the results is considered as the final value of accuracy.

The precision of computing is considered with 15 decimal places. The time complexity and memory complexity are measured by the number of input data, training iterations, clusters and the densities of clusters.

Needless to say, the scope of this research is ODUFFNN clustering and the proposed method clusters datasets without utilizing the class label. However, selected datasets have a training set and a test set with class labels, which are used to compute the accuracy of the proposed method, as is normally done in similar researches (Andrew, 2014; Bouchachia, et al., 2007; Camastra & Verri, 2005; Costa & Oliveira, 2007; Rendón, et al., 2011; Rendón, et al., 2011; Sung & Mukkamala, 2003). For simulation of the purposes, during training or testing modes, only one instance of the training or test data is randomly selected and is processed by the model.

4.3 Summary

This chapter outlined the steps of the research methodology followed in this thesis. First, a research methodology framework was proposed. This research focuses on a developing a dynamic unsupervised feedforward neural network clustering method. Therefore, as we explained, after an reviewing the problems, we proposed four models and subsequently developed the RUFFNN, RSFFNN, DUFFNN, and DSFFNN models. Finally, in the last stage of the research methodology framework, we discussed the design of evaluation procedure used for the proposed models. The details of the designs and development of the models are discussed in the next chapter.

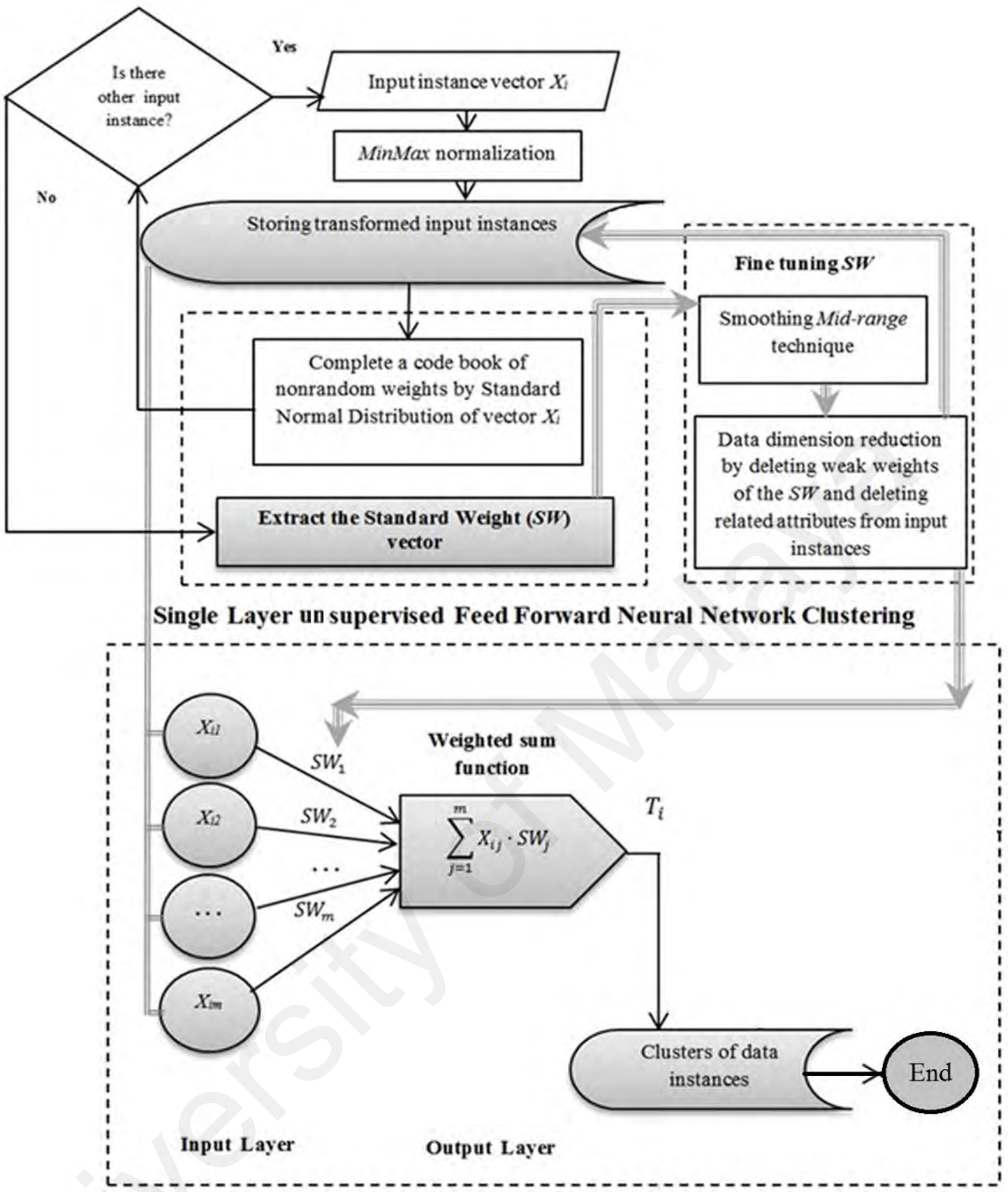
CHAPTER 5: SYSTEM DESIGN

5.1 Introduction

In this chapter, the system designs and algorithms of proposed models are discussed in details. The discussion starts with the designs and algorithms of the developed real unsupervised feedforward neural network (RUFFNN) clustering method in the stationary mode, and then subsequently the developed DUFFNN clustering based on the RUFFNN method for online clustering suitable for dynamic mode. As mentioned in the last chapter, in order to improve the accuracy of the results of clustering based on literature, the RUFFNN and DUFFNN clustering methods are improved to real semi-supervised FFNN (RSFFNN) and dynamic semi-supervised FFNN (DSFFNN) methods, respectively. In addition, the designs and algorithms of the RSFFNN and DSFFNN methods are discussed in this chapter too.

5.2 The Real Unsupervised Feedforward Neural Network Clustering

In this section, we illustrate the design of the real unsupervised feedforward neural network (RUFFNN) clustering method in detail; and show how it solves the aforementioned problems related to the static unsupervised feedforward neural network (UFFNN) clustering methods. The RUFFNN method can be applied as a UFFNN clustering method. Figure 5-1 shows the design of the real unsupervised feedforward neural network method for clustering:



Is there other input instance?
Yes
No
Input instance vector X_i
MinMax normalization
Storing transformed input instances
Fine tuning *SW*
Smoothing *Mid-range* technique
Data dimension reduction by deleting weak weights of the *SW* and deleting related attributes from input instances
Complete a code book of nonrandom weights by Standard Normal Distribution of vector X_i
Extract the Standard Weight (*SW*) vector
Single Layer unsupervised Feed Forward Neural Network Clustering
X_{i1}
X_{i2}
X_{im}
SW_1
SW_2
SW_m
Weighted sum function
$\sum_{j=1}^{m} X_{ij} \cdot SW_j$
T_i
Clusters of data instances
End
Input Layer
Output Layer

Figure 5-1: The design of real unsupervised feedforward neural network model for clustering

As shown in Figure 5-1, the RUFFNN considers a matrix of dataset as input data for clustering. During training, a non-random weights code book was learned through the input data matrix directly, by using normalized input data and standard normal distribution. A standard weight vector was extracted from the code book, and after fine tuning is applied by single layer feedforward neural network clustering section. The fine tuning process includes two techniques of smoothing the weights and pruning the weak weights. The first, the *Mid-range* technique, a popular smoothing technique, is used (Jean & Wang, 1994; Gui et al., 2001). Then, the model prunes the data node attributes with weak weights in order to reduce the dimension of data. Consequently, the single layer feedforward neural network clustering section generates the exclusive threshold of each input instance (record of the data matrix) based on standard weight vector. Finally, the input instances were clustered on the basis of their exclusive thresholds.

5.2.1 Analysis of the RUFFNN Clustering Method

In this section, we illustrate the complete algorithm of the real unsupervised feedforward neural network (RUFFNN) clustering method, and discuss the details of the proposed clustering method as shown in Figure 5-1 step by step. Figure 5-2 shows the algorithm of the real unsupervised feedforward neural network (RUFFNN) method:

```
Algorithm: The RUSFFNN clustering
Input :  dataset $X$;
Output: Clusters of dataset;
Initialize the parameters:
Let $X$: Data node set;
Let $n$ : Number of nodes;
Let $m$ : Number of attributes;
Let $i$ : Current number of the node;
Let $j$ : Current number of the attribute;
Let $X_i$: Current input instance of dataset;
Let $W_{ij}$ : Weight of attribute $j$ of input instance $X_i$;
Let $SW$: Standard Weight vector;
Let $SW_j$ : $j_{th}$ Component of the $SW$ vector;
Let $T_i$: Threshold of input instance of $X_i$ ;
Method:
{
 1- // Preprocessing of dataset
    {    // Data preprocessing based on $MinMax(X_{ij})$
          Forall $i=1$ to $n$
           Forall $j=1$ to $m$
           {
              $X_{ij} = \left( X_{ij} - Min(X_{ij}) \middle/ Max(X_{ij}) - Min(X_{ij}) \right)$;
           }
```

```
        // **Create the code book of the weights**
        // Compute the *standard normal distribution* (*SND*) of each input data attribute value
           $X_{ij}$ based on $\mu_i$ and $\sigma_i$ which are mean and standard deviation of the input data $X_i$:
          Forall $i=1$ to $n$
           Forall $j=1$ to $m$
           {
           $SND(X_{ij}) = (X_{ij} - \mu_i)/\sigma_i$ ;
           // Consider $W_{ij}$ as weight of $X_{ij}$ equal $SND(X_{ij})$
               $W_{ij} = SND(X_{ij})$;
           }
// Generate the *global geometric mean* vector of the code book of non-random weights as the *standard weight* vector (*SW*),
// The $SW_j$ is the geometric mean of the real weights of each attribute of the input dataset
        Forall $j$=1 to $m$
      $SW_j = \left(\prod_{i=1}^{n} W_{ij}\right)^{1/n}$ ;
// The $SW$ includes $SW_j$
        $\overrightarrow{SW} = (SW_1, SW_2, \ldots, SW_m)$
2- // **Fine tuning through two techniques:**
      // a) Smooth the components of the $SW$ vector
            Forall $j$=1 to $m$
                  *Mid-range* ($SW_j$) ;
      // b) Data dimension reduction
       Delete attributes with weak weights of $SW_j$ which are close to zero ;
3- // **Process of single layer unsupervised feedforward neural network for clustering of input dataset**
      // Compute the exclusive threshold of each input instance of $X_i$
            Forall $i=1$ to $n$
                  Forall $j=1$ to $m$
                  { If $BMW_j <> 0$
                   $T_i = T_i + X_{ij} . SW_j$;}
      // Recognize and delete noise
            Delete isolated input instances with solitary thresholds $T_i$;
      // Clustering
         {
         Group the data points of input instances with similar thresholds ($T_i$) in one cluster;
         Learn and generate optimized number of clusters and their densities;
         }
 }
```

Figure 5-2: The algorithm of the real unsupervised feedforward neural network clustering

The RUFFNN clustering method involves several stages:

1) **Preprocessing:** Commonly preprocessing is the contributing feature in developing efficient techniques for low training time and high accuracy of feedforward neural network clustering (Han & Kamber, 2011; Larochelle, et al., 2012; Oh & Park, 2011).

- Data Preprocessing: In the RUFFNN model, the *MinMax* normalization technique was used to transform an input value of each attribute to fit to a specific range such as [0,1] (Han & Kamber, 2011). The input matrix of values consists of every single value with individual measurement unit type and range. The fundamental assumption of the proposed method is that no missing values exist and every value is acceptable. For this purpose, other data preprocessing techniques such as data cleaning are valuable. Equation (5.1) shows the special formula used to normalize the input values (Han & Kamber, 2011):

$$Normalized(X_{ij}) = \left({X_{ij} - Min(X_{ij})} \middle/ {Max(X_{ij}) - Min(X_{ij})} \right) \times (newMax - newMin) + newMin \tag{5.1}$$

Where X_{ij} is j_{th} attribute value of input instance *i*. In the range of attribute j for all input instances, $Min(X_{ij})$ is minimum value, and $Max(X_{ij})$ is maximum value; $newMax$ is equal to 1 and $newMin$ is equal to 0.

- Creating a Code Book of Non-random Weights: In order to solve the associated problem with the use of random weights, the RUFFNN method creates a code book of nonrandom weights. At this stage, the proposed model computes the mean μ_i of the normalized record X_i. Then the standard deviation σ_i of the input instance of X_i is computed by considering μ_i . This is the definition of the *standard normal distribution* (*SND*) (Ziegel, 2002) as shown in Figure 5-3.

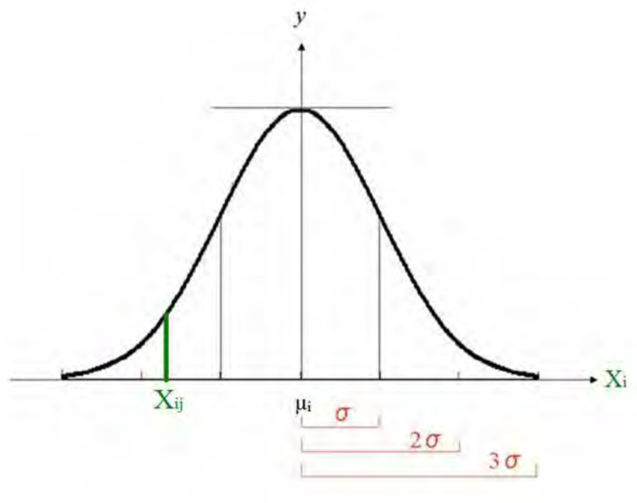


Figure 5-3: Standard normal distribution for each attribute value of input instance X_i

The *SND* shows how far each attribute value of the input instance X_i is from the mean, in the metric standard deviation unit. In this step, each normalized attribute value of the input instance X_i is considered as the weight W_{ij} for that value. Each element or code word of the weight code book is equal to W_{ij} . The model receives other input values of the instances and computes the code book of all weights of input values. Therefore, each weight vector of the code book is computed based on the *SND* of each input instance value of X_i as shown in Equation (5.2).

$$SND(X_{ij}) = {(X_{ij} - \mu_i)}/{\sigma_i} \tag{5.2}$$

The $SND(X_{ij})$ is a standard normalized value of each attribute value of the input instance (record). μ_i and σ_i are mean and standard deviation of the input instance record. Therefore, each $SND(X_{ij})$ shows the distance of each input value of each instance from the mean of the input instance. Accordingly, each W_{ij} as weight of X_{ij} is equal to $SND(X_{ij})$ as in Equation (5.3) and the initialization of weights is not at random:

$$W_{ij} = SND(X_{ij}) \qquad i = 1,2,\dots,n\ ; j = 1,2,\dots,m \tag{5.3}$$

- Achieving *standard weight* (*SW*) vector: In the SOM, the weight of the code book which is nearest to the input vector is distinguished as the winner node and the *best matching unit* (*BMU*). Similarly, the RUFFNN method tries to learn and extract a standard unique weight vector through real weights code book, but in a slightly different way. Each weight vector of the code book is related to each input data vector and is computed by applying the *SND* based on the mean of the input data vector. The *SW* vector is the *Geometric Mean* (Jacquier, et al., 2003; Vandesompele, et al., 2002) vector of the code book of the non-random weights, and it is computed based on the centre of gravity of the matrix of input data vectors. In the RUFFNN method, the code book of real weights is initialized by considering properties of input values directly and without using any random values or random parameters. In order to extract a standard unique weight vector through the real weights code book, several techniques exist, such as principal component analysis (PCA) by Jolliffe (1986) which is one powerful method in dimension reduction to date (Daffertshofer, et al., 2004; Jolliffe, 1986; Lindsay, et al., 2002; Van der Maaten, et al., 2009). However, the time complexity of the PCA is $O(p^2n)+O(p^3)$ and the PCA losses the input values during training. Therefore at this stage, the RUFFNN model computes the *SW* vector by training the real weights in the code book. The *SW* vector is the extract of the code book of real weights as a base and a criterion weight vector for clustering input instances of the dataset globally. In other words, the *SW* is the essential feature of the RUFFNN model. The *SW* consists of the components SW_j. There are several methods to compute the SW_j, such as computing by the square root of the sum of the weights of each attribute of the input data, or $\left(\sum_{i=1}^{n} W_{ij}^2\right)^{1/2}$, however, in the RUFFNN algorithm of Figure 6.2, the SW_j is computed by the n_{th} root of the product of the weights of each attribute of the input data (*Geometric Mean*). The parameter n is the number of input instances, i is current number of the node of input instance; m is the number of attributes and j is the current number of the attribute of input instance. Equations (5.4) and (5.5) show these relationships.

$$SW_j = \left(\prod_{i=1}^{n} W_{ij}\right)^{1/n} \tag{5.4}$$

$$\overrightarrow{SW} = (SW_1, SW_2, \ldots, SW_m) \tag{5.5}$$

Table 5-1 illustrates the code book of the weights and the process of extracting the standard weight vector.

Table 5-1: The code book of the weight vectors and the standard weight vector

The code book of weight vectors				
Weight vector of X_i	$Attribute_1$	$Attribute_2$	…	$Attribute_m$
Weight vector of X_1	W_{11}	W_{12}	…	W_{1m}
Weight vector of X_2	W_{21}	W_{22}	…	W_{2m}
…	…	…	…	…
Weight vector of X_n	W_{n1}	W_{n2}	…	W_{nm}
$\overrightarrow{SW}$	SW_1	SW_2	…	SW_m

In the RUFFNN model, learning does not require computing any error function, such as the mean square errors and updating weights in any training cycle; therefore the approach results in a reduced training time. The main goal of the RUFFNN model is learning of the *SW* vector as the criterion weight vector. The next stages will show how the thresholds are computed and the dataset of input instances will be clustered easily based on just the *SW*.

2) **Fine Tuning:** This process refers to the accurate modification and adjustment of the weights to obtain better input data clustering results (DeMers & Cottrell, 1993; Hinton & Salakhutdinov, 2006; Kamyshanska & Memisevic, 2013; Van der Maaten, et al., 2009). Smoothing the weights and pruning the weak weights are considered in this phase.

- Smoothing the Weights: As discussed in Section 3.7.1, noise is a random error or variance in a measured variable, and data smoothing is usually used to remove noise. Several techniques can be used to have smooth, flexible and robust parameters of the FFNN clustering tasks. These techniques include smoothing the amounts of the weights or thresholds to improve speed, accuracy, and training capability (Gui, et al., 2001; Jean & Wang, 1994; Peng & Lin, 1999; Tong, et al., 2010). *Mid-range* is a popular smoothing technique (Gui, et al., 2001; Jean & Wang, 1994). If some attributes of the input data have extremely high values of weight, their thresholds will be high, and they may affect the related data vector threshold and the data clustering results. Therefore, when some components of the *SW* vector are extremely higher than the other components, the *HighMidRange* technique can be used. To identify the *HighMidRange*, the *Mid-Range* of the *SW* vector components is computed, as shown in Equation (5.6).

$$MidRange(SW) = {(SW_{max} + SW_{min})}/{2} \tag{5.6}$$

SW_{max} is a component of the *SW* vector with maximum value. Consequently, the average of all components of the *SW* which are bigger than *MidRange(SW)* as $Averagemax$ are computed. Finally, the *HighMidRange* of the *SW* components is calculated as shown in Equations (5.7).

$$HighMidRange = {(SW_{max} + Averagemax)}/{2} \tag{5.7}$$

In the *HighMidRange* technique, if some components of the *SW* vector are higher than the *HighMidRange*, the model fixes their weights to the *HighMidRange* value.

Figure 5-4 shows, how the proposed model smooths the *SW* components of the breast cancer Wisconsin (original) dataset (Asuncion & Newman, 2007) by using the RUFFNN method. The *HighMidRange* of the *SW* components is equal 0.136665. The RUFFNN method changes the amount of SW_6 from 0.141244 to 0.136665 .

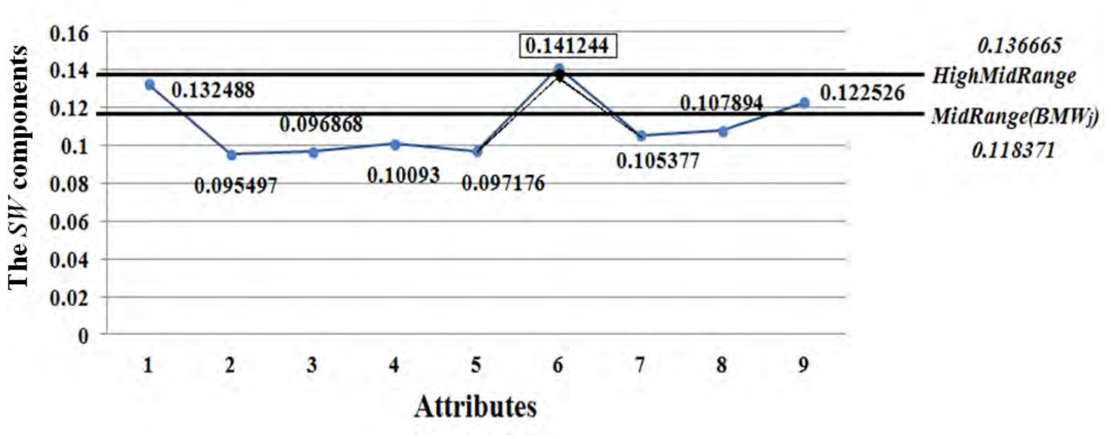


Figure 5-4: Smoothing the *BMW* components of the breast consin (original) dataset by using the RUFFNN method

- Data dimension reduction: datasets that have a high dimension and large number of data instances are difficult to handle and control the noise. Whereas pruning would cause loss of data details (Deng & Kasabov, 2003; Hinton & Salakhutdinov, 2006; Kohonen, 2000; Van der Maaten, et al., 2009). The RUFFNN method reduces the dimension of data by recognizing weak and ineffective components of the *SW* and removing the related attributes during the clustering task. The effects of this data dimensionality reduction technique are high speed of clustering and low network memory usage complexity due to the deletion of unnecessary attributes and related information (Chattopadhyay, et al., 2011; Hinton & Salakhutdinov, 2006; Jolliffe, 1986). If some components of the *SW* vector are close to zero, the method considers the values of these weights as zero and does not consider the attributes of the data point related the weak weights during clustering. Hence, the weights can be controlled and pruned in advance. Equation (5.8) shows that if SW_j is close to zero, function $Threshold(SW_j)$ will be considered as zero.

$$\lim_{SW_j \to 0} Threshold(SW_j) = 0 \tag{5.8}$$

3) **Single Layer Unsupervised Feedforward Neural Network Clustering:** The main section of the structure of RUFFNN model is a single layer feedforward neural network topology to cluster the data of the input instances by using normalized values

and the components of the SW vector. The topology is very simple, as illustrated in Figure 5-1 the number of layers and unit of nodes are clear, including just one input layer with n nodes (which is the same as the number of attributes) and an output layer with just one node. The units of the input layer are fed by the normalized data values from the data preprocessing stage of the RUFFNN model. Each unit has a related weight component SW_j of the SW vector. The output layer has one unit with a weighted sum function for computing the actual desired output. The training of the RUFFNN is carried out in just one iteration and is based on real weights, without any weight updating and error function such as the mean square error. The threshold or output is computed by using normalized values of input instances and the SW vector. Since the mean of the weights was used for computing the SW, the range and properties of the input values of instances cannot dominate the values of the thresholds. The exclusive distance threshold of T_i of the actual output unit is computed by the weighted sum function similar to Hebbian learning but in only one training epoch. Equation (5.9) shows the real threshold T_i for each input instance vector of X_i:

$$T_i = \sum_{j=1}^{m} X_{ij} \cdot SW_j \tag{5.9}$$

The threshold of each data point shows the distance between each data point and the central gravity of the matrix of input values. Each input instance, or data point has an exclusive and individual threshold. The art of the RUFFNN method is in finding the exclusive threshold for each input instance for better clustering results. Figure 5-5shows an example of the distribution of the normal attributes of data vector. Each X_{ij} has its own SW_i ratio to the gravity centre of the dataset.

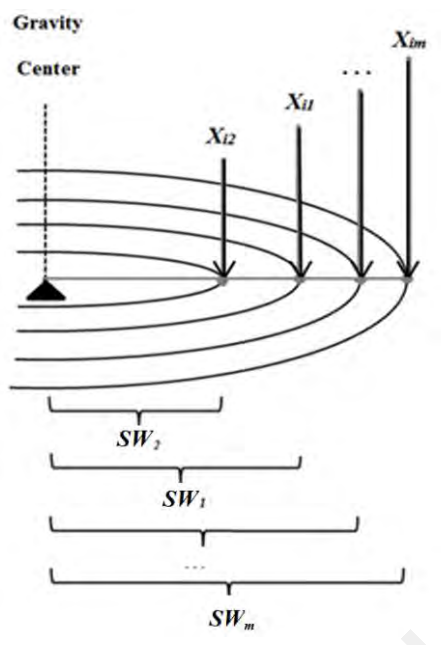


Figure 5-5: Distribution of the normal input data attributes and their distances from the gravity centre of the dataset

Some of the advantages of using exclusive thresholds in the RUFFNN clustering model:

The RUFFNN has the capability of recognizing isolated input data point through the solitary thresholds T_i . The threshold of an isolated data point is further from the thresholds of other clustered data point. Therefore, the data point lies out of the locations of other clusters. The proposed model considers these data points as noise and deletes them. The noise in the data cause difficulty in the clustering process, and recognizing the special property of each attribute and finding its related cluster will be difficult. Therefore, the action of deleting the noise speeds up and improve the accuracy of clustering and results in low memory usage of the network.

The RUFFNN clustering method groups the data points with similar thresholds into one cluster. For each data point, the model searches all clusters to find a suitable cluster with the thresholds of input instances similar or near to the threshold of the data point. Each input instance, has a distinct and special threshold. Data points should not have similar thresholds. The data points with similar thresholds are assumed to the noise.

Figure 5-6a and b shows the Iris dataset from UCI repository which is clustered to three clusters based on their distances to the gravity centre of the dataset or in the other word their thresholds. We can see the 10_{th} input data point has T_{10} equal 0.009907566 and lies inside of the cluster 3 or the cluster of the "Iris Virginica". Therefore, the proposed method is able to learn the number of clusters and their densities without having any constraint and parameter for controlling the clustering tasks based on the thresholds; and generates the clusters after just one epoch.

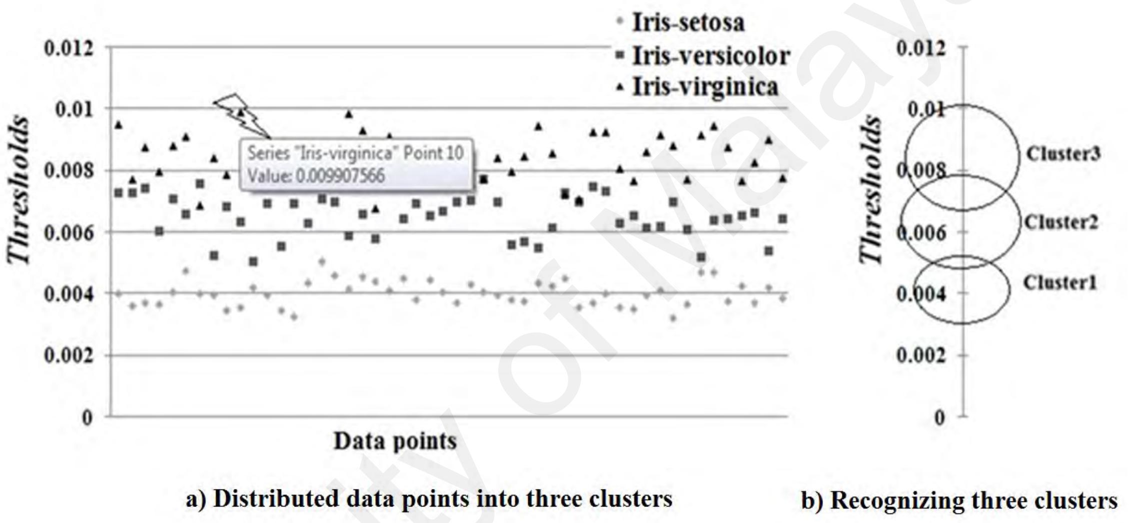


Figure 5-6: The outlook of clustering the Iris dataset by RUFFNN before using class labels

As a result, the essential feature of the proposed method is, computing the *SW* vector as the extract of the code book of weights without using random values or random parameters, without updating weights, or computing any error function. Unlike the current UFFNN methods, the training stage of the RUFFNN method must be performed completely to compute the *SW*, but during testing, the model just uses the vector of the *SW* to generate the total thresholds and immediately clusters the input data. This property affects the speed, accuracy and memory usage of the network. The proposed method is able to reduce the data dimension by deleting the attributes with weak weights SW_j and to recognize the isolated data points as the noise by deleting the isolated input data with solitary thresholds. The RUFFNN method is a linear clustering method and has time complexity and memory complexity of *O(n.m)* and *O(n.m.s_m)* where the parameters *c, k, n, m, s_m* are the number of epochs, clusters, nodes, attributes and size of each attribute. The clustering is carried out in just one iteration.

5.3 The Real Semi-supervised Feedforward Neural Network Clustering

As mentioned in the Section 3.3, there is a technique of converting a clustering method to semi-supervised clustering by considering some constraints or user guides as feedback from users. As discussed in the literature, the RSFFNN clustering is able to improve the accuracy of the RUFFNN clustering result by applying class labels as partial supervision. Figure 5-7 shows a modified section of the design of the RUFFNN clustering at the end of its algorithm, in order to convert the RUFFNN clustering method to the real semi-supervised feedforward neural network (RSFFNN) model,

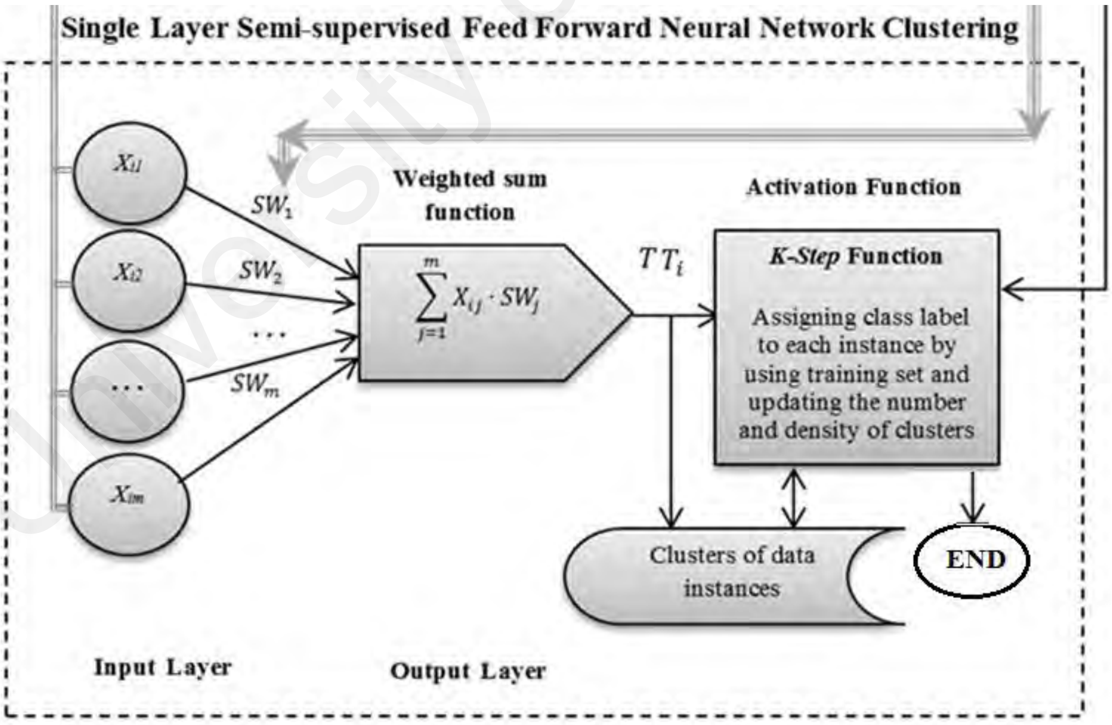

Single Layer Semi-supervised Feed Forward Neural Network Clustering
X_{i1}
X_{i2}
...
X_{im}
SW_1
SW_2
...
SW_m
Weighted sum function
$\sum_{j=1}^{m} X_{ij} \cdot SW_j$
TT_i
Activation Function
K-Step Function
Assigning class label to each instance by using training set and updating the number and density of clusters
Clusters of data instances
END
Input Layer
Output Layer

Figure 5-7: The design of changed section of the RUFFNN suitable for real semi-supervised feedforward neural network clustering method

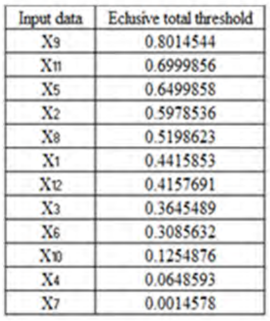
The RSFFNN assigns a class label to each unlabeled data by considering a linear activation function and the exclusive threshold. Through the use of *K-step* activation function (Alippi et al., 1995) or threshold function as a linear activation function for transformation of input values, the model considers the exclusive threshold of each input instance and the related class label. Consequently, based on *K* class labels, and exclusive thresholds in the training set, the proposed model expects *K* clusters and for each cluster considers a domain of thresholds. The model moves unlabeled input instance with a special threshold to the related cluster. Hence, the model updates the number of clusters and the density of each cluster by using class labels. Figure 5-8 shows the algorithm of the RSFFNN clustering method.

```
RSFFNN ( )
Input :  Online input data $X_i$;
Output: Semi-clusters of data;
Begin
{
Call RUFFNN();
// Semi-clustering  by using K-Step activation function
{
        Assign class label to each data node with similar total threshold by using training set;
        Prediction the class label to unlabeled data nodes;
        Updating the number of clusters and density of each cluster;
        }
        Output results;
         End;
```

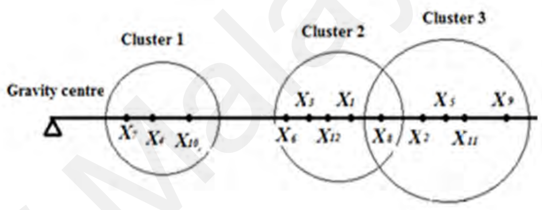
Figure 5-8: The algorithm of the real semisupervised feedforward neural network clustering method

On the other hand, in special cases such as prediction of survival time using the breast cancer dataset, the proposed model is able to consider additional techniques in order to manage the clustering process. If the threshold of an instance is not matched with any thresholds domain in any clusters then the input instance is considered as an unobserved or unknown data. There are several ways to predict the class label for unobserved data. For example, some authors consider two unsupervised and supervised neural network models such as a combination of the SOM and the BPN for the prediction of class labels of the unobserved data (Larochelle, et al., 2009; Larochelle, et al., 2012). In order to predict the class label for the unobserved data, we proposed a

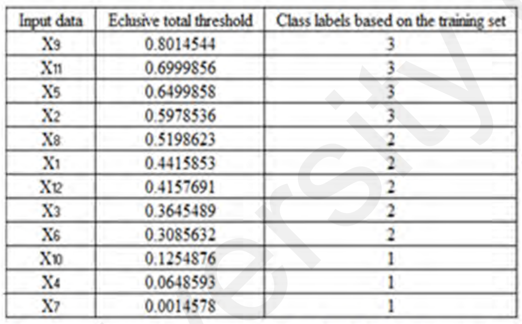
"Trial and Error" method. The class label of each unknown observation is signed and predicted based on the *K-step* function and the related cluster and thresholds domain of the cluster where the input instance is there. When, the semi-clustering accuracy is measured for example by *F-measure* function with 10 fold cross-validation, the accuracy will show the validation of the prediction. Figure 5-9a shows an example of clustering the data by the RUFFNN, and Figure 5-9b shows the improved result by the RSFFNN.



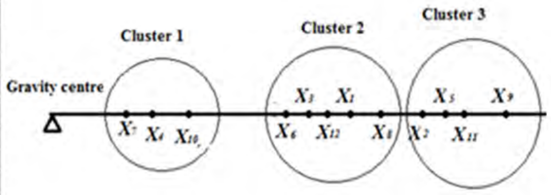
Input data	Eclusive total threshold
X9	0.8014544
X11	0.6999856
X5	0.6499858
X2	0.5978536
X8	0.5198623
X1	0.4415853
X12	0.4157691
X3	0.3645489
X6	0.3085632
X10	0.1254876
X4	0.0648593
X7	0.0014578



a) Clustering by the RUFFNN model



Input data	Eclusive total threshold	Class labels based on the training set
X9	0.8014544	3
X11	0.6999856	3
X5	0.6499858	3
X2	0.5978536	3
X8	0.5198623	2
X1	0.4415853	2
X12	0.4157691	2
X3	0.3645489	2
X6	0.3085632	2
X10	0.1254876	1
X4	0.0648593	1
X7	0.0014578	1

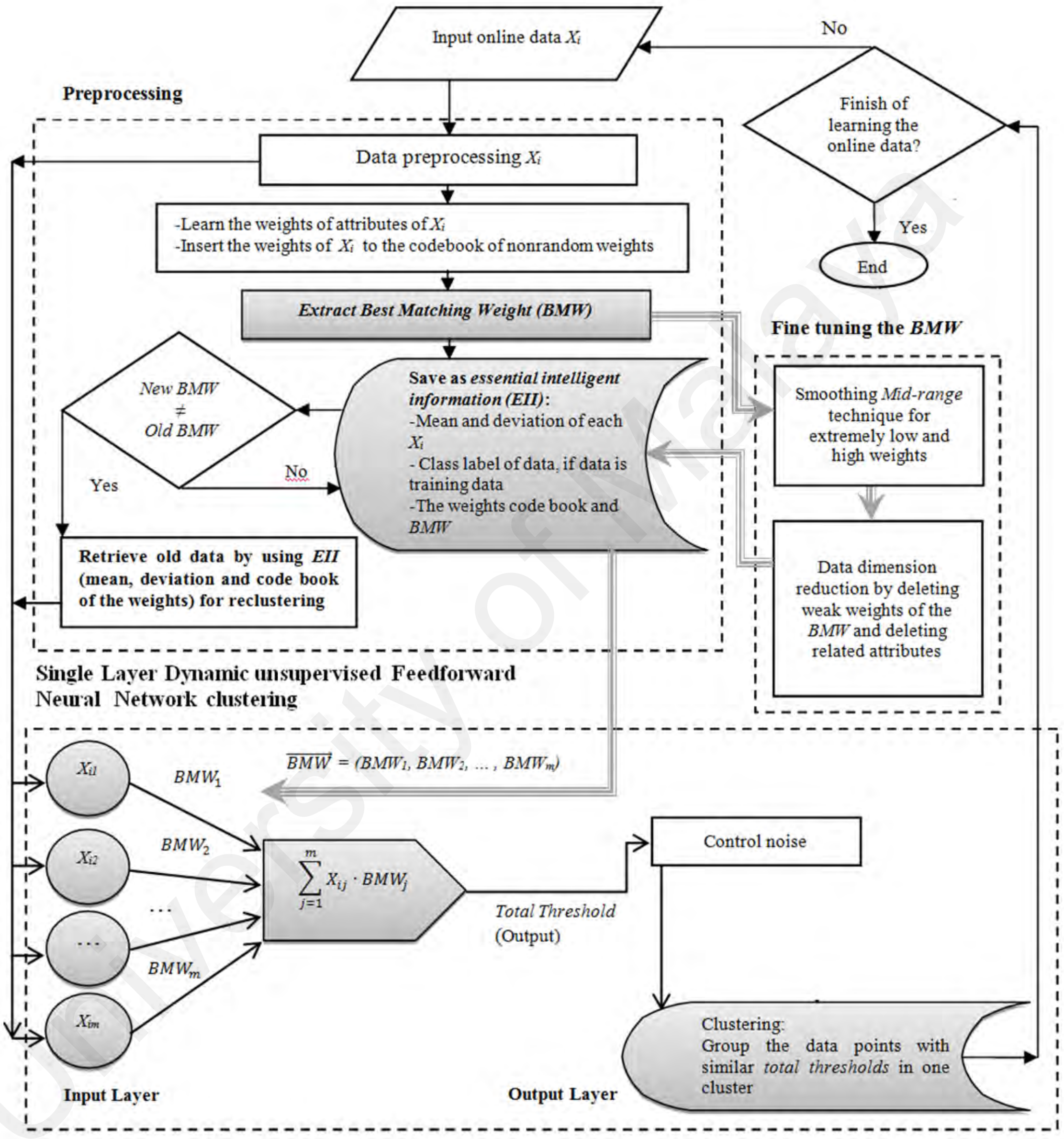


b) Semi-clustering by the RSFFNN model

Figure 5-9: An example of the real semi-supervised feedforward neural network clustering method

5.4 The Dynamic Unsupervised Feedforward Neural Network Clustering

In order to overcome the problems of the ODUFFNN clustering methods as discussed in Section 1.2, we developed dynamic unsupervised feedforward neural network (DUFFNN) clustering method. For this purpose, the RUFFNN clustering method is structurally improved. The DUFFNN model updates its structure, connections and knowledge through learning the online input data dynamically. The DUFFNN model starts without any random parameters or coefficient value which needs predefinition. Figure 5-10 shows the design of the DUFFNN clustering method.



Input online data X_i
No
Preprocessing
Finish of learning the online data?
Data preprocessing X_i
-Learn the weights of attributes of X_i
-Insert the weights of X_i to the codebook of nonrandom weights
Yes
End
Extract Best Matching Weight (BMW)
Fine tuning the BMW
New BMW ≠ Old BMW
Save as essential intelligent information (EII):
-Mean and deviation of each X_i
- Class label of data, if data is training data
-The weights code book and BMW
Smoothing Mid-range technique for extremely low and high weights
Yes
No
Retrieve old data by using EII (mean, deviation and code book of the weights) for reclustering
Data dimension reduction by deleting weak weights of the BMW and deleting related attributes
Single Layer Dynamic unsupervised Feedforward Neural Network clustering
$\overrightarrow{BMW} = (BMW_1, BMW_2, \dots, BMW_m)$
X_{i1}
BMW_1
X_{i2}
BMW_2
$\sum_{j=1}^{m} X_{ij} \cdot BMW_j$
Control noise
...
Total Threshold (Output)
BMW_m
X_{im}
Clustering: Group the data points with similar total thresholds in one cluster
Input Layer
Output Layer

Figure 5-10: The design of the dynamic unsupervised feedforward neural network clustering method.

As shown in Figure 5-10, the DUFFNN method includes two main sections: the preprocessing section, and the single layer dynamic semi-supervised feedforward neural network clustering section. In the preprocessing section, the DUFFNN method as an incremental ODUFFNN method considers a part of the memory called e*ssential important information (EII)* and initializes the *EII* by learning the important information about each online input data in order to store and fetch them, during training, without storing any input data in the memory. The code words of non-random weights are generated by training current online input data just in one epoch, and are inserted in the weights code book. Consequently, the unique standard vector called best matching weight (*BMW*) is mined from the code book of the weights and stored as the *EII* in the memory. The single layer of the DUFFNN clustering applies normalized data values, and fetches some information through the *EII* such as the *BMW* from the section of the preprocessing and generates thresholds and clusters the data nodes. The topology of the single layer dynamic semi-supervised feedforward neural network clustering model is very simple with incremental learning, as shown in Figure 5-10; it contains one input layer with *m* nodes and one output layer with just one node without any hidden layer. The output layer has one unit with a weighted sum function for computing the actual desired output.

5.4.1 Analysis of the DUFFNN Clustering Method

In this section, we illustrate the complete algorithm of the dynamic unsupervised feedforward neural network (DUFFNN) clustering method, and explain the details of the proposed clustering method as shown in Figure 5-11 step by step.

```
Algorithm: The DUFFNN clustering
Input : Online input data $X_i$;
Output: Clusters of data;
Initialize the parameters:
Let $X$: Data node domain;
Let $newMin_i$ : Minimum value of the specific domain of [0,1] which is zero;
Let $newMax_i$ : Maximum value of the specific domain [0,1] which is one;
Let $i$ : Current number of the online data node $X_i$;
Let $j$ : Current number of the attribute;
Let $X_i$: $i_{th}$ current online input data node from domain of $X$;
Let $D$: Retrieved old data node from the memory;
Let $f$ : Current number of the received data node;
Let $n$ : Number of received data nodes;
Let $m$ : Number of attributes;
Let $W_{ij}$ : Weight of attribute $j$ of $i_{th}$ current online data node $X_i$;
Let $\overrightarrow{Prod}$: A vector of the weights product of each attribute of the weights code book;
Let $Prod_j$ : $j_{th}$ component of the $\overrightarrow{Prod}$ vector which $\overrightarrow{Prod} = (Prod_1, Prod_2, \ldots, Prod_m)$;
Let $SumBMW$: a variable for storing the sum of the components of the $BMW$ vector;
Let $\overrightarrow{BMW}$: Best matching weight vector which $\overrightarrow{BMW} = (BMW_1, BMW_2, \ldots, BMW_m)$;
Let $BMW_j$ : $j_{th}$ component of the $\overrightarrow{BMW}$ vector;
Let $BMW_{jOld}$ : $j_{th}$ component of the old $\overrightarrow{BMW}$ vector;
Let $BMW_{jNew}$ : $j_{th}$ component of the new $\overrightarrow{BMW}$ vector;
Let $T_{ij}$: *Threshold* of attribute$_j$ of i$_{th}$ current online data node $X_i$;
Let $TT_i$ : *Total threshold* of $i_{th}$ current online data node $X_i$;
Let $T_{fj}$: *Threshold* of *attribute$_j$* of $f_{th}$ received data node;
Let $TT_f$ : *Total threshold* of $f_{th}$ received data node;
**Method:**
While the termination condition is not satisfied
{
Input a new online input data $X_i$;
//**1- Preprocessing**
// Data preprocessing of $X_i$ based on MinMax technique
        $\forall j=1$ to $m$
            $X_{ij} = \left( {X_{ij} - Min(X_{ij})} \Big/ {Max(X_{ij}) - Min(X_{ij})} \right) \times (newMax - newMin) + newMin$
  **// Compute the weight code words of the current online data $X_i$ and update the code book**
  // Compute the *standard normalize distribution* (*SND*) of the current online input data of $X_i$ based
     on $\mu_i$ and $\sigma_i$ which are mean and standard deviation of $X_i$ :
   $\forall j=1$ to $m$
    {
      $SND(X_{ij}) = {(X_{ij} - \mu_i)} \big/ {\sigma_i}$ ;
      // Consider $W_{ij}$ as weight of $X_{ij}$ equal $SND(X_{ij})$
      $W_{ij} = SND(X_{ij})$ ;
      Insert $W_{ij}$ into the weights code book;
      // Update $\overrightarrow{Prod}$ by considering $W_{ij}$
      $Prod_j = Prod_j \, . W_{ij}$;
    }
   // **Extracting the best matching weight *(BMW)* vector** Extract the global geometric mean
      vector of the code book of nonrandom weights as the *best matching weight vector* (*BMW*)
    $\forall j=1$ to $m$
       $BMW_j = Prod_j^{1/n}$ ;
     $SumBMW = \sum_{j=1}^{m} BMW_j$ ;
     $\forall j=1$ to $m$
       $BMW_j = Round\left(\left({BMW_j} \big/ {SumBMW}\right), 2\right)$ ;
  // **Update essential intelligent information (*EII*):** Store the weights, mean and standard
       deviation of $X_i$ as the *EII* in the memory
    {
    *Memory(EII)* $\leftarrow$ *Store* $(W_{ij}, \mu_i, \sigma_i)$ ;
      *Memory(EII)* $\leftarrow$ *Store* $(\overrightarrow{BMW}, \overrightarrow{Prod})$ ;
    If $X_i$ is from training data and has class label
      {
       *Memory(EII)* $\leftarrow$ *Store (class label of* $X_i$*)*
      }
    }
//**2- Fine-tuning through two techniques**
  // a) Smooth the components of the *BMW* vector
     $\forall j=1$ to $m$
       *Mid-range* $(BMW_j)$ ;
```

```
 // b) Data dimension reduction
       Delete attributes with weak weights of the $BMW_j$ , that are close to zero ;
//3- Single layer DUFFNN clustering

       Fetch ($\overrightarrow{BMW}$) ←Memory(EII);

       // Compute the exclusive total threshold of just $X_i$ based on the new BMW

              ∀ j=1 to m
                $TT_i = TT_i + X_{ij} . BMW_j$;
                If $X_i$ is from training data and has class label
                   {
                   Memory(EII) ← Store (class label of $X_i$ , $TT_i$) ;
                   }
       // If new BMW is different with old BMW, the model fetches $W_{fj}$ , $\sigma_f$ and $\mu_f$ as the EII from memory
            and updates thresholds of just the changed attributes, and update the exclusive total threshold
            of related data point
       If ($\overrightarrow{BMW}_{New} \neq \overrightarrow{BMW}_{Old}$ )

          {
           ∀ j=1 to m
             If $BMW_{jNew} \neq BMW_{jOld}$
                ∀ f=1 to n-1
                   {
                   Fetch ($W_{fj}$ , $\sigma_f$, $\mu_f$) ←Memory(EII);
                     $D_{fj}=(W_{fj} . \sigma_f)+ \mu_f$;
                     $T_{fj\,Old} = D_{fj} . BMW_{jOld}$ ;
                    $TT_f = TT_f - T_{f\,Old}$;
                     $T_{fj\,New} = D_{fj} . BMW_{jNew}$ ;
                    $TT_f = TT_f + T_{fNew}$ ;
                   }
             Memory(EII) ← Update list of thresholds and related class labels;
          }
       // Recognize and delete noise
          Delete isolated input data with solitary thresholds TT;
       // Clustering
          {
          Group the data points with similar thresholds ($TT_i$) in one cluster;
          Learn and generate optimized number of clusters and their densities;
          }
          If learning is finished
             {Output results;
             End;
             }
          Else { Continue to train and cluster next online input data node
               }
}
```

Figure 5-11: The algorithm of the dynamic unsupervised feedforward neural network clustering

The DUFFNN clustering method involves several phases:

1) **Preprocessing:** The DUFFNN clustering method, as opposed to the RUFFNN, applies the preprocessing method which is suitable for online input data preprocessing. The preprocessing phase of the DUFFNN clustering method is a combination of data preprocessing and computing the weight code words of the current online data X_i and update the code book:

- Data Preprocessing: As shown in Figure 5-11, the DUFFNN method considers the *MinMax* normalization technique according Equation (5.1), which is suitable for online input data preprocessing, and is independent of the other data points (Han & Kamber, 2011). The *MinMax* normalization technique is used to transform an input value of each attribute to fit in specific range such as [0,1], where X_{ij} is j_{th} attribute value of online input data X_i, and has special range and domain, in which $Min(X_{ij})$ is the minimum value in this domain, and $Max(X_{ij})$ is the maximum value in this domain. The $newMin_i$ is the minimum value of the specific domain of [0,1] which is equal to zero, and $newMax_i$ is the maximum value of the domain which is equal to one. After transformation of current online input data, the model continues to learn current data in the next stages.
- Compute the Weight Code Words of the Current Online Data X_i and Update the Code book**:** The DUFFNN method creates a code book of nonrandom weights by entrance of first online input data and consequently completes the code book by inserting the code words of each future online input data. In this stage, the proposed model computes the mean μ_i of the normalized current online input data X_i. Then, the standard deviation σ_i of the input data of X_i is computed by considering μ_i . Table 5-2 illustrates this matter.

Table 5-2: The online input data *X*

The online input data X						
Input data vector X_i	Attribute$_1$	Attribute$_2$	…	Attribute$_m$	Mean	Standard deviation
X_i	X_{i1}	X_{i2}	…	X_{1m}	μ_i	σ_i

Each weight vector of the code book is computed based on the *SND* of each online input data value of X_i as shown in Equation (5.2) and (5.3), similar to the RUFFNN clustering method. The weights of attributes of current input data X_i are inserted in the weights code book as the code words of X_i. The model considers a vector of the weights as the product of each attribute of the weights code book. The $Prod$ vector consists of the components $Prod_j$ for the attributes which is computed by the product of the weights of each attribute of the code book. The parameter n is the number of received data nodes, which were trained by the model, i is current number of the online input data node X_i; m is the number of attributes and j is the current number of the attribute. The Equations (5.10) and (5.11) show these relationships.

$$Prod_j = Prod_j \,.\, W_{ij} \tag{5.10}$$

$$\overrightarrow{Prod} = (Prod_1, Prod_2, \ldots, Prod_m) \tag{5.11}$$

- Extracting the Best Matching Weight Vector: The DUFFNN learns the standard unique weight as *best matching weight (BMW)* similar to the RUFFNN clustering method. The *BMW* vector is the global *geometric mean* (Vandesompele et al., 2002; Jacquier et al., 2003) vector of the code book of the non-random weights, and is computed based on the centre of gravity of the current and last trained data nodes. In the DUFFNN method, the code book of real weights is initialized by considering properties of input values directly and without using any random values or random parameters similar to the RUFFNN clustering method. The RUFFNN computes the standard weight once through processing of all input data instances in the input matrix, however, the DUFFNN dynamically computes the *BMW* based on the gravity centre of the current and last trained data nodes and with the entrance of each online input data, the BMW is updated. The BMW vector is computed by Equations (5.12), (5.13), (5.14) as follows:

$$BMW_j = Prod_j^{1/n} \tag{5.12}$$

$$BMW_j = Round\left(\left({BMW_j}/_{SumBMW}\right), 2\right) \tag{5.13}$$

$$SumBMW = \sum_{j=1}^{m} BMW_j \tag{5.14}$$

$$\overrightarrow{BMW} = (BMW_1, BMW_2, \ldots, BMW_m) \tag{5.15}$$

Equations (5.14) and (5.15) => $\sum_{j=1}^{m} BMW_j = 1$ (5.16)

The parameter *n* is the number of received data nodes (contains of the old data points and current online input data node X_i), m is the number of attributes and j is the current number of attribute. In Equation (5.14), the parameter *SumBMW* is equal to the sum of the components of the BMW vector. Equation (5.15) shows the BMW_j is the global geometric mean of all W_{ij} of the attributes X_i. As shown in Equation (5.13), the model considers *Round* function with 2 digits of each BMW_j ratio to *SumBMW*, because in this way, the model is able to control the changing BMW_{jNew} ratio to BMW_{jOld} . This technique affects the low time complexity and low memory complexity of the model, because the model just updates the thresholds related to changed BMW_j . Equation (5.16) shows the sum of components BMW is equal to one through Equations (5.14) and (5.15), therefore, we can understand the distribution of weights between attributes. Table 5-3 illustrates the code book of the weights and the process of extracting the BMW vector.

Table 5-3: The code book of the weight vectors and the *BMW* vector

The code book of weight vectors				
Weight vector of X_i	Attribute$_1$	Attribute$_2$	…	Attribute$_m$
Weight vector of X_1	W_{11}	W_{12}	...	W_{1m}
Weight vector of X_2	W_{21}	W_{22}	...	W_{2m}
…	...	...	...	...
Weight vector of X_n	W_{n1}	W_{n2}	...	W_{nm}
$\overrightarrow{Prod}$	$Prod_{1}$	$Prod_{2}$	...	$Prod_{m}$
$\overrightarrow{BMW}$	BMW_1	BMW_2	...	BMW_m

As mentioned, the difference of Table 5-3 from Table 5-1 is in the dynamic computing and updating of the code book of the weights and the BMW vector. Therefore, the weights code book of the DUFFNN clustering method unlike the RUFFNN method is not a static matrix. The main goal of the DUFFNN model is learning of the *BMW* vector as the criterion weight vector. The next stages will show how the threshold of current online data is computed, the current online input data is dynamically clustered based on the computed *BMW* vector, and consequently the network of clusters is updated.

- Update Essential Intelligent Information (*EII)*: In this stage after learning the weights of the attributes of online input data X_i, the model stores the mean and standard deviation of X_i, and the new BMW and $Prod$ as the *EII* in the memory. Some data nodes that acts as training data have a class label, therefore in this phase if the current online input data has class label, consequently, the model keeps it with other important information of the data in the memory. After clustering, the DUFFNN model similar to the RUFFNN model is able to improve to the DSFFNN model. Therefore, the model will consider the class label and its related total threshold in order to semi-supervise clustering of the data in the future stages.

2) **Fine Tuning:** The DUFFNN, like the RUFFNN clustering method, applies the techniques of fine tuning, but after each updating the *BMW* process. As mentioned

earlier, in order to adapt the weights accurately to achieve better results of clustering the data points, the two techniques of smoothing the weights and pruning the weak weights can be considered:

- Smoothing the weights: The speed, accuracy and capability of the training of the feedforward neural network clustering can be improved, by applying some techniques such as smoothing parameters and weights interconnection (Jean & Wang, 1994; Peng & Lin, 1999; Gui et al., 2001; Tong & Zhou, 2010). In *Mid-range* technique, which is an accepted smoothing technique (Jean & Wang, 1994; Gui et al., 2001), the average of high weight components of the *BMW* vector is computed and considered as middle range (*Mid-range*). The input data attributes with too high weight values may cause them to dominate the high thresholds and affect the clustering results. When some BMW_j are considerably higher than other components, the BMW_j can be smoothed based on the *Mid-range* smooth technique. If the weights of some components of the *BMW* are higher than the *Mid-range*, the model will reset their weights to be the same as *Mid-range* value.
- Data dimensionality reduction: The DSFFNN model can reduce the dimension of data by recognizing the weak weights BMW_j and deleting the related attributes. The weak weights which are close to zero, are less effective on thresholds and the desired output. Hence, the weights can be controlled and pruned in advance.

3) **Single Layer Dynamic Unsupervised Feedforward Neural Network Clustering:** The main section of the DUFFNN clustering model is a single layer FFNN topology to cluster the online data by using normalized values and the components of the *BMW* vector. The proposed model is able to re-cluster all old data points by retrieving information from the *essential intelligent information (EII)*. The DUFFNN clustering is carried out in one training iteration based on non-random weights, without

any weight update, activation function or error function, such as, the mean square error. The threshold or output is computed by using normalized values of the input data node and the *BMW* vector is fetched from the *EII.* When DUFFNN model learns a huge amount of data for clustering, the new *BMW* changes slowly and to close as to the last computed *BMW*. IF components of the new BMW_j is equal to the components of the old BMW_j, the model just clusters X_i. In this case, computing the threshold of each attribute of current online input data X_{ij} and *total threshold* of the online input data vector X_i are done based on Equations (5.17) and (5.18).

$$T_{ij} = X_{ij}.BMW_j \quad (5.17)$$

$$TT_i = \sum_{j=1}^{m} X_{ij} \cdot BMW_j \text{ or } TT_i = \sum_{j=1}^{m} T_i \quad (5.18)$$

When the current online input data X_i has class label, the DUFFNN model stores the computed total threshold as related TT_i to this class label in the memory, because the DSFFNN clustering needs this class label. If the new *BMW* vector is different from the last *BMW*, the model considers the BMW_{New} vector based on the feature of Hebbian learning and re-clusters all old data points by retrieving information from the *essential intelligent information (EII)*, based on their related total threshold during one iteration. In this case, the model considers Equations (5.19),(5.20),…, (5.23), respectively, and checks which components of BMW_{New} are changed. Consequently, the model fetches related W_{fj}, σ_f, μ_f of each component BMW_{jOld} and retrieves related data node D_{fj} from *EII* based on Equations (5.2) and (5.3), and the computed related threshold of the data node D_{fj}. By considering the total threshold of the D_{fj} as the *EII* from the memory and replacing the values of the old threshold of attribute j of D_{fj} as $T_{f\,Old}$ and new threshold of attribute j of D_{fj} as $T_{f\,New}$, the data node D_{fj} lies in the special place of axis of total threshold and suitable cluster.

$$D_{fj}=(W_{fj}\,.\,\sigma_f)+\ \mu_f \quad (5.19)$$

$T_{f\,Old} = D_{fj} \,.\, BMW_{jOld}$ (5.20)

$TT_f = TT_f - T_{f\,Old}$ (5.21)

$T_{f\,New} = D_{f\,j} \,.\, BMW_{jNew}$ (5.22)

$TT_f = TT_f + T_{fNew}$ (5.23)

Therefore, it is not necessary to compute and update all thresholds of all old data nodes attributes. The single layer DUFFNN computes the total threshold of the current input data node X_i , updates the total thresholds of old data nodes, and list of thresholds and related class labels. Consequently, the model re-clusters all received data nodes. Figure 5-12 illustrates how the data nodes are clustered by the DUFFNN model.

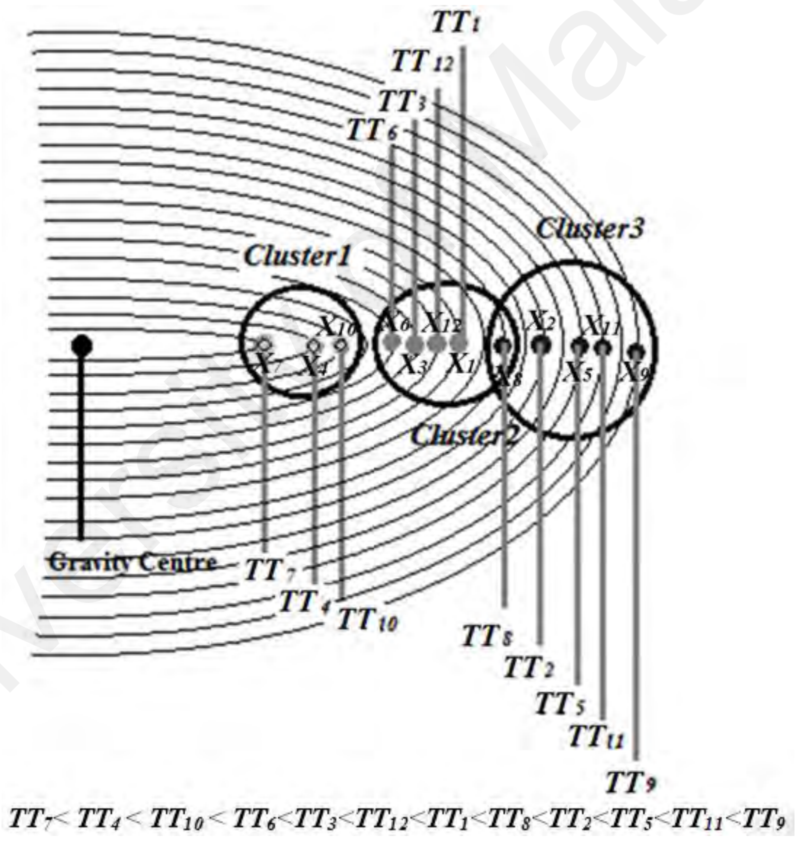


Figure 5-12: An example for clustering the data nodes to three clusters by the DUFFNN

As shown in Figure 5-12, each TT_i is the *total threshold* vector of each data point in ratio to the gravity centre of the data points. Therefore, each data vector takes its own position on the thresholds axis. The online input data X_i, based on its exclusive TT_i, lies on the axis respectively. Each data point has an exclusive and individual threshold. If two data points possess an equal *total threshold*, but are in different clusters, clustering accuracy is decreased because of the error of the clustering method. The DSFFNN considers the data points with close *total thresholds* into one cluster. Figure 5-13 is an example of clustering the Iris data points into three clusters by the DSFFNN clustering. In Figure 5-13a and b, the Iris data from UCI repository is clustered to three clusters based on a unique *total threshold* of each data point by the DSFFNN method. Data point 22 has $TT_{22} = 0.626317059$ and lies inside of cluster 2 or the cluster of the "Iris Versicolour".

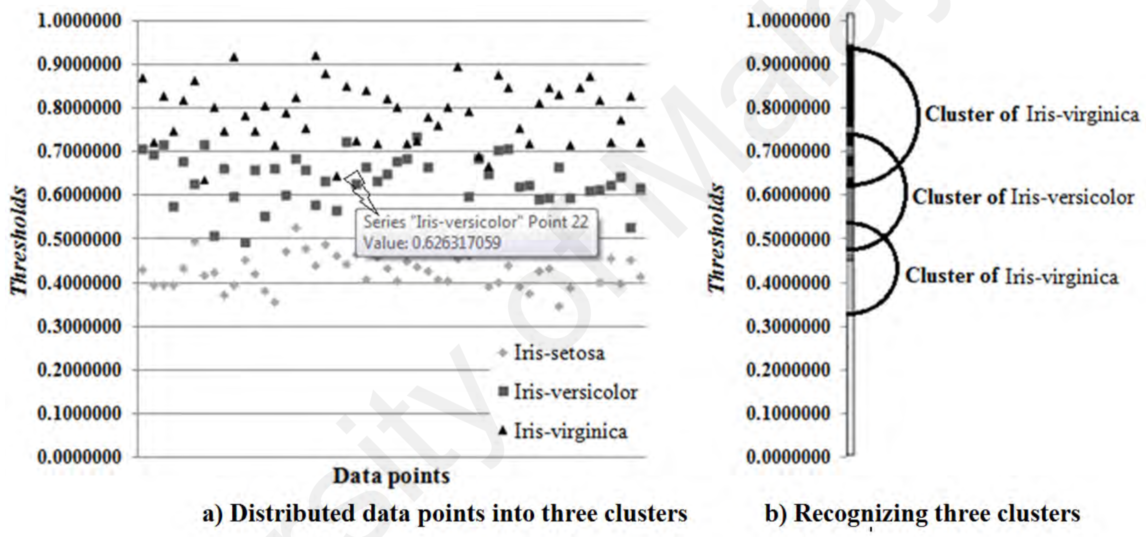
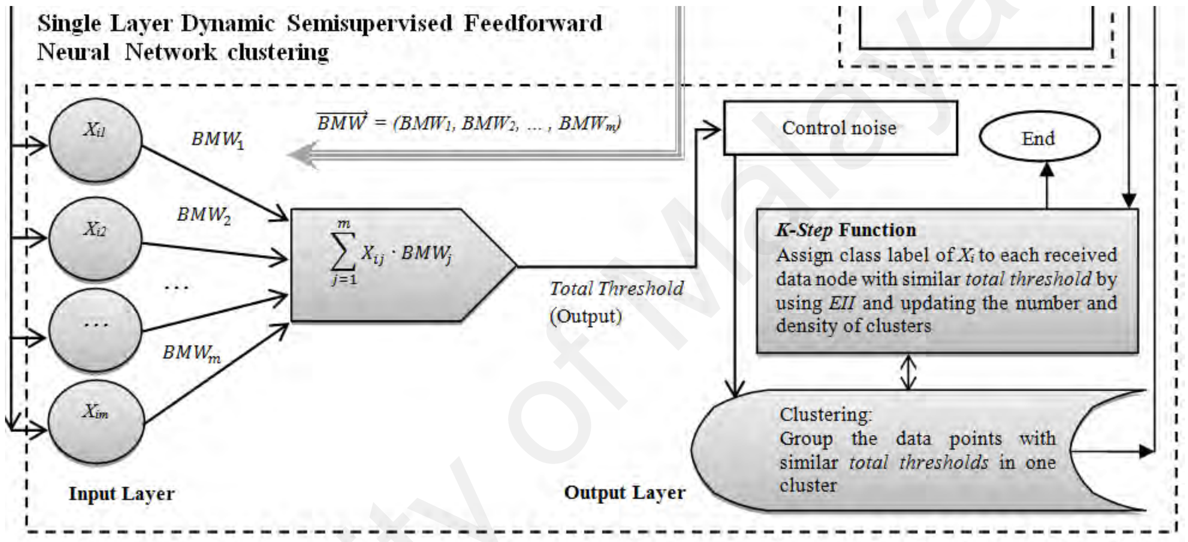


Figure 5-13: The outlook of clustering the online Iris data to three clusters based on a unique total threshold of each data point by the DUFFNN

Pruning the noise: The DUFFNN distinguishes isolated data points through the solitary total thresholds. The *total threshold* of an isolated data point is not close to the *total thresholds* of other clustered data point. Therefore, the isolated data point lies out of the locations of other clusters. The proposed DUFFNN method sets apart these data points as noise and removes them.

5.5 The Dynamic Semi-supervised Feedforward Neural Network Clustering

After clustering the data node by the DUFFNN, the proposed method considers the class label as constraint in order to improve the accuracy of the result of clustering. Figure 5-14 shows a changed section of the design of the DUFFNN clustering related to the dynamic semi-supervised feedforward neural network (DSFFNN) model, which it adds to the end of the DUFFNN clustering method.



Single Layer Dynamic Semisupervised Feedforward Neural Network clustering
$\overrightarrow{BMW} = (BMW_1, BMW_2, \ldots, BMW_m)$
X_{i1}
X_{i2}
$\ldots$
X_{im}
BMW_1
BMW_2
BMW_m
$\sum_{j=1}^{m} X_{ij} \cdot BMW_j$
Total Threshold (Output)
Control noise
End
K-Step Function
Assign class label of X_i to each received data node with similar total threshold by using EII and updating the number and density of clusters
Clustering: Group the data points with similar total thresholds in one cluster
Input Layer
Output Layer

Figure 5-14: The design of changed section of the DUFFNN suitable for dynamic semi-supervised feedforward neural network clustering method

If the current online input data X_i is a training data and has class label, the model fetches the class label and related *total threshold* of X_i from memory as *EII* and assigns this class label to the data nodes with similar threshold and updates the data nodes by using the *K-step* function. Consequently, based on the *K* class labels and related exclusive thresholds in the *EII*, the proposed method expects *K* clusters and for each cluster considers a domain of thresholds. By considering the cluster results of the last phase, if there are some data points with a related threshold in each cluster but without a class label, the model supposes that these data nodes have the same class label as their clusters. During the process of future online data nodes, when the model updates the data nodes, the class labels of these unknown data nodes will be recognized and adjusted for suitable cluster if necessary. Hence, the class label of data is predicted in two ways: 1) during clustering by considering the related cluster, in which the data lies, 2) by using *K-step* function based on the relationship between thresholds and class labels in the *EII*. Figure 5-15 shows the algorithm of the dynamic semi-supervised feedforward neural network (DSFFNN) clustering method.

```
DSFFNN ( )
Input :  Online input data $X_i$;
Output: Semi-clusters of data;
Begin
{
Call DUFFNN();
// Semi-clustering  by using K-Step activation function
        {
          Assign class label to each data node with similar total threshold by using EII;
          Prediction the class label to unlabeled data nodes;
          Updating the number of clusters and density of each cluster;
          }
          Output results;
           End;
Else { Continue to train and cluster next online input data node
                  }
}
```

Figure 5-15: Dynamic semi-supervised feedforward neural network clustering algorithm

The DSFFNN clustering method like the RSFFNN clustering method, in special cases, such as the prediction of survival time, is able to consider the "Trial and Error" method in order to manage the clustering process. Finally, the DSFFNN method is able to update the clusters and their densities based on assigning a class label to each input data by using the *K-step* activation function. The performance of the DSFFNN method is in just during one training iteration, for each online input data.

5.6 The Time Complexity and Memory Complexity of the Purposes

In this section, we present the time complexity and the memory complexity of the proposed methods, based on their algorithms. For example, the RUFFNN briefly clusters the matrix of a dataset in three phases as mentioned in Section 5.2.1, and we calculate the number of necessary processes in front of them, as follows:

1) Preprocessing includes:

Data preprocessing .... ($n.m$)

Create the code book of the weights .... ($n.m$)

2) Fine tuning includes:

Smoothing .... (m)

Reducing attributes .... (m)

3) Process of single layer unsupervised feedforward neural network for clustering input dataset, includes:

Compute the exclusive threshold of each input .... ($n.m$)

Recognize and delete noise .... (n)

Clustering .... ($n.m$)

Therefore, the total number of processes will be $4\ (n.m)+3m+n$, and the time complexity will be $O(n.m)$. Since the process is carried out sequentially during one training iteration on the matrix of the dataset. Also, the memory complexity will be $O(n.m.s_m)$. These complexities are the functions of the longest possible time or memory usage for processing the algorithm in the worst case scenario. The time complexity and the memory complexity of the RSFFNN clustering method are similar to the RUFFNN clustering method, since the RSFFNN is based on the RUFFNN method.

As shown in Table 5-4, the time complexity and the memory complexity of the NG and GNG are $O(c.n^2.m)$ and $O(c.n^2.m.s_m)$, respectively. Also, the time complexity and the memory complexity of the SOM are $O(c.n.m^2)$ and $O(c.n.m^2.s_m)$, respectively. However, the RUFFNN method has better results in the time complexity and memory complexity which are $O(n.m)$ and $O(n.m.s_m)$. Where the parameters *c, k, n, m,* s_m are the number of epochs, clusters, nodes, attributes and size of each attribute.

Table 5-4 shows time complexity and memory complexity of the proposed clustering methods and some related methods.

Table 5-4: Comparison of time complexity and memory complexity of the proposed clustering methods with some related methods

Method	Time Complexity	Memory Complexity
K-means	$O(c.k.n.m)$	$O((n+k).m.s_m)$
NG	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
GNG	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
SOM	$O(c.n.m^2)$	$O(c.n.m^2.s_m)$
Semi-SOM	$O(c.n.m^2)$	$O(c.n.m^2.s_m)$
RUFFNN	$O(n.m)$	$O(n.m.s_m)$
RSFFNN	$O(n.m)$	$O(n.m.s_m)$
ESOM	$O(n^2.m)$	$O(n^2.m.s_m)$
Semi-ESOM	$O(n^2.m)$	$O(n^2.m.s_m)$
ESOINN	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
DSOM	$O(c.n.m^2)$	$O(c.n.m^2.s_m)$
IGNGU	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
EDSOM	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
HI-GNG	$O(c.n^2.m)$	$O(c.n^2.m.s_m)$
DUFFNN	$O(n.m)$	$O(n.m.s_m)$
DSFFNN	$O(n.m)$	$O(n.m.s_m)$

$*$ *note*:
n is the number of data points, *m* is the number of attributes, *K* the number of clusters, *c* is the number of iterations, S_m is size of each attribute.

As shown in Table 5-4, the time complexity and memory complexity of the DSOM are $O\ (c.n.m^2)$ and $O\ (c.n.m^2.s_m)$, respectively. Also, the time complexity and the memory complexity of the ESOM are $O(n^2.m)$ and $O(n^2.m.s_m)$, respectively. However, the DUFFNN method has better results in terms of time complexity and memory complexity, which are *O(n.m)* and *O(n.m.s_m)* in the worst case scenario, based on the time complexity and memory complexity of the RUFFNN. Actually, in order to re-cluster the nodes using the DUFFNN, the model has to update some nodes, and consequently some of their attributes, as mentioned in Section 5.4.1. The DSFFNN and the DUFFNN clustering methods have the same time complexity and memory complexity.

5.7 Summary

In this chapter, we explained the details of the algorithms and designs of the methods, which were proposed in Chapter 4. We specially developed the dynamic unsupervised feedforward neural network (DUFFNN) clustering method, in order to overcome the problems of low training speed, low accuracy as well as high time complexity and high memory complexity in the scope of the research. After the entrance of each online input data, the DUFFNN dynamically computes and stores *Essential Intelligent Information (EII)* of the current data, such as the *best matching weight* (*BMW*) vector. Consequently, a single layer DUFFNN fetches the *BMW* vector and the normalized current data to generate its exclusive *total threshold (TT)* during one training epoch. If the new *BMW* is not equal to the old *BMW* vector, the single layer DUFFNN retrieves the old data nodes by using the *EII* and updates their *TT* based on a new *BMW*. The proposed method is able to control and delete attributes with weak weights to reduce the data dimensions, and data with solitary thresholds in order to reduce noise. The DUFFNN clustering is improved by applying class labels as partial supervision, which is entitled dynamic semi-supervised feedforward neural network (DSFFNN) clustering, to assign a class label to the each unlabeled data by considering a linear activation function and the exclusive threshold for more accurate clustering results. In Chapters 6 and 7, we will introduce the controlled test and investigation of CPU time, accuracy, time complexity and memory complexity of the proposed clustering models using several datasets from the UCI repository, and the breast cancer dataset from the UMMC.

CHAPTER 6: EXPERIMENTAL RESULTS AND EVALUATION ON THE RUFFNN AND RSFFNN CLUSTERING METHODS

6.1 Introduction

In this chapter, we implement the proposed real unsupervised feedforward neural network (RUFFNN) and real semisupervised feedforward neural network (RSFFNN) clustering methods, and evaluate their performances based on measuring the CPU time usage, clustering accuracy, time complexity and memory complexity of each method, on the five datasets obtained from the University of California at Irvine (UCI) Machine Learning Repository (Asuncion & Newman, 2007). Furthermore, the performance of the RUFFNN is compared against some effective related static unsupervised feedforward neural network clustering methods, such as the SOM and the GNG.

In addition, an improved version of the RSFFNN, is also tested on the six datasets from UCI Repository and the breast cancer dataset from the UMMC.

6.2 Experimental Evaluation on the RUFFNN and RSFFNN Clustering Methods

The proposed RUFFNN and RSFFNN clustering methods were tested on the breast cancer-Wisconsin, Iris, Spambase, Arcene and Yeast dataset from UCI Repository (Asuncion & Newman, 2007), as mentioned in Section 4.2.6. The methods were also compared with several current related methods and a popular supervised feedforward neural network method, such as the back propagation network (BPN). Furthermore, the RSFFNN method was performed on the breast cancer dataset of the UMMC, in order to predict the survival time of each patient.

6.2.1 The RUFFNN and RSFFNN Clustering on the Breast Cancer Wisconsin

The breast cancer Wisconsin (original) dataset from the UCI Repository has two classes: benign and malignant related to the malignancy of breast tumours. For computing clustering accuracy of the RUFFNN on the breast cancer Wisconsin dataset, *F-measure* with 10 folds of the test set was considered. The breast cancer Wisconsin dataset has 683 data after cleaning. Therefore, in each performance, we considered 614 data as the training set and 69 data as the test set.

Table 6-1 shows the details of the computation of the *F-measure* in different performances on the test set for computing clustering accuracy. $|t_p|$ is true positives, which refers to the number of the positive objects that are laid in the correct related cluster, $|t_n|$ is true negatives, which refers to the number of the negative objects that are laid in correct related cluster. The error clustering $|f_n|$ is false negative, which refers to the number of the negative objects that are laid in incorrect related cluster, and $|f_p|$ false positive which refers to the number of the positive objects that are laid in the incorrect related cluster.

Table 6-1: The clustering results of the RUFFNN method the breast cancer Wisconsin (Original) dataset

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	35	34	31	32	4	2	91.30	0.939394	0.885714	0.911765
2	44	25	42	25	2	0	97.10	1	0.954545	0.976744
3	48	21	48	20	0	1	98.55	0.979592	1	0.989691
4	29	40	28	37	1	3	94.20	0.903226	0.965517	0.933333
5	35	34	35	32	0	2	97.10	0.945946	1	0.972222
6	56	13	56	12	0	1	98.55	0.982456	1	0.99115
7	52	17	50	17	2	0	97.10	1	0.961538	0.980392
8	54	15	54	15	0	0	100	1	1	1
9	41	28	41	28	0	0	100	1	1	1
10	57	12	57	11	0	1	98.55	0.982759	1	0.991304

As shown in Table 6-1, the average of the evaluated *F-measure* for the 10 folds of the test set using the RUFFNN clustering was 97.47% after just one epoch of training which takes 8.7262 milliseconds.

Table 6-2 and Figure 6-1 show the computed *SW* vector of the breast cancer Wisconsin dataset by the RUFFNN clustering method.

Table 6-2: The *SW* vector of the breast cancer Wisconsin (original) dataset using the RUFFNN method

Vector of the *SW* for the Breast Cancer Wisconsin (original) dataset								
SW_1	SW_2	SW_3	SW_4	SW_5	SW_6	SW_7	SW_8	SW_9
0.150988	0.108832	0.110394	0.115023	0.110745	0.160967	0.120091	0.12296	0.139635

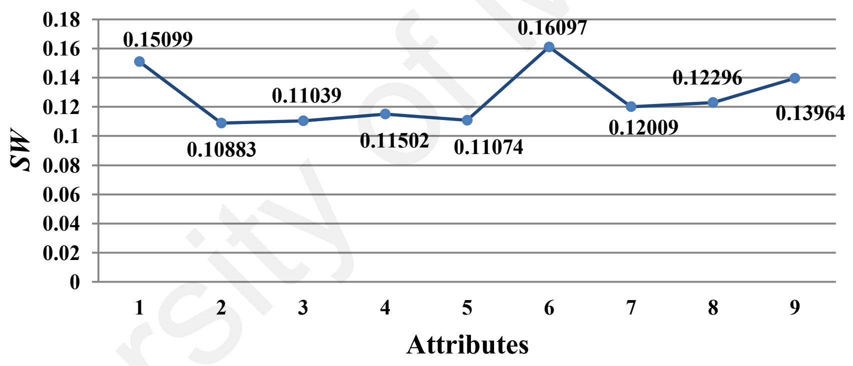


Figure 6-1: The *SW* vector of the Breast cancer Wisconsin (original) dataset using the RUFFNN method

The total thresholds of the input data were computed based on the *SW* vector, and the input data were subsequently clustered. As shown in Figure 6-2, two distinct clusters can be recognized.

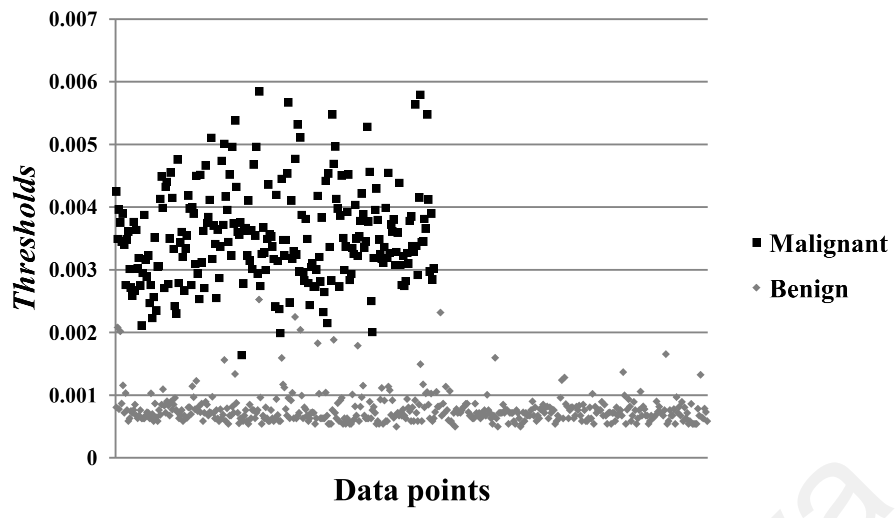


Figure 6-2: The clusters of the RUFFNN method on breast cancer Wisconsin (original) dataset

We compared the results of the proposed RUFFNN method with the results of some related methods. Table 6-3 shows the speed of processing on the basis of the number of epochs and the accuracy on the basis of the density of the *CCN* and the *F-measure* for the breast cancer Wisconsin dataset.

Table 6-3: Comparison of the clustering results on the breast cancer Wisconsin dataset by the RUFFNN, RSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch
K-means	96.19	-	20
Neural Gas	96.19	-	20
GNG	69.84	-	5
SOM	96.63	98.06	20
Semi-SOM	98.39	98.75	20
RUFFNN	97.25	97.47	1
RSFFNN	100	100	1

In Table 6-3 based on the outcomes of the experiment, the density of *CCN* of the *K-means* and the NG methods are 96.19% after 20 epochs (Camastra & Verri, 2005). The density of the *CCN* of the GNG method is 69.84% after 5 epochs (Bouchachia, et al., 2007). Camastra and Verri reported, the SOM produced 96.63% for the density of the *CCN* and 98.06% by using the *F-measure* after 20 epochs. The SOM clustering result was improved by considering class labels like the semi-SOM method, and the accuracy was 98.39% for the density of the *CCN* and 98.75% by using *F-measure*. The accuracy of the proposed RUFFNN clustering method after one epoch was 97.25% for the density of the *CCN* and 97.47% for the *F-measure.* The density of the *CCN* and the accuracy of the *F-measure* of the semi-clustering by the proposed RSFFNN method after just one epoch of training was 100%. While for the BPN, the accuracy by *F-measure* was 99.28% after 1000 epochs of training. All clustering methods show two distinct clusters for this dataset.

6.2.2 The RUFFNN and RSFFNN Clustering on Iris

The Iris dataset from the UCI Repository has three classes: Iris Setosa, Iris Versicolour and Iris Virginica. For computing clustering accuracy of the RUFFNN on the Iris dataset, the *F-measure* for 10 folds of the test set was considered. The Iris dataset has 150 data records. Therefore, in each performance, we consider 135 data as the training set and 15 data as the test set. Table 6-4 shows the details of the computation of the *F-measure* in different performances on the test set for computing clustering accuracy. Parameter i refers to the special class, that $i=\{1,2,3\}$, and j refers to the special cluster, that $j=\{1,2,3\}$. t_{ci} is true objects, which refers to the number of the objects that are laid in the correct related cluster. f_{ci} and f'_{ci} are false objects related to first and second other classes, which refer to the number of the objects that are laid in incorrect cluster.

Table 6-4: The clustering results of the RUFFNN method on the Iris dataset

Test subset	Cluster1			Cluster2			Cluster3			*CCN%*	*F-measure*
	t_{c1}	f_{c1}	f'_{c1}	f_{c2}	t_{c2}	f'_{c2}	f_{c3}	f'_{c3}	t_{c3}		
1	9	3	0	0	3	0	0	0	0	80	0.7619
2	9	1	0	2	3	0	0	0	0	80	0.76905
3	8	0	0	0	6	0	0	0	1	100	1
4	13	0	0	0	2	0	0	0	0	100	1
5	7	0	0	0	5	0	0	0	3	100	1
6	0	0	0	2	6	0	0	0	7	86.67	0.9285
7	0	0	0	0	14	0	0	0	1	100	1
8	0	0	0	0	7	0	0	0	8	100	1
9	0	0	0	0	0	0	0	0	15	100	1
10	0	0	0	0	0	0	0	0	15	100	1

As shown in Table 6-4, the average of *F-measure* for 10 folds of the test set using the RUFFNN clustering was 94.59% after just one epoch of training taking 4.1744 milliseconds.

Table 6-5 and Figure 6-3 show the computed *SW* vector of the Iris dataset by the RUFFNN clustering method.

Table 6-5: The *SW* vector of the Iris dataset using the RUFFNN method

Vector of the *SW* for the Iris dataset			
SW_1	SW_2	SW_3	SW_4
0.158091	0.342448	0.29537	0.204091

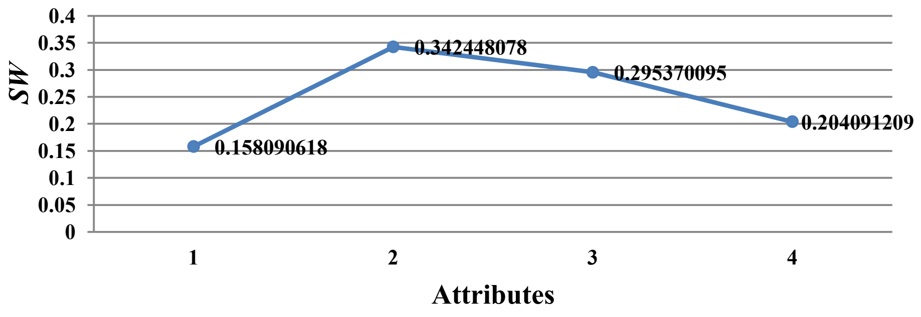


Figure 6-3: The *SW* vector of the Iris dataset using the RUFFNN method

The total thresholds of input data were computed based on the *SW* vector, and the input data were subsequently clustered. As shown in Figure 6-4, three distinct clusters can be recognized.

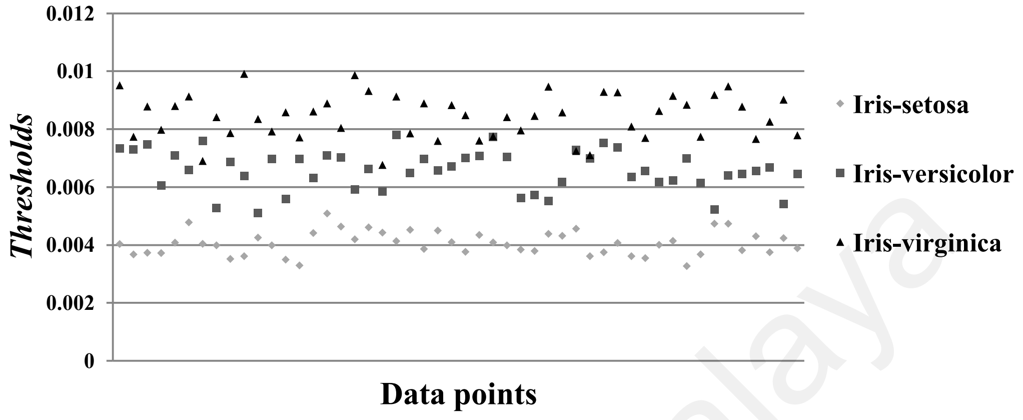


Figure 6-4: The clusters of the RUFFNN method on the Iris dataset

We compared the results of the proposed RUFFNN method with the results of some related methods. Table 6-6 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* for the Iris dataset.

Table 6-6: Comparison of the clustering results on the Iris dataset by the RUFFNN, RSFFNN and some related methods

The clustering method	Density of *CCN* %	*F-measure%*	Epoch
K-means	89.33	-	20
Neural Gas	92.67	-	20
GNG	90.00	-	10
SOM	85.33	85.22	140
Semi-SOM	92.67	92.66	140
RUFFNN	94.67	94.59	1
RSFFNN	100	100	1

In Table 6-6 based on the outcomes of the experiment, the density of *CCN* of the *K-means* and the NG methods are 89.33% and 92.67% respectively, after 20 epochs of training (Camastra & Verri, 2005). The density of *CCN* of the GNG method is 90.00% after 10 epochs (Costa & Oliveira, 2007). The SOM produced 85.33% for the density of the *CCN* and 85.22% by using the *F-measure* after 140 epochs. The SOM clustering result was improved by considering class labels like the semi-SOM method, and the accuracy was 92.67% for the density of the *CCN* and 92.66% for the *F-measure*. The accuracy of the proposed RUFFNN clustering method after one epoch was 94.67% for the density of the *CCN* and 94.59% for the *F-measure.* The density of the *CCN* and the accuracy of the *F-measure* using the proposed RSFFNN method after just one epoch of training was 100%. While for the BPN, the accuracy by *F-measure* was 94.00% after 140 epochs of training. All clustering methods show three distinct clusters for this dataset.

6.2.3 The RUFFNN and RSFFNN clustering on Spambase

The Spambase dataset from the UCI Repository has two classes: Spam and Non-Spam. For computing clustering accuracy of the RUFFNN on the Spambase dataset, the *F-measure* with 10 folds of the test set was considered. The Spambase dataset has 4601 data records. Therefore, in each performance, we considered 4141 data as the training set and 460 data as the test set. Table 6-7 shows the details of the computation of the *F-measure* in different performances on the test set for computing clustering accuracy.

Table 6-7: The clustering results of the RUFFNN method on the Spambase dataset

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	360	100	234	51	126	49	61.96	0.82685512	0.65	0.72783826
2	239	221	143	72	96	149	46.80	0.48972603	0.59832636	0.5386064
3	435	25	311	25	124	0	73.04	1	0.71494253	0.83378016
4	164	296	107	149	57	147	55.65	0.42125984	0.65243902	0.51196172
5	189	271	120	125	69	146	53.26	0.45112782	0.63492063	0.52747253
6	80	380	61	187	19	193	53.91	0.24015748	0.7625	0.36526946
7	412	48	280	14	132	34	63.91	0.89171975	0.67961165	0.77134986
8	159	301	138	139	21	162	60.22	0.46	0.86792453	0.60130719
9	382	78	274	78	108	0	76.52	1	0.71727749	0.83536585
10	367	93	184	38	183	55	48.26	0.76987448	0.5013624	0.60726073

As shown in Table 6-7, the average of the evaluated *F-measure* with 10 folds of the test set for the RUFFNN clustering was 63.20% after just one epoch of training in

333.1057 milliseconds. Table 6-8 and Figure 6-5 show the computed *SW* vector of the Spambase dataset by the RUFFNN clustering method.

Table 6-8: The *SW* vector of the Spambase dataset by the RUFFNN method

Vector of the *SW* for the Spambase dataset									
SW_1	SW_2	SW_3	SW_4	SW_5	SW_6	SW_7	SW_8	SW_9	SW_{10}
0.019631	0.015705	0.018765	0.012721	0.019144	0.020836	0.020431	0.019307	0.020269	0.017956
SW_{11}	SW_{12}	SW_{13}	SW_{14}	SW_{15}	SW_{16}	SW_{17}	SW_{18}	SW_{19}	SW_{20}
0.019922	0.017798	0.021841	0.018471	0.018038	0.019766	0.019989	0.020218	0.017627	0.017276
SW_{21}	SW_{22}	SW_{23}	SW_{24}	SW_{25}	SW_{26}	SW_{27}	SW_{28}	SW_{29}	SW_{30}
0.017263	0.015108	0.019839	0.018922	0.017958	0.017586	0.019759	0.017094	0.015709	0.016812
SW_{31}	SW_{32}	SW_{33}	SW_{34}	SW_{35}	SW_{36}	SW_{37}	SW_{38}	SW_{39}	SW_{40}
0.015201	0.014767	0.019572	0.014796	0.015992	0.017664	0.019267	0.014371	0.017750	0.015158
SW_{41}	SW_{42}	SW_{43}	SW_{44}	SW_{45}	SW_{46}	SW_{47}	SW_{48}	SW_{49}	SW_{50}
0.016484	0.017875	0.017785	0.018333	0.020550	0.020827	0.013631	0.017322	0.017239	0.015578
SW_{51}	SW_{52}	SW_{53}	SW_{54}	SW_{55}	SW_{56}	SW_{57}			
0.018241	0.017722	0.017128	0.014225	0.014375	0.012340	0.014047			

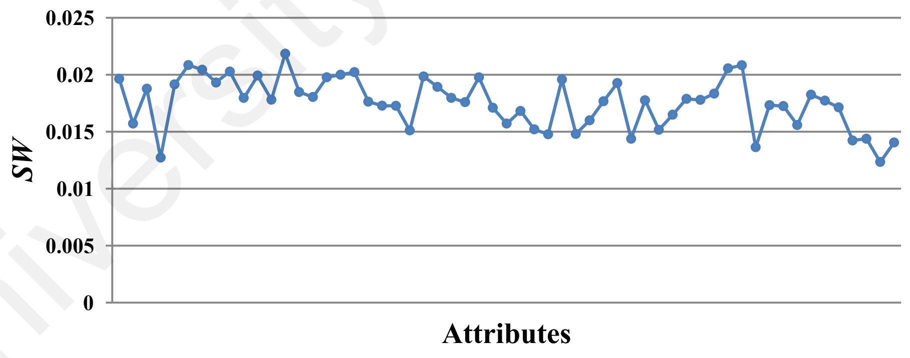


Figure 6-5: The *SW* vector of the Spambase dataset using the RUFFNN method

The total thresholds of input data were computed based on the *SW* vector, and the input data were subsequently clustered. As shown in Figure 6-6, two distinct clusters can be recognized.

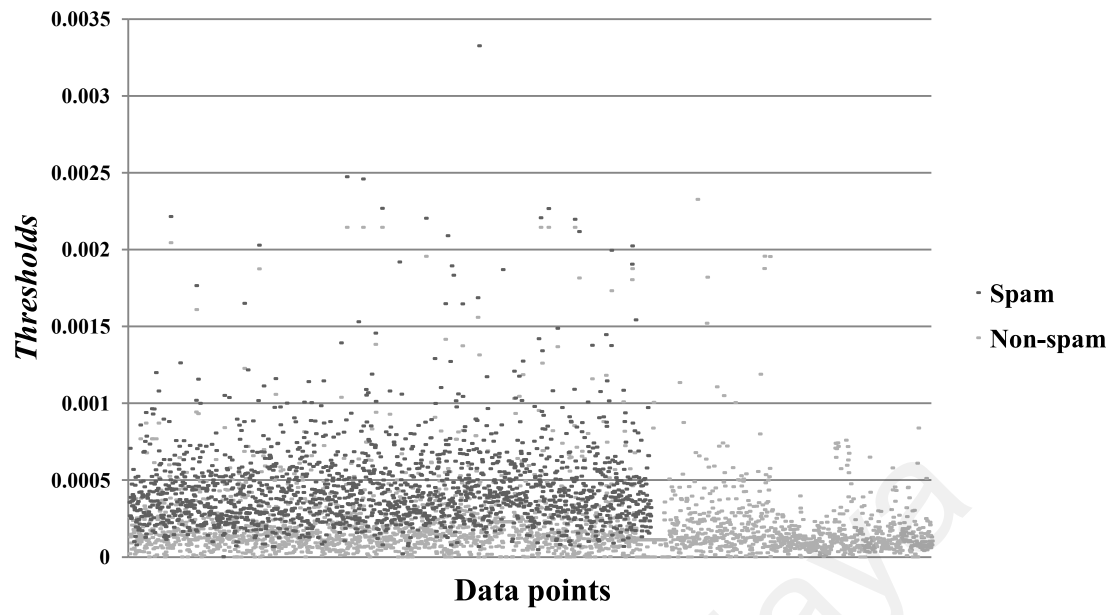


Figure 6-6: The clusters of the RUFFNN method on the Spambase dataset

We compared the results of the proposed RUFFNN method with the results of some related methods. Table 6-9 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* in the Spambase dataset.

Table 6-9 : Comparison of the clustering results on the Spambase dataset by the RUFFNN, RSFFNN and some related methods

The clustering method	Density of *CCN* %	*F-measure%*	Epoch
K-means	23.54	-	20
Neural Gas	22.82	-	20
GNG	21.02	-	5
SOM	57.84	65.21	140
Semi-SOM	76.67	80.74	140
RUFFNN	59.35	63.20	1
RSFFNN	99.90	99.89	1

In Table 6-9 based on the outcomes of the experiment, the density of *CCN* of the *K-means* and the NG methods are 23.54% and 22.82% respectively, after 20 epochs of training (Camastra & Verri, 2005). The density of *CCN* of the GNG method is 21.02% after 5 epochs (Costa & Oliveira, 2007). The SOM produced 57.84% for the density of the *CCN* and 65.21% of the *F-measure* after 140 epochs. The SOM clustering result was improved by considering class labels like the semi-SOM method, and the accuracy was improved to 76.67% for the density of the *CCN* and 80.74% of the *F-measure*. The accuracy of the proposed RUFFNN clustering method after one epoch was 59.35% for the density of the *CCN* and 63.20% of the *F-measure.* The density of the *CCN* and the accuracy by of the *F-measure* of the semi-clustering by the proposed RSFFNN method after just one epoch of training was 99.90% and 99.89%, respectively. While for the BPN, the accuracy of *F-measure* was 79.50% after 2000 epochs of training. All clustering methods show two distinct clusters for this dataset.

6.2.4 The RUFFNN and RSFFNN clustering on Arcene

The Arcene dataset from the UCI Repository has two classes, namely cancer patients and healthy patients. We considered the training dataset and validation dataset with 200 total instances together as one set. Therefore, in each performance, we considered 180 data as the training set and 20 data as the test set. Table 6-10 shows the details of the computation for the *F-measure* in different performances on the test set for computing clustering accuracy.

Table 6-10: The clustering results of the RUFFNN method on the Arcene dataset

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	13	7	7	5	6	2	60.00	0.7777778	0.5384615	0.6363636
2	12	8	9	2	3	6	55.00	0.6	0.75	0.6666667
3	11	9	7	4	4	5	55.00	0.5833333	0.6363636	0.6086957
4	11	9	6	6	5	3	60.00	0.6666667	0.5454545	0.6
5	9	11	7	6	2	5	65.00	0.5833333	0.7777778	0.6666667
6	10	10	6	6	4	4	60.00	0.6	0.6	0.6
7	13	7	8	5	5	2	65.00	0.8	0.6153846	0.6956522
8	11	9	8	5	3	4	65.00	0.6666667	0.7272727	0.6956522
9	12	8	8	5	4	3	65.00	0.7272727	0.6666667	0.6956522
10	10	10	8	6	2	4	70.00	0.6666667	0.8	0.7272727

As shown in Table 6-10, the average of the evaluated *F-measure* for 10 folds of the test set for the RUFFNN clustering was 65.93% after just one epoch of training in 5 seconds and 386.139 milliseconds. Table 6-11 shows the computed *SW* vector of the Arcene dataset by the RUFFNN clustering method.

Table 6-11: The *SW* vector of the Arcene dataset by the RUFFNN method

Vector of the *SW* for the Arcene dataset									
SW_1	SW_2	SW_3	SW_4	SW_5	SW_6	SW_7	SW_8	SW_9	SW_{10}
0.000113	7.41E-05	0.000135	0.000148	9.08E-05	0.000167	0.000195	9.07E-05	0.000113	9.05E-05
SW_{11}	SW_{12}	SW_{13}	SW_{14}	SW_{15}	SW_{16}	SW_{17}	SW_{18}	SW_{19}	SW_{20}
7.90E-05	8.66E-05	8.94E-05	8.09E-05	0.000173	8.65E-05	9.20E-05	8.21E-05	0.000113	7.70E-05
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
SW_{99991}	SW_{99992}	SW_{99993}	SW_{99994}	SW_{99995}	SW_{99996}	SW_{99997}	SW_{99998}	SW_{9999}	SW_{10000}
0.000155	0.000211	0.000107	0.000101	0.000102	0.000136	0.000176	8.69E-05	9.70E-05	0.000231

The total thresholds of input data were computed based on the *SW* vector, and the input data were subsequently clustered. As shown in Figure 6-7, two distinct clusters can be recognized.

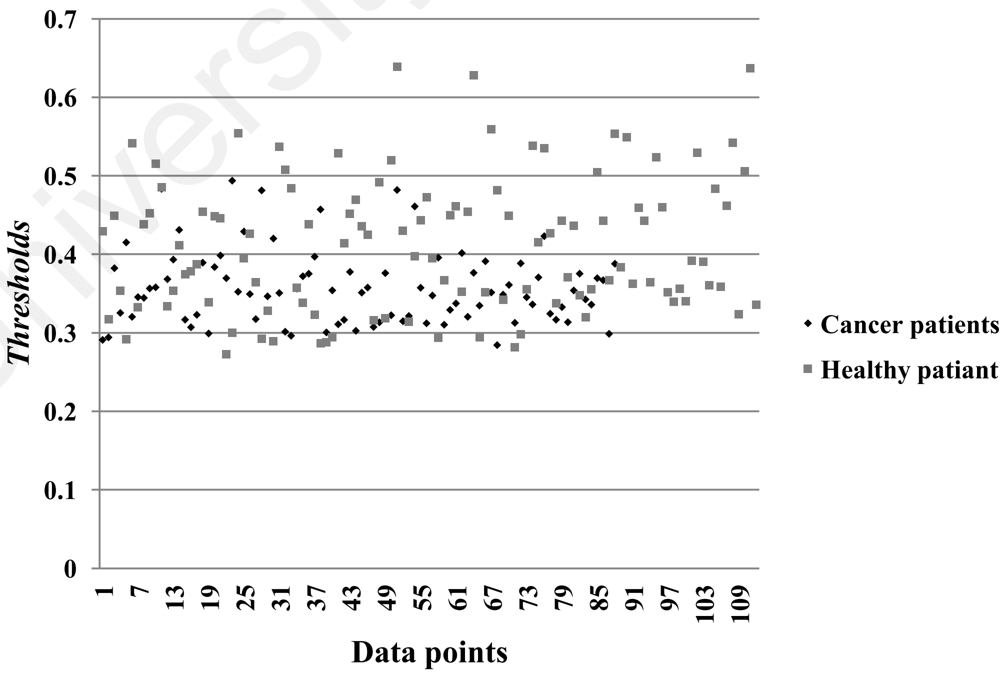


Figure 6-7: The clusters of the RUFFNN method on the Arcene dataset

We compared the results of the proposed RUFFNN method with the results of some related methods. Table 6-12 shows the speed of processing on the basis of the number of epochs and the accuracy on the basis of the density of the *CCN* in the Arcene dataset.

Table 6-12: Comparison of the clustering results on the Arcene dataset by the RUFFNN, RSFFNN and some related methods

The clustering method	Density of *CCN* %	*F-measure%*	Epoch
K-means	59.00	-	10
SOM	53.00	58.04	140
Semi-SOM	64.00	67.86	140
RUFFNN	62.00	65.93	1
RSFFNN	100	100	1

In Table 6-12, the *K-means* produced 59.00% for the density of the *CCN* after 10 epochs (Sárközy, Song, Szemerédi, & Trivedi, 2012). The SOM produced 53.00% for the density of the *CCN* and 58.04% by using the *F-measure* after 140 epochs. The SOM clustering result was improved by considering class labels regarding the semi-SOM method, and the accuracy was changed to 64.00% for the density of the *CCN* and 67.86% based on the *F-measure*. The accuracy of the proposed RUFFNN clustering method after one epoch was 62.00% for the density of the *CCN* and 65.93% by using the *F-measure.* The density of the *CCN* and the accuracy by using the *F-measure* of the semi-clustering by the proposed RSFFNN method after just one epoch of training was 100%. Recently, Veenu Mangat and Renu Vig (Mangat & Vig, 2014) reported classification of the Arcene dataset by several classification methods such as *K*-NN. *K*-nearest neighbour (*K*-NN) is a supervised classifier that is able to learn by analogy and performs on *n*-dimensional numeric attributes (Dasarathy, 1990). Given an unknown instance, *K*-NN finds *K* instances in the training set that are closest to the given instance pattern and predicts one or average of class labels or credit-rates. Unlike BPN, *K*-NN assigns equal weights to the attributes. The *K*-NN (*K*=10) was able to classify the

Arcene dataset with 77.00% accuracy by the *F-measure* after several epochs and 10 times re-executing the method. All clustering methods show two distinct clusters for this dataset.

6.2.5 The RUFFNN and RSFFNN clustering on Yeast

The Yeast dataset from the UCI Repository contains 1484 samples with 8 attributes and 10 classes. Table 6-13 and Figure 6-8 shows the computed *SW* vector of the Yeast dataset by the RUFFNN clustering method.

Table 6-13: The *SW* vector of the Yeast dataset by the RUFFNN method

Vector of the *SW* for the Yeast dataset							
SW_1	SW_2	SW_3	SW_4	SW_5	SW_6	SW_7	SW_8
0.067878	0.070346	0.087059	0.128631	0.226319	0.226192	0.073931	0.119643

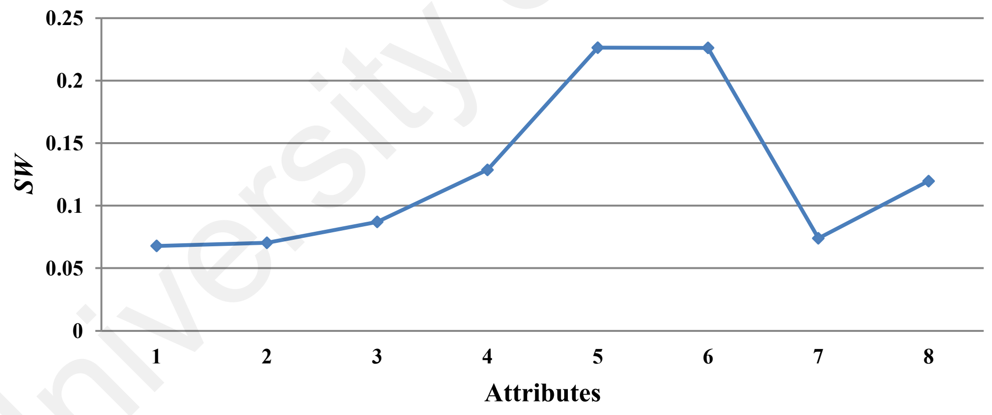


Figure 6-8: The final *SW* vector of the Yeast dataset using the RUFFNN method

The total thresholds of input data were computed based on the *SW* vector, and the input data were subsequently clustered. Finally, the clusters that the RUFFNN can recognized are shown in Table 6-14.

Table 6-14: The clusters of the RUFFNN method on the Yeast dataset

	Class CYT	Class ERL	Class EXC	Class ME1	Class ME2	Class ME3	Class MIT	Class NUC	Class POX	Class VAX
Cluster 1	201	0	2	2	7	38	39	164	4	0
Cluster 2	0	5	0	0	0	0	1	1	0	0
Cluster 3	5	0	3	5	3	1	13	7	0	0
Cluster 4	9	0	5	6	1	0	7	14	1	0
Cluster 5	30	0	7	12	6	5	43	45	0	0
Cluster 6	59	0	1	0	4	40	8	46	0	0
Cluster 7	52	0	7	15	13	7	80	64	1	0
Cluster 8	183	0	4	3	9	24	46	149	3	0
Cluster 9	8	0	0	0	1	1	1	7	1	0
Cluster 10	13	0	0	0	2	2	2	8	1	3

As shown in Table 6-14, clustering of the Yeast dataset was so difficult and the clusters often overlapped other near neighbour clusters. Therefore, each cluster shared members of other near neighbour clusters. We compared the results of the proposed RUFFNN method with the results of some related methods. Table 6-15 shows the speed of processing on the basis of the number of epochs and the accuracy based on the density of the *CCN* in the Yeast dataset.

Table 6-15 : Comparison of the clustering results on the Yeast dataset by the RUFFNN, RSFFNN and some related methods

The clustering method	Density of *CCN* %	*F-measure%*	Epoch
SOM	32.55	24.02	140
Semi-SOM	40.23	34.05	140
RUFFNN	33.29	27.43	1
RSFFNN	100	100	1

In Table 6-15, the SOM produced 32.55% for the density of the *CCN* and 24.02% by using the *F-measure* after 140 epochs. The SOM clustering result was improved by considering class labels regarding the semi-SOM method, and the accuracy was improved to 40.23% for the density of the *CCN* and 34.05% using the *F-measure*. The accuracy of the proposed RUFFNN clustering method after one epoch was 33.29% for the density of the *CCN* and 27.43% by using the *F-measure,* after 684.833 milliseconds in one epoch of training. The density of the *CCN* and the accuracy by using the *F-measure* of the proposed RSFFNN method after one epoch was 100%, after just one epoch of training. Several literature reported the difficulty of clustering or classification of the Yeast dataset. Longadge et al. (Longadge & Dongre, 2013) reported the classification of the Yeast dataset by several classification methods such as *K*-NN. The *K*-NN (*K*=3) was able to classify the Yeast dataset with 0.11% accuracy by the *F-measure* after several epochs and times re-executing the method. Also, Ahirwar (Ahirwar, 2014) reported that the *K-means* was able to classify the Yeast dataset with 65.00% accuracy by the *F-measure* after several epochs.

6.2.6 The RSFFNN Clustering on the Breast Cancer (UMMC)

The dataset was collected by the University of Malaya Medical centre (UMMC), Kuala Lumpur from 1992 until 2002 (Hazlina, et al., 2004). Table 6-16 shows the results of the implementation of the proposed RSFFNN method. The number of data of each subset; CPU time usage per second for training each subset during one epoch; and the accuracy of the semi-clustering of each subset of the breast cancer dataset based on the *F-measure* with 10 folds of the test dataset by using the RSFFNN clustering model are shown.

Table 6-16: The RSFFNN results for each subset of the UMMC breast cancer data

Year	Density of *CCN* (%)	The number of data in each subset	Epoch	CPU Time usage (milliseconds)	Accuracy of the RSFFNN (*F-measure%*)
1_{st} year	99.03	827	1	43	99.55
2_{nd} year	98.96	673	1	34.5	98.85
3_{rd} year	98.44	561	1	32.5	99.04
4_{th} year	97.5	440	1	32	98.29
5_{th} year	100	355	1	29.4	100
6_{th} year	100	270	1	15.8	100
7_{th} year	100	200	1	15	100
8_{th} year	100	124	1	14.5	100
9_{th} year	100	56	1	13.7	100

Table 6-16 shows that the training process for each sub-set of the breast cancer dataset took one epoch between [13.7, 43] milliseconds of CPU time; and the accuracies of the RSFFNN using the *F-measure* for the breast cancer sub-data sets were between [98.29% - 100%]. For comparison with other similar methods in the scope of this research, we implemented the SOM using BPN as a hybrid method. The SOM clustered each subset of breast cancer dataset and found the *SW* vector of each instance after 20 epochs. The BPN model fine-tuned the weights code book of unfolding SOM model instead of random weights. The training process in the BPN was 25 epochs. The results of the hybrid method of the SOM-BPN are shown in Table 6-17 for every subset of data.

Table 6-17: Comparison of the results of the PCA-BPN, the SOM-BPN and the RSFFNN for each subset of the UMMC breast cancer dataset

Year	PCA-BPN (*F-measure%*)	SOM-BPN (*F-measure%*)	RSFFNN (*F-measure%*)
1st year	76	82	99.55
2nd year	63	72	98.85
3rd year	62	71	99.04
4th year	77	78	98.29
5th year	83	86	100
6th year	93	93	100
7th year	98	98	100
8th year	99	99	100
9th year	99	99	100

The PCA was considered as a preprocessing technique for dimension reduction and used by the BPN model. Table 6-17 shows the result of the PCA-BPN hybrid model for every subset of the breast cancer dataset of the UMMC. The PCA took the time of the CPU for dimension reduction and the BPN used the output of the PCA for classification after several epochs. The results of Table 6-17 shows the accuracies of implementation of the PCA-BPN model using the *F-measure* for the breast cancer dataset which were between [62%- 99%], and the accuracies of implementation of the SOM-BPN model using the *F-measure* for each subset of the breast cancer dataset which were between [71%- 99%].

6.3 Summary

This chapter illustrated the experimental results and evaluation of the RUFFNN and RSFFNN clustering performances, and compared the proposed methods to several related clustering methods by using the various datasets from UCI Repository and the UMMC. For experimentation of the purposes, the training speed was measured by the number of epochs and the CPU time usage. The accuracy of the clustering methods was measured by employing the *F-measure* with 10 folds of the test set, and also through the number of clusters and the density of the *correctly classified nodes (CCN)*. The time complexity and the memory complexity were measured through the number of input data, training iterations and clusters; and the densities of the clusters. The outcomes of the experiments proved that the RUFFNN clustering method, which is the fundamental pattern to propose the DUFFNN clustering method, had the superior results. Also, this chapter showed the results of the RSFFNN on the five datasets from UCI Repository and the UMMC breast cancer dataset were more accurate than the results of the RUFFNN clustering method.

CHAPTER 7: EXPERIMENTAL RESULTS AND EVALUATION ON THE DUFFNN AND DSFFNN CLUSTERING METHODS

7.1 Introduction

In this chapter, we implement the proposed dynamic unsupervised feedforward neural network (DUFFNN) clustering method, and evaluate their performances based on measuring the CPU time usage, clustering accuracy, time complexity and memory complexity of each method in order to achieve the third objective of the research, as mentioned in Section 1.3. Furthermore, the performance of the DUFFNN is compared against the static unsupervised feedforward neural network (UFFNN) clustering and some effective related ODUFFNN clustering methods, such as the DSOM and ESOM. There is a trade-off between training time, clustering accuracy, time complexity and memory complexity of the ODUFFNN clustering methods, and there is surprisingly and comparatively very little works dealing with them together in one ODUFFNN clustering model (Furao & Hasegawa, 2006; Hebooul, et al., 2015; Kulkarni & Mulay, 2013; Liu, et al., 2013; Prudent & Ennaji, 2005; Rougier & Boniface, 2011). In this chapter, we evaluate the exprimental results of the DSOM, ESOM and semi-ESOM on the nine datasets obtained from the University of California at Irvine (UCI) Machine Learning Repository (Asuncion & Newman, 2007) and compare them with the DUFFNN clustering results. Moreover, an improved version of the DUFFNN, called the dynamic semi-supervised feedforward neural network (DSFFNN) method was also tested on the nine datasets from UCI Repository and the breast cancer dataset from the UMMC.

7.2 Experimental Evaluation on the DUFFNN and DSFFNN Clustering

The proposed DUFFNN and DSFFNN clustering methods were tested on the breast cancer-Wisconsin, Iris, Spambase, Spect Heart, Spectf Heart, Musk1, Musk2, Arcene and Yeast datasets from the UCI Repository (Asuncion & Newman, 2007), as mentioned in Section 4.2.6. The methods were also compared with several current related methods such as the ESOM and DSOM, and some popular supervised feedforward neural network methods such as the BPN classification. There are a few published papers regarding implementation of the ODUFFNN clustering on common datasets. We implemented the ESOM and DSOM as effective methods in the ODUFFNN clustering, and compared their results with the results of our proposed methods. In order to initialize the parameters with the best values in the ESOM and DSOM, we executed three times these clustering models on each selected dataset. We implemented the ESOM, by considering the parameter set $P=\{\varepsilon, \delta, \gamma, T_p\}$, ε as distance threshold, δ for controlling the spread of the neighbourhood, γ as a small constant learning rate, T_p as the steps of learning time, and β as forgetting constant. Also, we implemented the DSOM, by considering the parameters ε the rate of learning and δ control distance from the *BMU* as the elasticity.

Furthermore, the DSFFNN method was performed on the breast cancer dataset of the UMMC, in order to predict the survival time of the patients.

7.2.1 The DUFFNN and DSFFNN Clustering on the Breast Cancer Wisconsin

In order to compute the clustering accuracy of the DUFFNN on the breast cancer Wisconsin dataset, the *F-measure* for 10 folds of the test set was considered. Table 7-1 illustrates the details of the clustering results.

Table 7-1: The clustering result of the DUFFNN method on the breast cancer Wisconsin

Test subset	*p*	*n*	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	51	18	48	16	3	2	92.75	0.96	0.941176	0.950495
2	44	25	42	25	2	0	97.10	1	0.954545	0.976744
3	42	27	42	26	0	1	98.55	0.976744	1	0.988235
4	46	23	44	20	2	3	92.75	0.93617	0.956522	0.946237
5	48	21	46	17	2	4	91.30	0.92	0.958333	0.938776
6	43	26	43	26	0	0	100	1	1	1
7	51	18	49	17	2	1	95.65	0.98	0.960784	0.970297
8	42	27	42	27	0	0	100	1	1	1
9	41	28	41	28	0	0	100	1	1	1
10	43	26	43	26	0	0	100	1	1	1

As shown in Table 7-1, the average of the evaluated *F-measure* for 10 folds of the test set for the DUFFNN clustering was 97.71% after just one epoch of training during 15.66 milliseconds.

Table 7-2 and Figure 7-1 show the final computed *BMW* vector of the breast cancer Wisconsin dataset by the DUFFNN clustering method.

Table 7-2: The final *BMW* vector of the breast cancer Wisconsin (original) using the DUFFNN method

Vector of the final *BMW* for the received Breast cancer Wisconsin (original) data								
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8	BMW_9
0.180225	0.076316	0.081864	0.093955	0.11711	0.112795	0.123886	0.093721	0.120128

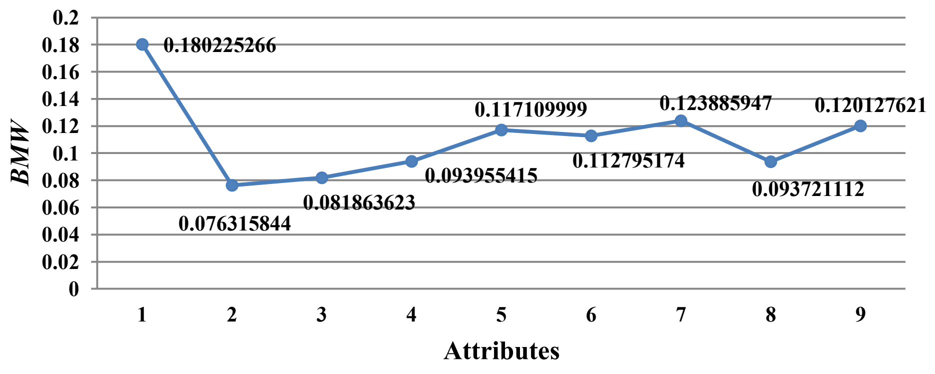


Figure 7-1: The final *BMW* vector of the breast cancer Wisconsin (original) using the DUFFNN method

The total thresholds of the received data points were computed on the basis of the final *BMW* vector, and the data points were sequentially clustered. As shown in Figure 7-2, two distinct clusters can be recognized.

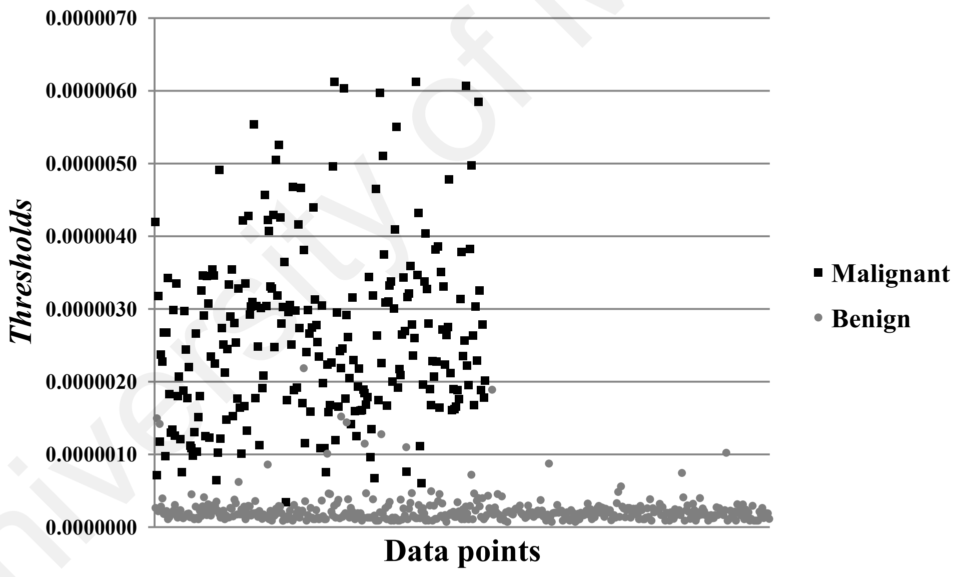


Figure 7-2: The clusters of the DUFFNN method on the breast cancer Wisconsin (original)

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In order to initialize the parameters of the ESOM, we considered: β = 0.8, ε = 0.005, δ = ε and γ = 0.005. Also, in order to initialize the parameters of the DSOM, we considered ε =0.100 and δ =1.25. Table 7-3 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and the *F-measure* for the breast cancer Wisconsin data points.

Table 7-3: Comparison of the clustering results on the breast cancer Wisconsin data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch	CPU TIME (milliseconds)
SOM	96.63	-	20	-
K-means	96.19	-	20	-
Neural Gas	96.19	-	20	-
GNG	69.84	-	5	-
DSOM	67.20	74.77	700	4' , 22" and 165
ESOM	93.41	95.08	1	2" and 912
Semi-ESOM	94.00	95.52	1	2" and 912
DUFFNN	96.81	97.71	1	15.66
DSFFNN	100	100	1	15.66

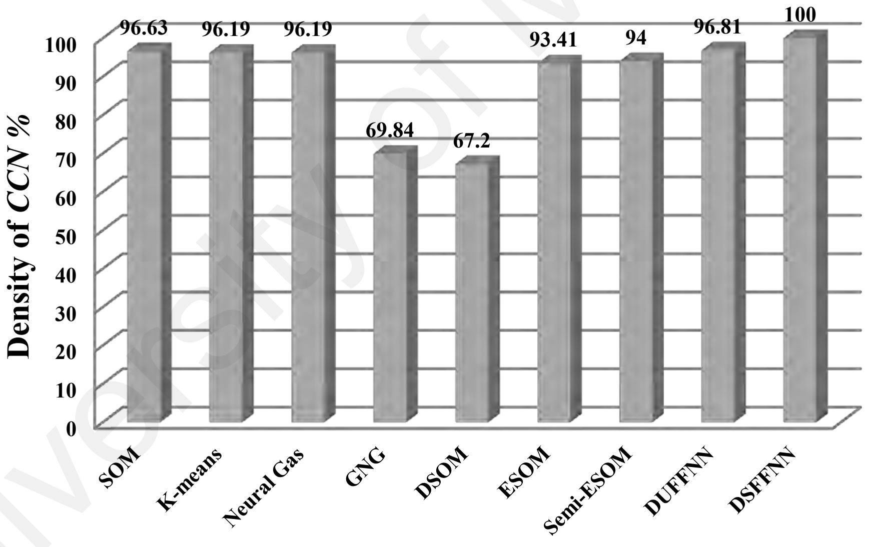


Figure 7-3: Comparison of the clustering density of *CCN %* on the breast cancer Wisconsin data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-3 and Figure 7-3 based on the experiments, the density of *CCN* of the SOM was 96.63% after 20 epochs, which was the best result of the mentioned UFFNN clustering methods. The density of *CCN* and the *F-measure* of the DSOM method were 67.20% and 74.77%, respectively, after 700 epochs in 4' , 22" and 165 milliseconds. The density of the *CCN* and the *F-measure* of the ESOM method were 93.41% and 95.08% , respectively, after 1 epoch in 2" and 912 milliseconds. The ESOM clustering result was improved by considering class labels like the semi-ESOM method, and the accuracy was improved to 94.00% for the density of the *CCN* and 95.52% by the *F-measure*. The accuracy of the proposed DUFFNN clustering method was 96.81% for the density of the *CCN* and 97.71% by the *F-measure*, respectively after 1 epoch in 15.66 milliseconds. The density of the *CCN* and the *F-measure* accuracy of the proposed DSFFNN method after one epoch was 100%, after just one epoch of training. While for the BPN, the *F-measure* was 99.28% after 1000 epochs of training. All clustering methods show two distinct clusters for this dataset.

7.2.2 The DUFFNN and DSFFNN Clustering on the Iris

In order to compute the clustering accuracy of the DUFFNN on on the Iris data points, the *F-measure* with 10 folds of the test set was considered. Table 7-4 shows the details.

Table 7-4: The clustering result of the DUFFNN method on the Iris

Test subset	Cluster1			Cluster2			Cluster3			*CCN%*	*F-measure*
	t_{c1}	f_{c1}	$f^{'}_{c1}$	f_{c2}	t_{c2}	$f^{'}_{c2}$	f_{c3}	$f^{'}_{c3}$	t_{c3}		
1	9	1	0	0	5	0	0	0	0	93.33	0.9545
2	9	1	0	2	3	0	0	0	0	80.00	0.76905
3	8	0	0	0	6	0	0	0	1	100	1
4	13	0	0	0	2	0	0	0	0	100	1
5	7	0	0	0	5	0	0	0	3	100	1
6	2	0	0	0	6	0	0	0	7	100	1
7	0	0	0	0	14	0	0	0	1	100	1
8	0	0	0	0	7	0	0	0	8	100	1
9	0	0	0	0	0	0	0	0	15	100	1
10	0	0	0	0	0	0	0	0	15	100	1

As shown in Table 7-4, the *F-measure* with 10 folds of the dataset was 97.24% after just one epoch of training in 26.68 milliseconds, by the DUFFNN clustering.

Table 7-5 and Figure 7-4 show the final computed *BMW* vector components of the received Iris data using the DUFFNN clustering method.

Table 7-5: The final *BMW* vector of the Iris using the DUFFNN method

Vector of the final *BMW* for the received Iris data			
BMW_1	BMW_2	BMW_3	BMW_4
0.163002	0.328439	0.226318	0.282241

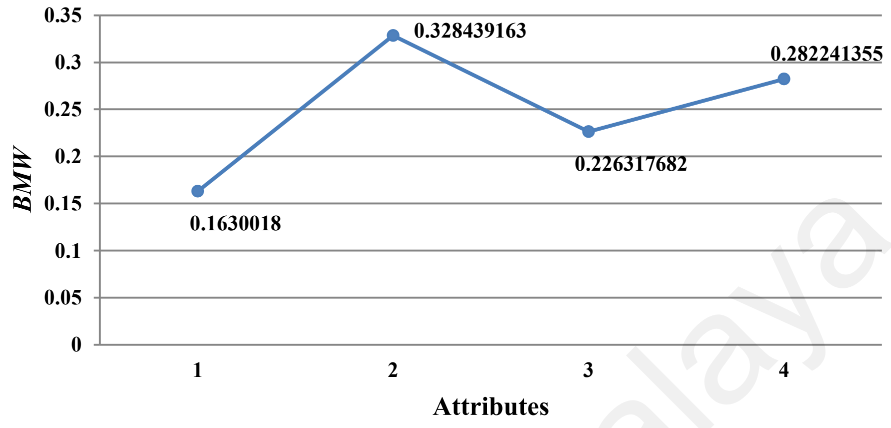


Figure 7-4: The final *BMW* vector of the Iris by the DUFFNN method

The total thresholds of the received data points were computed based on the final *BMW* vector, and the data points were sequentially clustered. As shown in Figure 7-5, three distinct clusters can be recognized.

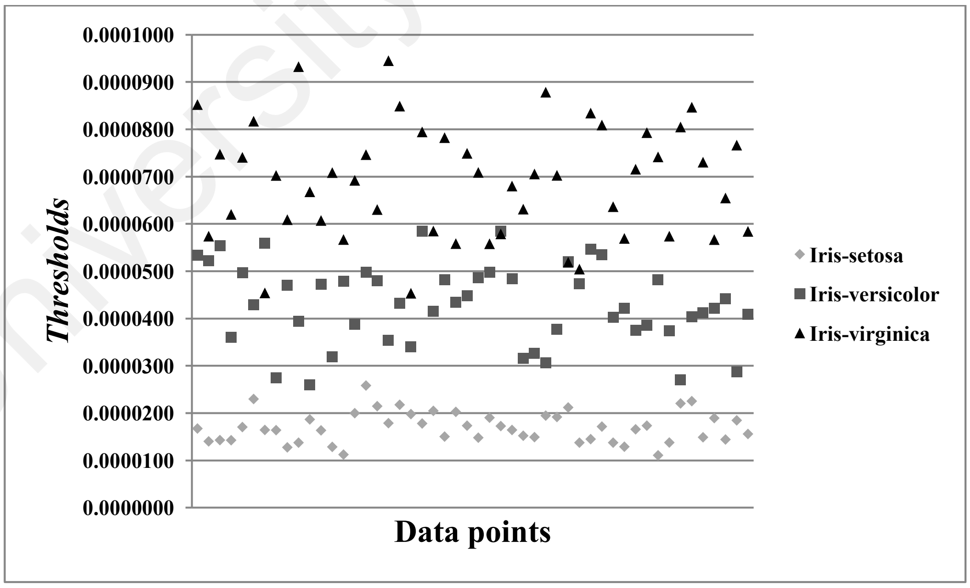


Figure 7-5: The clusters of the DUFFNN method on the Iris

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In the ESOM, we considered the parameter: β = 0.8, ε = 0.004, δ = ε and γ = 0.005. In the DSOM, we considered the parameters ε

=0.100 and δ =1.24. Table 7-6 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and the *F-measure* for the Iris data points.

Table 7-6: Comparison of the clustering results on the Iris data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch	CPU TIME (milliseconds)
SOM	85.22	-	140	-
K-means	89.33	-	20	-
Neural Gas	92.67	-	20	-
GNG	90.00	-	10	-
ESOM	96.00	96.00	1	36
DSOM	90.00	90.00	700	39" and 576
Semi-ESOM	100	100	1	36
DUFFNN	97.33	97.24	1	26.68
DSFFNN	100	100	1	26.68

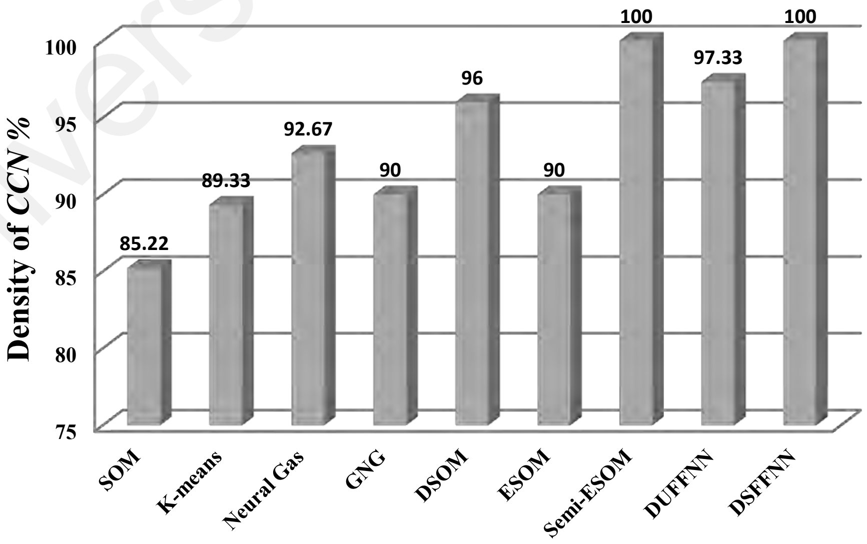


Figure 7-6: Comparison of the clustering density of *CCN %* on the Iris data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-6 and Figure 7-6, the density of the *CCN* for the NG was 92.67% after 20 epochs, which was the best result of the mentioned UFFNN clustering methods. The DSOM clustered the data points with 90.00% for the density of the *CCN* and 90.00% for the *F-measure* after 700 epochs in 39" and 576 milliseconds. The ESOM clustered the Iris data points with 96.00% for the density of the *CCN* and 96.00% accuracy for the *F-measure* after 1 epoch in 36 milliseconds. The semi-ESOM clustered the Iris data points with 100% for the density of the *CCN* and 100% accuracy for the *F-measure*. The DUFFNN method clustered this dataset with 97.33% for the density of the *CCN* and 97.24% accuracy by the *F-measure*. The BPN classification model, learnt this dataset after 140 epochs with the accuracy of 94.00% by the *F-measure*. The results of the DSFFNN show 100% for the density of the *CCN* and 100% accuracy for the *F-measure* after 1 epoch for training just in 26.68 milliseconds. All clustering methods show three distinct clusters for this dataset.

7.2.3 The DUFFNN and DSFFNN Clustering on Spambase

In order to compute the clustering accuracy of the DUFFNN on the Spambase dataset, the *F-measure* with 10 folds of the test set was considered. Table 7-7 shows the details.

Table 7-7: The clustering result of the DUFFNN method on the Spambase

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	304	157	158	134	146	23	63.34	0.872928	0.519737	0.651546
2	290	171	166	136	124	35	65.51	0.825871	0.572414	0.676171
3	361	100	274	79	87	21	76.57	0.928814	0.759003	0.835366
4	320	141	204	127	116	14	71.80	0.93578	0.6375	0.758364
5	362	99	267	86	95	13	76.57	0.953571	0.737569	0.831776
6	242	219	217	124	25	95	73.97	0.695513	0.896694	0.783394
7	217	244	189	123	28	121	67.68	0.609677	0.870968	0.717268
8	174	287	141	132	33	155	59.22	0.476351	0.810345	0.6
9	263	198	216	127	47	71	74.40	0.752613	0.821293	0.785455
10	264	197	239	19	25	178	55.97	0.573141	0.905303	0.701909

As shown in Table 7-7, the average of the evaluated *F-measure* for 10 folds of the test set for the DUFFNN clustering was 73.41% after just one epoch of training in 35" and 339 milliseconds. Table 7-8 and Figure 7-7 show the final computed *BMW* vector of the Spambase data by the DUFFNN clustering method.

Table 7-8: The final computed *BMW* vector from the received Spambase data using the DUFFNN method

Vector of the final *BMW* for the received Spambase data									
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8	BMW_9	BMW_{10}
0.02041	0.013759	0.039625	0.011594	0.024689	0.017592	0.017119	0.013722	0.019188	0.014086
BMW_{11}	BMW_{12}	BMW_{13}	BMW_{14}	BMW_{15}	BMW_{16}	BMW_{17}	BMW_{18}	BMW_{19}	BMW_{20}
0.020673	0.040274	0.018911	0.013514	0.015144	0.014275	0.019062	0.019466	0.062005	0.012114
BMW_{21}	BMW_{22}	BMW_{23}	BMW_{24}	BMW_{25}	BMW_{26}	BMW_{27}	BMW_{28}	BMW_{29}	BMW_{30}
0.04812	0.013873	0.018592	0.011919	0.023215	0.017149	0.019887	0.016455	0.012683	0.017408
BMW_{31}	BMW_{32}	BMW_{33}	BMW_{34}	BMW_{35}	BMW_{36}	BMW_{37}	BMW_{38}	BMW_{39}	BMW_{40}
0.011425	0.013366	0.014677	0.013428	0.011509	0.016	0.021896	0.011719	0.013494	0.014402
BMW_{41}	BMW_{42}	BMW_{43}	BMW_{44}	BMW_{45}	BMW_{46}	BMW_{47}	BMW_{48}	BMW_{49}	BMW_{50}
0.013976	0.015335	0.016717	0.013087	0.016456	0.015715	0.012225	0.012602	0.013692	0.013523
BMW_{51}	BMW_{52}	BMW_{53}	BMW_{54}	BMW_{55}	BMW_{56}	BMW_{57}			
0.012327	0.010982	0.013187	0.01097	0.012174	0.009847	0.018746			

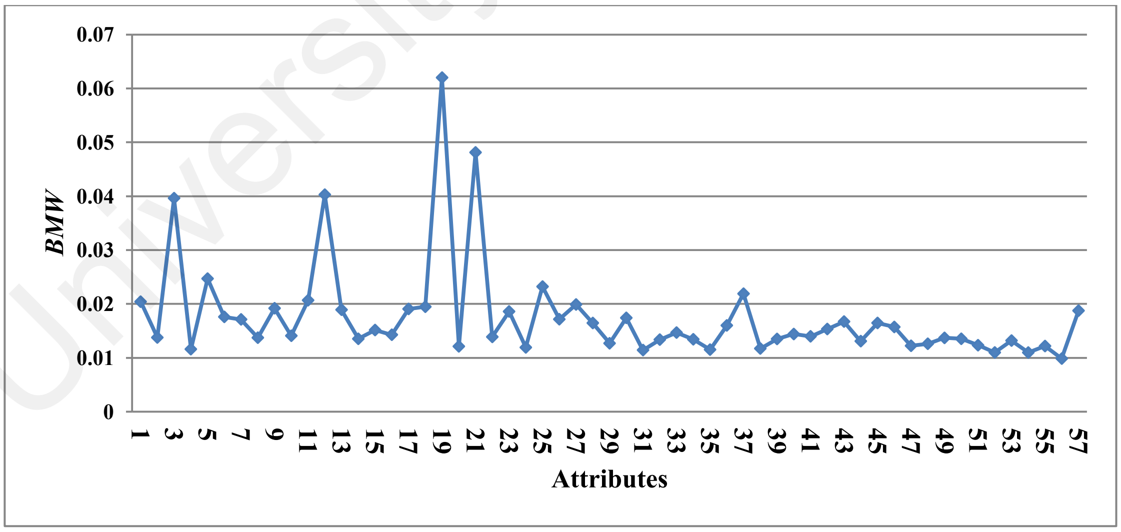


Figure 7-7: The final *BMW* vector of the Spambase dataset by the DUFFNN method

The total thresholds of the received data points were computed based on the final *BMW* vector, and the data points were subsequently clustered. As shown in Figure 7-8, two distinct clusters can be recognized.

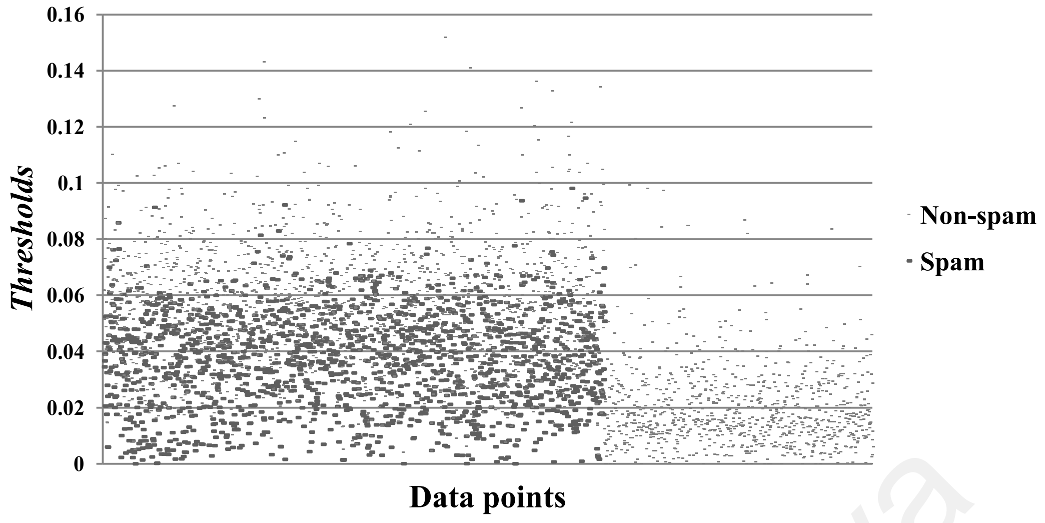


Figure 7-8: The clusters of the DUFFNN method on the Spambase

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.0053$, $\delta = \varepsilon$ and $\gamma = 0.005$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.25$. Table 7-9 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and the *F-measure* for the Spambase data points.

Table 7-9: Comparison the clustering results on the Spambase data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch	CPU TIME (milliseconds)
SOM	26.30	-	20	-
K-means	23.54	-	20	-
Neural Gas	22.82	-	20	-
GNG	21.02	-	5	-
DSOM	55.83	62.78	700	33' , 27" and 90
ESOM	49.21	57.85	1	14' , 39" and 773
Semi-ESOM	58.29	65.03	1	14' , 39" and 773
DUFFNN	68.50	73.41	1	35" and 339
DSFFNN	99.97	99.96	1	35" and 339

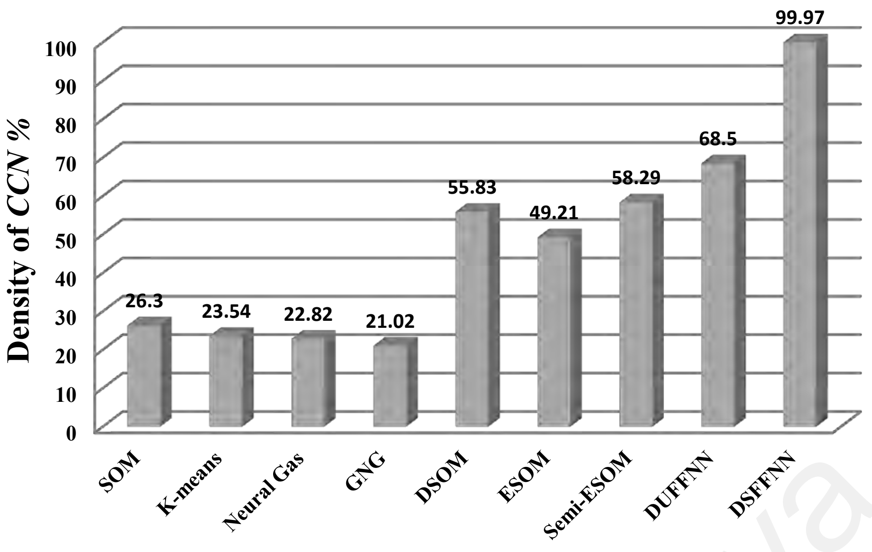


Figure 7-9: Comparison of the clustering density of *CCN %* on the Spambase data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-9 and Figure 7-9, the density of *CCN* of the SOM was 26.30% after 20 epochs (Camastra & Verri, 2005), which was the best result of the mentioned UFFNN clustering methods. The DSOM clustered the data points with 55.83% for the density of the *CCN* and 62.78% accuracy for the *F-measure* after 700 epochs during 33' , 27' and 90 milliseconds. The ESOM clustered the Spambase data points with 49.21% for the density of the *CCN* and 57.85% accuracy for the *F-measure* after 1 epoch during 14' , 39" and 773 milliseconds. The Semi-ESOM clustered the Spambase data points with 58.29% for the density of the *CCN* and 65.03% accuracy for the *F-measure.* The DUFFNN clustering method clustered this dataset with 68.50% for the density of the *CCN* and 73.41% accuracy for the *F-measure* after one epoch in 35" and 339 milliseconds. The BPN learnt this dataset after 2000 epochs with the accuracy of 79.50% using the *F-measure*. The results of the DSFFNN show 99.97% for the density of the *CCN* and 99.96% accuracy for the *F-measure* with 1 epoch of training. All clustering methods show two distinct clusters for this dataset.

7.2.4 The DUFFNN and DSFFNN Clustering on SPECT Heart

The SPECT Heart dataset from the UCI Repository has two classes: Normal and Abnormal. In order to compute the clustering accuracy of the RUFFNN on the SPECT Heart dataset, the *F-measure* with 10 folds of the test set was considered. The SPECT Heart dataset has 267 cases. Therefore, in each performance, we considered 240 data as the training set and 27 data as the test set.

Table 7-10 shows the details of the computation of the *F-measure* in different performances on the test set for computing clustering accuracy.

Table 7-10: The clustering result of the DUFFNN method on the SPECT Heart

Test subset	*p*	*n*	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	18	9	15	7	3	2	81.48	0.882353	0.833333	0.857143
2	23	4	18	0	5	4	66.67	0.818182	0.782609	0.8
3	27	0	21	0	6	0	77.78	1	0.777778	0.875
4	16	11	16	10	0	1	96.30	0.941176	1	0.969697
5	23	4	23	0	0	4	85.19	0.851852	1	0.92
6	25	2	25	0	0	2	92.59	0.925926	1	0.961538
7	26	1	19	0	7	1	70.37	0.95	0.730769	0.826087
8	22	5	22	0	0	5	81.48	0.814815	1	0.897959
9	17	10	17	9	0	1	96.30	0.944444	1	0.971429
10	18	9	18	8	0	1	96.30	0.947368	1	0.972973

As shown in Table 7-10, the average of the evaluated *F-measure* with 10 folds of the test set was 90.52% after just one epoch of training in 12.70 milliseconds using the DUFFNN clustering.

Table 7-11 and Figure 7-10 show the final computed *BMW* vector of the SPECT Heart data by the DUFFNN clustering method.

Table 7-11: The final computed *BMW* vector from the received SPECT Heart data using the DUFFNN method

Vector of the *BMW* for the SPECT HEART dataset									
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8	BMW_9	BMW_{10}
0.049134	0.042387	0.045839	0.044352	0.048043	0.043589	0.043897	0.049406	0.045298	0.047402
BMW_{11}	BMW_{12}	BMW_{13}	BMW_{14}	BMW_{15}	BMW_{16}	BMW_{17}	BMW_{18}	BMW_{19}	BMW_{20}
0.044492	0.043557	0.051054	0.045249	0.040808	0.043584	0.04315	0.04187	0.04384	0.045287
BMW_{21}	BMW_{22}								
0.045333	0.05243								

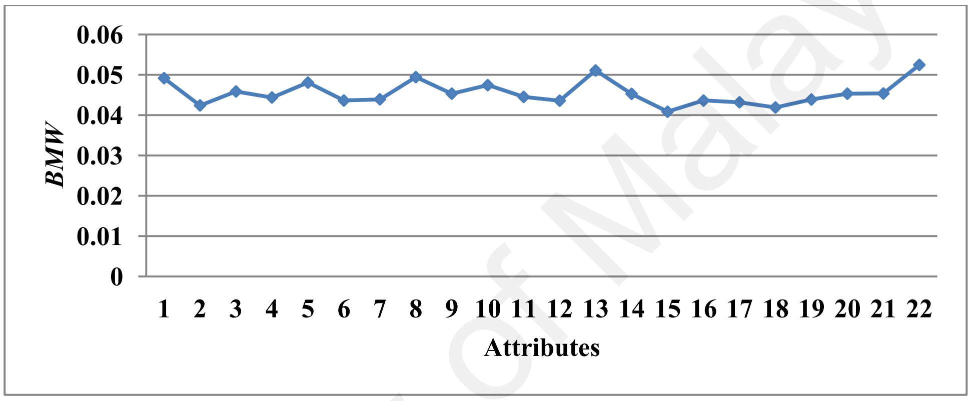


Figure 7-10: The final *BMW* vector of the SPECT Heart dataset using the DUFFNN method

The total thresholds of the received data points were computed on the basis of the final *BMW* vector, and the data points were subsequently clustered. As shown in Figure 7-11, two distinct clusters can be recognized.

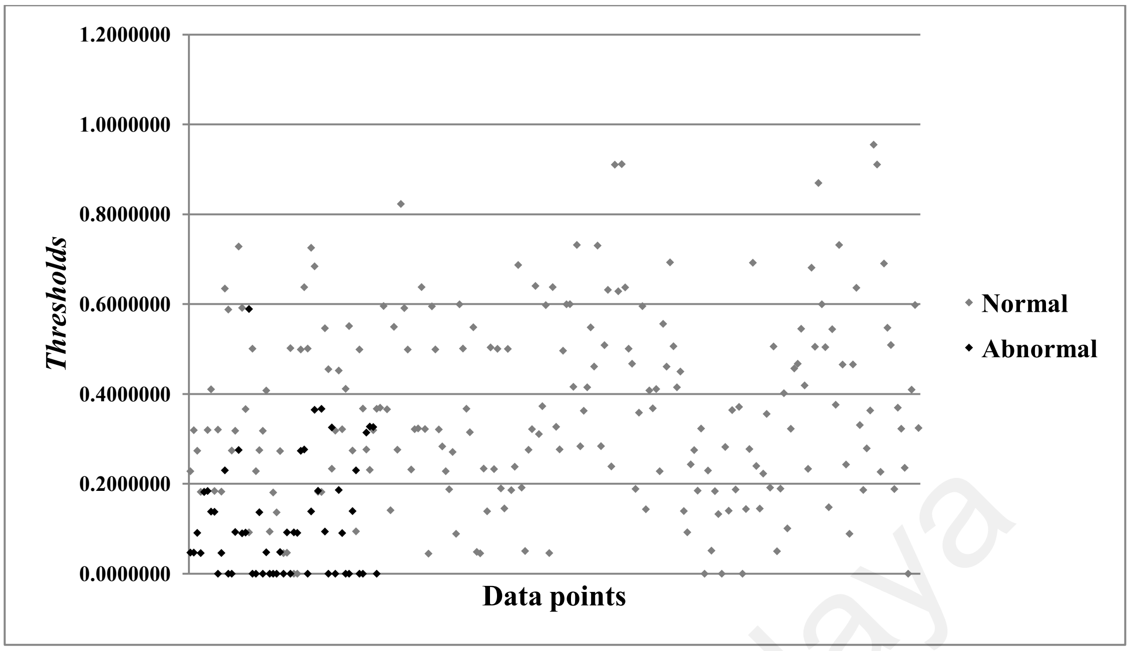


Figure 7-11: The clusters of the DUFFNN method on the SPECT Heart

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.005$, $\delta = \varepsilon$ and $\gamma = 0.005$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.25$. Table 7-12 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and *F-measure* for the SPECT Heart data points.

Table 7-12: Comparison of the clustering results on the SPECT Heart data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN* %	*F-measure%*	Epoch	CPU TIME (milliseconds)
DSOM	85.27	90.09	700	1', 16" and 993
ESOM	66.29	78.77	1	197
Semi-ESOM	73.03	83.02	1	197
DUFFNN	84.44	90.52	1	12.20
DSFFNN	95.50	97.20	1	12.20

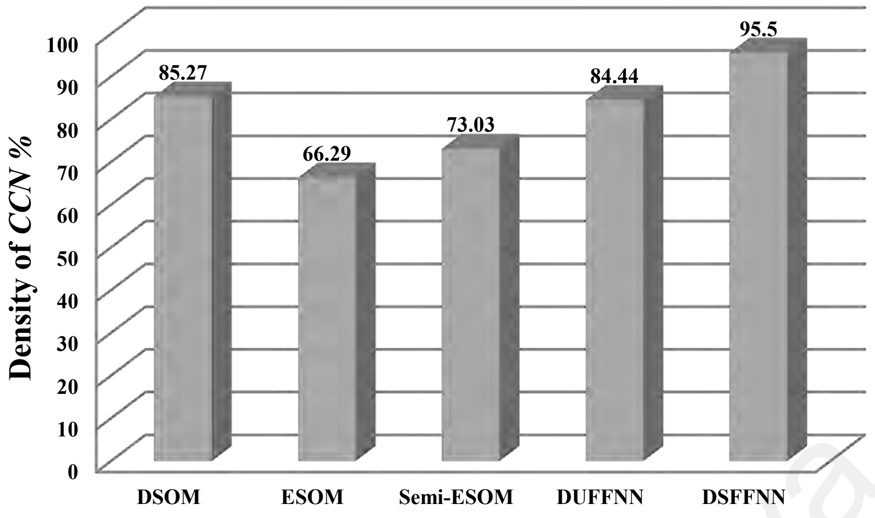


Figure 7-12: Comparison of the clustering density of *CCN %* on the SPECT Heart data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-12 and Figure 7-12, the DSOM clustered the data points with 85.27% for the density of the *CCN* and 90.09% accuracy for the *F-measure* after 700 epochs during 1', 16'' and 993 milliseconds. The ESOM clustered the data points with 66.29% for the density of the *CCN* and 78.77% accuracy by the *F-measure* after 1 epoch in 197 milliseconds. The Semi-ESOM clustered the Spambase data points with 73.03% density of the *CCN* and 83.02% accuracy by the *F-measure.* The DUFFNN clustering method clustered this dataset with 84.44% density of the *CCN* and 90.52% accuracy by the *F-measure* after one epoch in 12.20 milliseconds. The BPN can learn this dataset after 25 epochs with an accuracy of 87.00% by the *F-measure* and the BPN using principal component analysis (PCA) learnt this dataset after 14 epochs training with an accuracy of 73.00% of the *F-measure*. The results of the DSFFNN show 95.50% for the density of the *CCN* and 97.20% accuracy of the *F-measure* with 1 epoch of training. All clustering methods show two distinct clusters for this dataset.

7.2.5 The DUFFNN and DSFFNN Clustering on the SPECTF Heart

The SPECTF Heart dataset from the UCI Repository has two classes: Normal and Abnormal. In order to compute the clustering accuracy of the DUFFNN on the SPECT Heart dataset, the *F-measure* with 10 folds of the test set was considered. The SPECT Heart dataset has 267 cases. Therefore, in each performance, we considered 240 data as the training set and 27 data as the test set.

Table 7-13 shows the details of the computation of the *F-measure* in different performances on the test set for computing clustering accuracy.

Table 7-13: The clustering result of the DUFFNN method on the SPECT Heart

Test subset	*p*	*n*	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	17	10	17	3	0	7	74.07	0.708333	1	0.829268
2	18	9	18	7	0	2	92.59	0.9	1	0.947368
3	24	3	21	0	3	3	77.78	0.875	0.875	0.875
4	22	5	20	2	2	3	81.48	0.869565	0.909091	0.888889
5	18	9	16	3	2	6	70.37	0.727273	0.888889	0.8
6	27	0	17	0	10	0	62.96	1	0.62963	0.772727
7	27	0	18	0	9	0	66.67	1	0.666667	0.8
8	23	4	20	1	3	3	77.78	0.869565	0.869565	0.869565
9	19	8	19	5	0	3	88.89	0.863636	1	0.926829
10	20	7	20	5	0	2	92.59	0.909091	1	0.952381

As shown in Table 7-13, the accuracy of the DUFFNN using the *F-measure* with 10 folds of the test was 86.62% after just one epoch of training in 34.97 milliseconds.

Table 7-14 and Figure 7-13 show the final computed *BMW* vector components of the received SPECTF Heart data by using the DUFFNN clustering method.

Table 7-14: The final computed *BMW* vector from the received SPECTF Heart data using the DUFFNN method

Vector of the final *BMW* for the received SPECTF HEART data									
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8	BMW_9	BMW_{10}
0.020301	0.02324	0.024093	0.025002	0.020584	0.031001	0.016547	0.020498	0.019843	0.020924
BMW_{11}	BMW_{12}	BMW_{13}	BMW_{14}	BMW_{15}	BMW_{16}	BMW_{17}	BMW_{18}	BMW_{19}	BMW_{20}
0.028376	0.027647	0.021853	0.021501	0.02167	0.019151	0.026944	0.01728	0.030749	0.019439
BMW_{21}	BMW_{22}	BMW_{23}	BMW_{24}	BMW_{25}	BMW_{26}	BMW_{27}	BMW_{28}	BMW_{29}	BMW_{30}
0.018362	0.021105	0.021016	0.024655	0.020383	0.02414	0.019098	0.02794	0.018298	0.022476
BMW_{31}	BMW_{32}	BMW_{33}	BMW_{34}	BMW_{35}	BMW_{36}	BMW_{37}	BMW_{38}	BMW_{39}	BMW_{40}
0.02199	0.02616	0.025613	0.01806	0.018809	0.021199	0.026339	0.024922	0.019154	0.021264
BMW_{41}	BMW_{42}	BMW_{43}	BMW_{44}						
0.022103	0.0227	0.03403	0.023541						

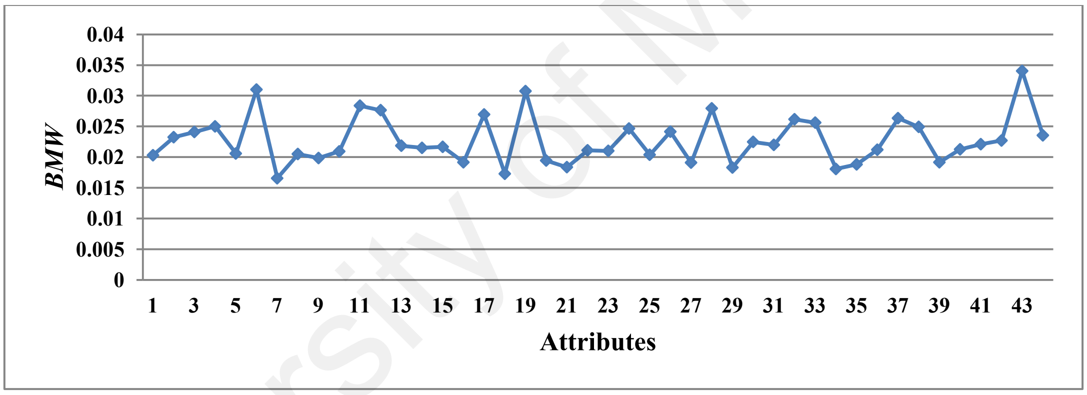


Figure 7-13: The final *BMW* vector of the SPECTF Heart dataset of the DUFFNN method

The total thresholds of the received data points were computed on the basis of the final *BMW* vector, and the data points were subsequently clustered. As shown in Figure 7-14, two distinct clusters can be recognized.

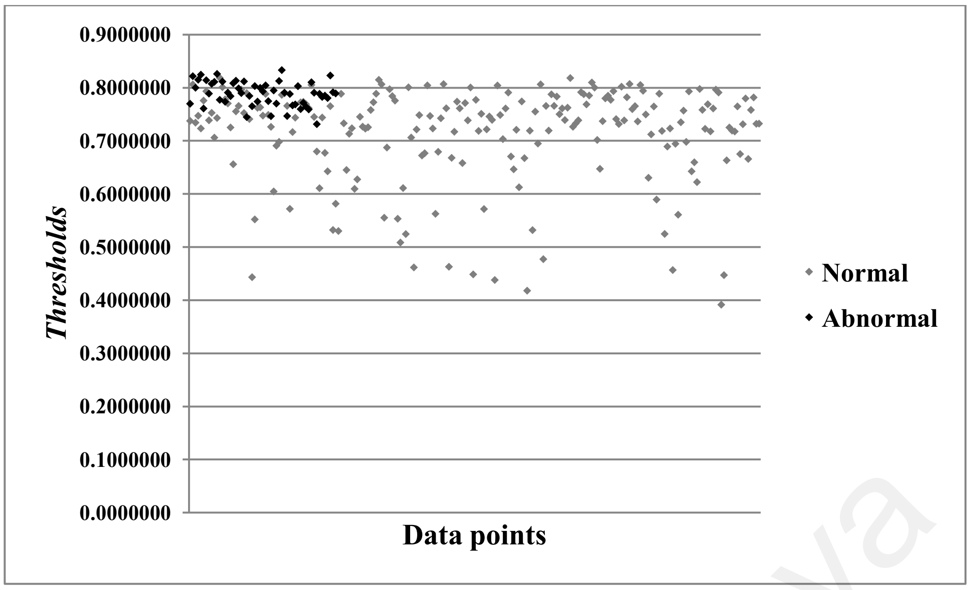


Figure 7-14: The clusters of the DUFFNN method on the SPECTF Heart

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.005$, $\delta = \varepsilon$ and $\gamma = 0.005$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.25$. Table 7-15 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and the *F-measure* for the SPECTF Heart data points.

Table 7-15: Comparison of the clustering results on the SPECTF Heart data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch	CPU TIME (milliseconds)
DSOM	78.65	86.52	700	1' , 38" and 165
ESOM	61.80	77.83	1	342
Semi-ESOM	63.67	80.19	1	342
DUFFNN	78.52	86.62	1	34.97
DSFFNN	100	100	1	34.97

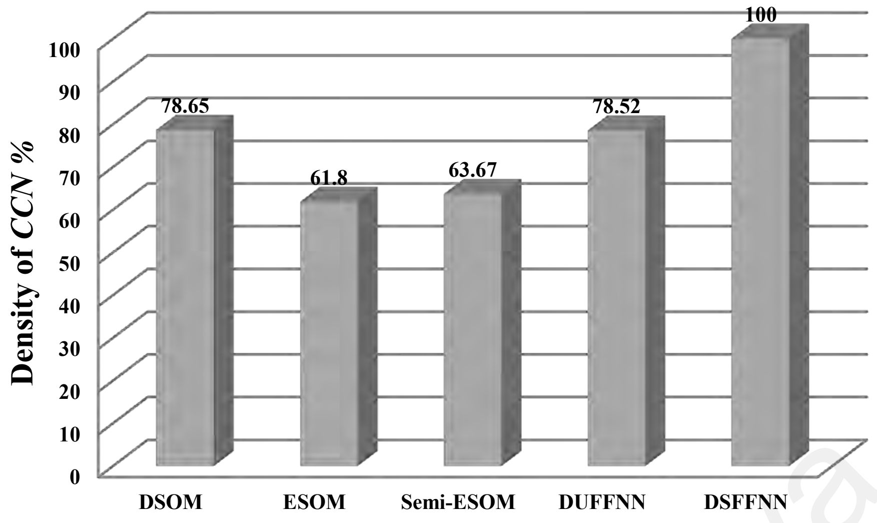


Figure 7-15: Comparison of the clustering density of *CCN %* on the SPECTF Heart data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-15 and Figure 7-15, the DSOM clustered the data points with 78.65% for the density of the *CCN* and 86.52% accuracy by the *F-measure* after 700 epochs in 1' , 38" and 165 milliseconds. The ESOM clustered the data points with 61.80% for the density of the *CCN* and 77.83% accuracy by the *F-measure* after 1 epoch during 342 milliseconds. The semi-ESOM clustered the Spambase data points with 63.67% for the density of the *CCN* and 80.19% accuracy by the *F-measure.* The DUFFNN clustering method clustered this dataset with 78.52% for the density of the *CCN* and 86.62% accuracy by the *F-measure* after one epoch in 34.97 milliseconds. The BPN can learn this dataset after 25 epochs with an accuracy of 79.00% by the *F-measure* and the BPN using PCA learnt this dataset after 14 epochs training with an accuracy of 75.00% by the *F-measure*. The results of the DSFFNN show 100% density of the *CCN* and 100% accuracy by the *F-measure* with 1 epoch for training. All clustering methods show two distinct clusters for this dataset.

7.2.6 The DUFFNN and DSFFNN Clustering on MUSK1

The MUSK1 dataset from the UCI Repository has two classes: Musks and Non-Musks. In order to compute the clustering accuracy of the DUFFNN on the MUSK1 dataset, the *F-measure* with 10 folds of the test set was considered. The Musk1 dataset has 476 cases. Therefore, in each performance, we consider 430 cases as the training set and 48 cases as the test set.

Table 7-16 shows the details of the computation of the *F-measure* in different performances on the test set for computing clustering accuracy.

Table 7-16: The clustering result of the DUFFNN method on the Musk1

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	15	33	3	10	12	23	27.08	0.115385	0.2	0.146341
2	34	14	18	0	16	14	37.50	0.5625	0.529412	0.545455
3	38	10	19	5	19	5	50.00	0.791667	0.5	0.612903
4	44	4	21	0	23	4	43.75	0.84	0.477273	0.608696
5	48	0	24	0	24	0	50.00	1	0.5	0.666667
6	28	20	14	16	14	4	62.50	0.777778	0.5	0.608696
7	16	32	16	11	0	21	56.25	0.432432	1	0.603774
8	15	33	15	21	0	12	75.00	0.555556	1	0.714286
9	10	38	10	18	0	20	58.33	0.333333	1	0.5
10	25	23	5	13	20	10	37.50	0.333333	0.2	0.25

As shown in Table 7-16, the accuracy of the DUFFNN using the *F-measure* with 10 folds of the test was 52.57% after just one epoch of training in 1 second and 839 milliseconds.

Table 7-17 and Figure 7-16 show the final computed *BMW* vector components of the received MUSK1 data using the DUFFNN clustering method.

Table 7-17: The final computed *BMW* vector from the received MUSK1 data using the DUFFNN method

Vector of the final *BMW* for the received Musk1 data									
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8	BMW_9	BMW_{10}
0.001904	0.007284	0.008115	0.006095	0.013277	0.006319	0.006934	0.006201	0.006132	0.002867
BMW_{11}	BMW_{12}	BMW_{13}	BMW_{14}	BMW_{15}	BMW_{16}	BMW_{17}	BMW_{18}	BMW_{19}	BMW_{20}
0.006883	0.003441	0.006037	0.006483	0.005469	0.009414	0.009752	0.004449	0.003581	0.001959
BMW_{21}	BMW_{22}	BMW_{23}	BMW_{24}	BMW_{25}	BMW_{26}	BMW_{27}	BMW_{28}	BMW_{29}	BMW_{30}
0.00267	0.005639	0.004471	0.010562	0.008453	0.004952	0.0081	0.005643	0.002777	0.009472
BMW_{31}	BMW_{32}	BMW_{33}	BMW_{34}	BMW_{35}	BMW_{36}	BMW_{37}	BMW_{38}	BMW_{39}	BMW_{40}
0.010655	0.007139	0.004034	0.008164	0.004536	0.00646	0.011952	0.005675	0.007572	0.005078
BMW_{41}	BMW_{42}	BMW_{43}	BMW_{44}	BMW_{45}	BMW_{46}	BMW_{47}	BMW_{48}	BMW_{49}	BMW_{50}
0.00817	0.004171	0.003545	0.007424	0.006327	0.003427	0.007848	0.004693	0.005164	0.002236
BMW_{51}	BMW_{52}	BMW_{53}	BMW_{54}	BMW_{55}	BMW_{56}	BMW_{57}	BMW_{58}	BMW_{59}	BMW_{60}
0.002047	0.00583	0.006205	0.003075	0.006655	0.002251	0.007617	0.009336	0.00358	0.005953
BMW_{61}	BMW_{62}	BMW_{63}	BMW_{64}	BMW_{65}	BMW_{66}	BMW_{67}	BMW_{68}	BMW_{69}	BMW_{70}
0.008415	0.009311	0.00491	0.010308	0.007203	0.003889	0.013175	0.00243	0.007645	0.002844
BMW_{71}	BMW_{72}	BMW_{73}	BMW_{74}	BMW_{75}	BMW_{76}	BMW_{77}	BMW_{78}	BMW_{79}	BMW_{80}
0.008343	0.00651	0.005426	0.005536	0.004047	0.009249	0.007558	0.00478	0.004087	0.00533
BMW_{81}	BMW_{82}	BMW_{83}	BMW_{84}	BMW_{85}	BMW_{86}	BMW_{87}	BMW_{88}	BMW_{89}	BMW_{90}
0.003836	0.005383	0.00234	0.008231	0.007726	0.006663	0.003217	0.004397	0.004205	0.009043
BMW_{91}	BMW_{92}	BMW_{93}	BMW_{94}	BMW_{95}	BMW_{96}	BMW_{97}	BMW_{98}	BMW_{99}	BMW_{100}
0.009071	0.006601	0.003965	0.001891	0.003279	0.006831	0.005427	0.006354	0.009716	0.006489
BMW_{101}	BMW_{102}	BMW_{103}	BMW_{104}	BMW_{105}	BMW_{106}	BMW_{107}	BMW_{108}	BMW_{109}	BMW_{110}
0.00718	0.003033	0.006162	0.006432	0.005628	0.003237	0.007276	0.005715	0.002115	0.005837
BMW_{111}	BMW_{112}	BMW_{113}	BMW_{114}	BMW_{115}	BMW_{116}	BMW_{117}	BMW_{118}	BMW_{119}	BMW_{120}
0.004958	0.003658	0.004631	0.005866	0.006539	0.003587	0.009881	0.008634	0.008303	0.007703
BMW_{121}	BMW_{122}	BMW_{123}	BMW_{124}	BMW_{125}	BMW_{126}	BMW_{127}	BMW_{128}	BMW_{129}	BMW_{130}
0.008981	0.006791	0.007775	0.004613	0.008823	0.004725	0.003702	0.008954	0.002726	0.004571
BMW_{131}	BMW_{132}	BMW_{133}	BMW_{134}	BMW_{135}	BMW_{136}	BMW_{137}	BMW_{138}	BMW_{139}	BMW_{140}
0.005336	0.007745	0.002839	0.002939	0.005629	0.003648	0.006744	0.004151	0.005545	0.002566
BMW_{141}	BMW_{142}	BMW_{143}	BMW_{144}	BMW_{145}	BMW_{146}	BMW_{147}	BMW_{148}	BMW_{149}	BMW_{150}
0.00219	0.004705	0.006233	0.007283	0.013224	0.012268	0.008316	0.003957	0.003644	0.00699
BMW_{151}	BMW_{152}	BMW_{153}	BMW_{154}	BMW_{155}	BMW_{156}	BMW_{157}	BMW_{158}	BMW_{159}	BMW_{160}
0.005458	0.003988	0.003697	0.00596	0.004732	0.006086	0.011766	0.006248	0.010729	0.003068
BMW_{161}	BMW_{162}	BMW_{163}	BMW_{164}	BMW_{165}	BMW_{166}				
0.004954	0.002789	0.007779	0.007997	0.009746	0.003192				

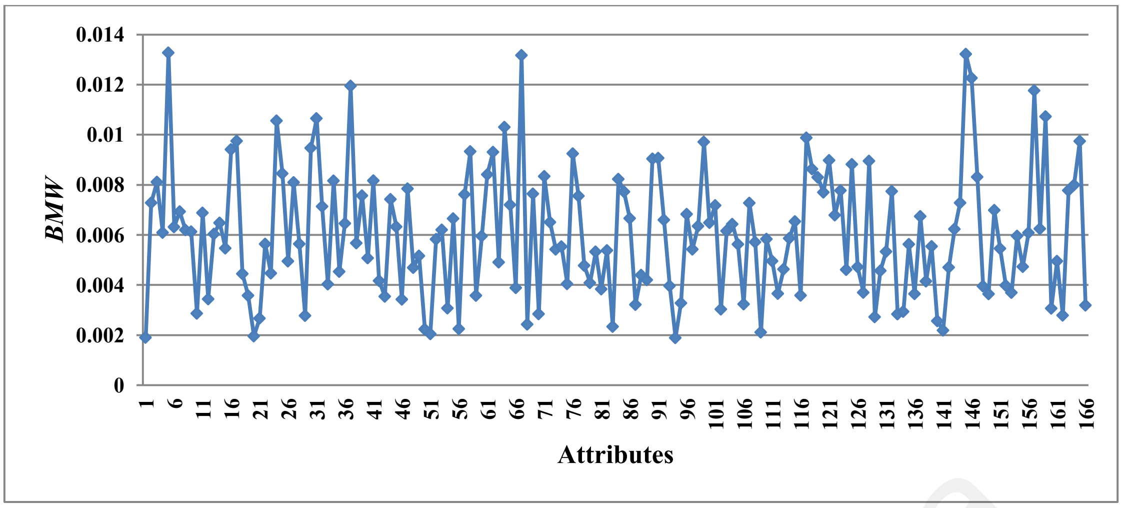


Figure 7-16: The final *BMW* vector of the MUSK1 dataset by the DUFFNN method

The total thresholds of the received data points were computed on the basis of the final *BMW* vector, and the data points were sequentially clustered. As shown in Figure 7-17, two distinct clusters can be recognized.

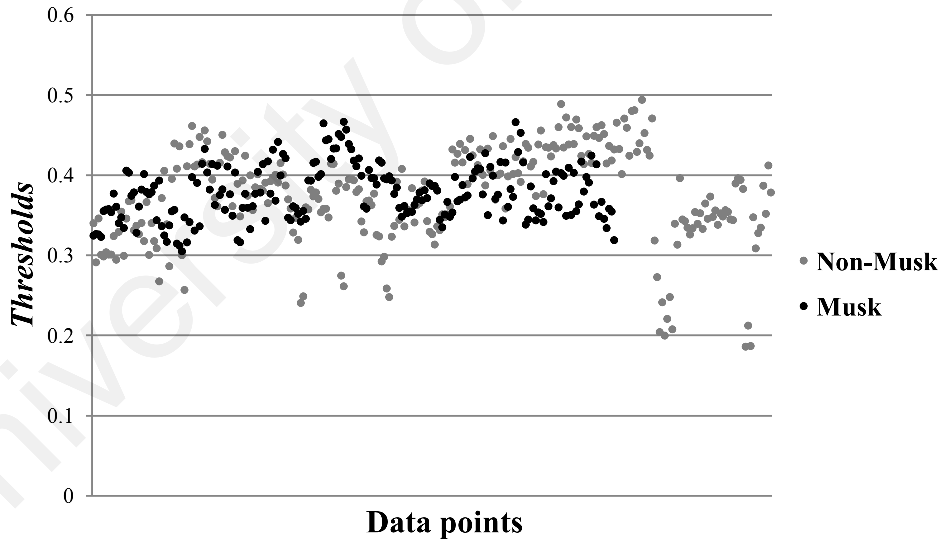


Figure 7-17: The clusters of the DUFFNN method on the MUSK1

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.006$, $\delta = \varepsilon$ and $\gamma = 0.007$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.28$. Table 7-18 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and the *F-measure* for the MUSK1 data points.

Table 7-18: Comparison of the clustering results on the MUSK1 data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch	CPU TIME (milliseconds)
DSOM	42.44	48.19	700	4' , 52" and 562
ESOM	42.44	48.19	1	1" and 1
Semi-ESOM	57.98	62.83	1	1" and 1
DUFFNN	49.79	52.57	1	382.49
DSFFNN	100	100	1	382.49

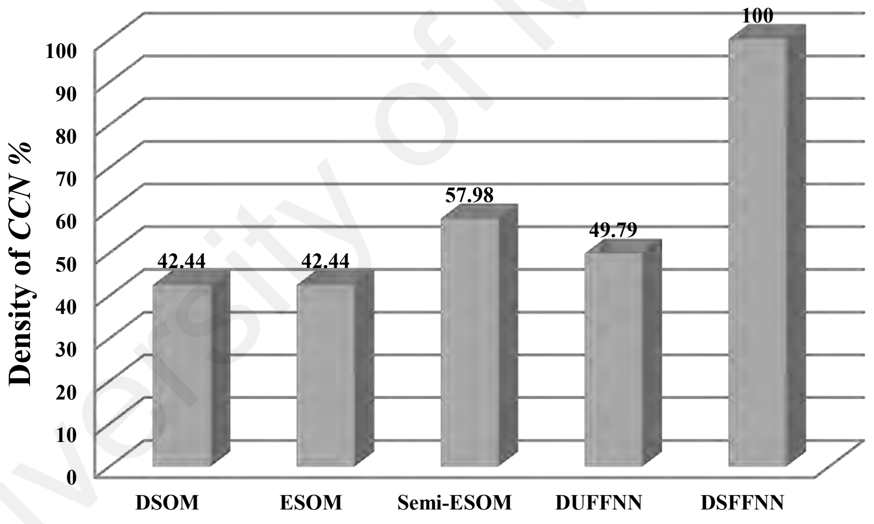


Figure 7-18: Comparison of the clustering density of *CCN %* on the MUSK1 data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-18 and Figure 7-18, the DSOM clustered the data points with 42.44% for the density of the *CCN* and 48.19% accuracy by the *F-measure* after 700 epochs in 4' , 52" and 562 milliseconds. The ESOM just like the DOM clustered the data points with 42.44% for the density of the *CCN* and 48.19% accuracy by the *F-measure* but after only 1 epoch taking 1" and 1 milliseconds. The semi-ESOM clustered the MUSK1 data points with 57.98% for the density of the *CCN* and 62.83% accuracy by the *F-measure.* The DUFFNN clustering method clustered this dataset with 49.79% for the density of the *CCN* and 52.57% accuracy by the *F-measure* after one epoch in 382.49 milliseconds. The BPN can learn this dataset after 100 epochs with an accuracy of 75.00% by the *F-measure*. The results of the DSFFNN show 100% for the density of the *CCN* and 100% accuracy by the *F-measure* after only 1 epoch of training. All clustering methods show two distinct clusters for this dataset.

7.2.7 The DUFFNN and DSFFNN Clustering on the MUSK2

In order to compute the clustering accuracy of the DUFFNN on the MUSK2 data points, *F-measure* with 10 folds of the test set was considered. Table 7-19 shows the details.

Table 7-19: The clustering result of the DUFFNN method of the Musk2

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	571	89	537	11	34	78	83.03	0.873171	0.940455	0.905565
2	579	81	534	10	45	71	82.42	0.882645	0.92228	0.902027
3	626	34	595	9	31	25	91.52	0.959677	0.950479	0.955056
4	582	78	511	10	71	68	78.94	0.882556	0.878007	0.880276
5	623	37	456	13	167	24	71.06	0.95	0.731942	0.826836
6	607	53	464	8	143	45	71.52	0.911591	0.764415	0.831541
7	621	39	517	10	104	29	79.85	0.946886	0.832528	0.886033
8	546	114	419	9	127	105	64.85	0.799618	0.767399	0.783178
9	400	260	356	68	44	192	64.24	0.649635	0.89	0.751055
10	427	233	349	25	78	208	56.67	0.626571	0.81733	0.70935

As shown in Table 7-19, the accuracy of the DUFFNN clustering by using the *F-measure* with 10 folds of the test set was 84.31% after just one epoch of training in 24" and 127 milliseconds.

Table 7-20 and Figure 7-19 show the final computed *BMW* vector components of the received MUSK2 data using the DUFFNN clustering method.

Table 7-20: The final *BMW* vector of the MUSK2 using the DUFFNN method

Vector of the final *BMW* for the received Musk2 data									
BMW_{1}	BMW_{2}	BMW_{3}	BMW_{4}	BMW_{5}	BMW_{6}	BMW_{7}	BMW_{8}	BMW_{9}	BMW_{10}
0.004979	0.00811	0.007209	0.006912	0.01085	0.006452	0.007426	0.003967	0.006969	0.004096
BMW_{11}	BMW_{12}	BMW_{13}	BMW_{14}	BMW_{15}	BMW_{16}	BMW_{17}	BMW_{18}	BMW_{19}	BMW_{20}
0.005264	0.004024	0.003666	0.004983	0.004491	0.008007	0.008793	0.003662	0.003994	0.00377
BMW_{21}	BMW_{22}	BMW_{23}	BMW_{24}	BMW_{25}	BMW_{26}	BMW_{27}	BMW_{28}	BMW_{29}	BMW_{30}
0.003676	0.006154	0.004953	0.008663	0.007852	0.005894	0.007276	0.007464	0.004961	0.008334
BMW_{31}	BMW_{32}	BMW_{33}	BMW_{34}	BMW_{35}	BMW_{36}	BMW_{37}	BMW_{38}	BMW_{39}	BMW_{40}
0.009754	0.006034	0.003968	0.007862	0.005849	0.004044	0.010295	0.006357	0.007718	0.006441
BMW_{41}	BMW_{42}	BMW_{43}	BMW_{44}	BMW_{45}	BMW_{46}	BMW_{47}	BMW_{48}	BMW_{49}	BMW_{50}
0.007821	0.004108	0.003606	0.004995	0.004567	0.004389	0.007494	0.004222	0.003778	0.003419
BMW_{51}	BMW_{52}	BMW_{53}	BMW_{54}	BMW_{55}	BMW_{56}	BMW_{57}	BMW_{58}	BMW_{59}	BMW_{60}
0.003258	0.006377	0.006932	0.004449	0.00693	0.003102	0.007348	0.008249	0.005205	0.006733
BMW_{61}	BMW_{62}	BMW_{63}	BMW_{64}	BMW_{65}	BMW_{66}	BMW_{67}	BMW_{68}	BMW_{69}	BMW_{70}
0.008153	0.008655	0.004134	0.008733	0.007407	0.005216	0.010683	0.004947	0.007093	0.005101
BMW_{71}	BMW_{72}	BMW_{73}	BMW_{74}	BMW_{75}	BMW_{76}	BMW_{77}	BMW_{78}	BMW_{79}	BMW_{80}
0.00776	0.007545	0.004475	0.004373	0.003703	0.00845	0.007671	0.004528	0.004209	0.004811
BMW_{81}	BMW_{82}	BMW_{83}	BMW_{84}	BMW_{85}	BMW_{86}	BMW_{87}	BMW_{88}	BMW_{89}	BMW_{90}
0.005043	0.006863	0.003563	0.007002	0.007237	0.007328	0.005594	0.005357	0.00574	0.008271
BMW_{91}	BMW_{92}	BMW_{93}	BMW_{94}	BMW_{95}	BMW_{96}	BMW_{97}	BMW_{98}	BMW_{99}	BMW_{100}
0.007865	0.005341	0.006465	0.003516	0.003211	0.007248	0.006413	0.007147	0.009298	0.006919
BMW_{101}	BMW_{102}	BMW_{103}	BMW_{104}	BMW_{105}	BMW_{106}	BMW_{107}	BMW_{108}	BMW_{109}	BMW_{110}
0.007512	0.004194	0.004843	0.004909	0.004829	0.004165	0.007385	0.004374	0.004735	0.004294
BMW_{111}	BMW_{112}	BMW_{113}	BMW_{114}	BMW_{115}	BMW_{116}	BMW_{117}	BMW_{118}	BMW_{119}	BMW_{120}
0.004344	0.004423	0.006095	0.006572	0.006286	0.004139	0.008496	0.008231	0.007904	0.007148
BMW_{121}	BMW_{122}	BMW_{123}	BMW_{124}	BMW_{125}	BMW_{126}	BMW_{127}	BMW_{128}	BMW_{129}	BMW_{130}
0.008146	0.006795	0.007315	0.004979	0.006593	0.003027	0.004985	0.008164	0.003727	0.007894
BMW_{131}	BMW_{132}	BMW_{133}	BMW_{134}	BMW_{135}	BMW_{136}	BMW_{137}	BMW_{138}	BMW_{139}	BMW_{140}
0.003467	0.005972	0.003673	0.003682	0.00381	0.005599	0.007382	0.005526	0.005099	0.003667
BMW_{141}	BMW_{142}	BMW_{143}	BMW_{144}	BMW_{145}	BMW_{146}	BMW_{147}	BMW_{148}	BMW_{149}	BMW_{150}
0.003529	0.006354	0.006419	0.006725	0.010709	0.010109	0.008924	0.005774	0.005377	0.006918
BMW_{151}	BMW_{152}	BMW_{153}	BMW_{154}	BMW_{155}	BMW_{156}	BMW_{157}	BMW_{158}	BMW_{159}	BMW_{160}
0.005664	0.006998	0.006783	0.006431	0.00479	0.006987	0.009542	0.006788	0.008947	0.00487
BMW_{161}	BMW_{162}	BMW_{163}	BMW_{164}	BMW_{165}	BMW_{166}				
0.004896	0.003695	0.004871	0.001694	0.003514	0.007096				

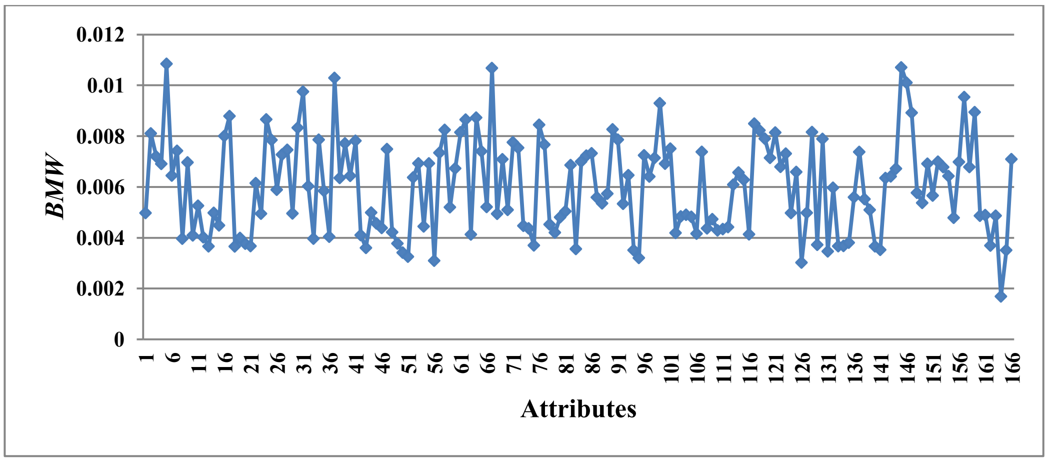


Figure 7-19: The final *BMW* vector of the MUSK2 using the DUFFNN method

The total thresholds of the received data points were computed based on the final *BMW* vector, and the data points were subsequently clustered. As shown in Figure 7-20, two distinct clusters can be recognized.

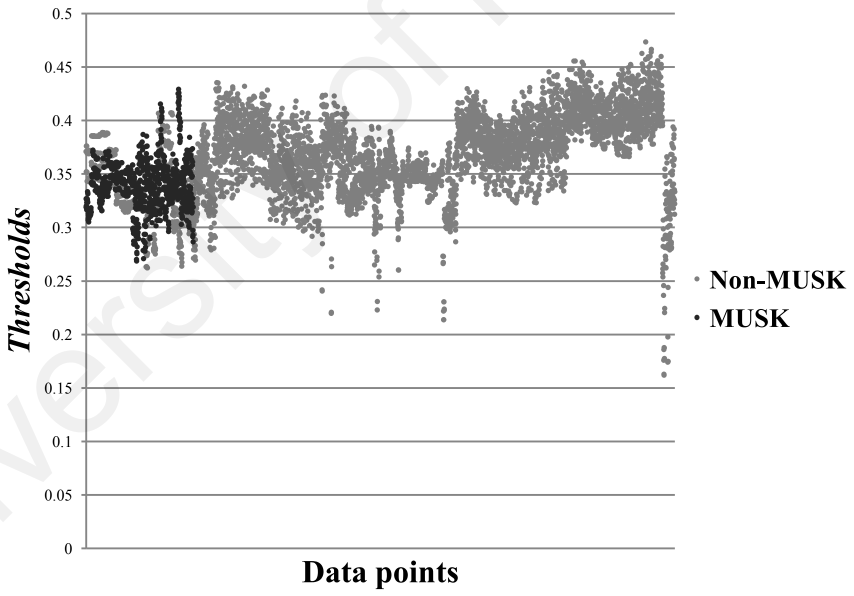


Figure 7-20: The clusters of the DUFFNN method on the MUSK2

We compared the results of the proposed DUFFNN method with the results of some related ODUFFNN and UFFNN methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.006$, $\delta = \varepsilon$ and $\gamma = 0.007$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.28$. Table 7-21 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* and the *F-measure* for the MUSK2 data points.

Table 7-21: Comparison of the clustering results on the MUSK2 data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN* %	*F-measure%*	Epoch	CPU TIME (milliseconds)
DSOM	60.28	41.40	700	41' , 1" and 633
ESOM	70.58	56.40	1	28" and 1
Semi-ESOM	78.34	87.19	1	28' and 1
DUFFNN	74.41	84.31	1	27" and 572
DSFFNN	100	100	1	27" and 572

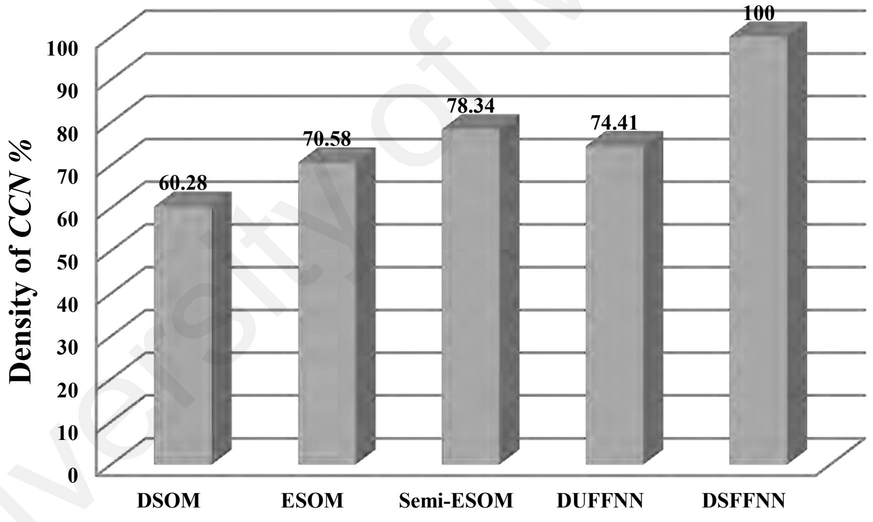


Figure 7-21: Comparison of the clustering density of *CCN %* on the MUSK2 data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-21 and Figure 7-21, the density of *CCN* of the DSOM clustered the data points with 60.28% for the density of the *CCN* and 41.40% accuracy by the *F-measure* after 700 epochs in 41' , 1" and 633 milliseconds. The ESOM clustered the Spambase data points with 70.58% for the density of the *CCN* and 56.40% accuracy by the *F-measure* after 1 epoch in 28" and 1 milliseconds. The semi-ESOM clustered the Spambase data points with 78.34% for the density of the *CCN* and 87.19% accuracy by the *F-measure.* The DUFFNN clustering method clustered this dataset with 74.41% for the density of the *CCN* and 84.31% accuracy by the *F-measure* after one epoch in 27" and 572 milliseconds. The BPN learnt this dataset after 100 epochs with an accuracy of 67.00% by using the *F-measure*. The results of the DSFFNN show 100% for the density of the *CCN* and 100% accuracy for the *F-measure* after 1 epoch of training. All clustering methods show two distinct clusters for this dataset.

7.2.8 The DUFFNN and DSFFNN Clustering on Arcene

In order to compute the clustering accuracy of the DUFFNN on the Arcene data points, an *F-measure* with 10 folds of the test set was considered. Table 7-22 shows the details.

Table 7-22: The clustering result of the DUFFNN method on the Arcene

Test subset	p	n	t_p	t_n	f_p	f_n	*CCN%*	*Recall*	*Precision*	*F-measure*
1	13	7	7	5	6	2	60.00	0.777777778	0.538461538	0.636363636
2	12	8	9	2	3	6	55.00	0.6	0.75	0.666666667
3	11	9	7	4	4	5	55.00	0.583333333	0.636363636	0.608695652
4	11	9	6	6	5	3	60.00	0.666666667	0.545454545	0.6
5	9	11	7	6	2	5	65.00	0.583333333	0.777777778	0.666666667
6	10	10	6	6	4	4	60.00	0.6	0.6	0.6
7	13	7	8	5	5	2	65.00	0.8	0.615384615	0.695652174
8	11	9	8	5	3	4	65.00	0.666666667	0.727272727	0.695652174
9	12	8	8	5	4	3	65.00	0.727272727	0.666666667	0.695652174
10	10	10	8	6	2	4	70.00	0.666666667	0.8	0.727272727

As shown in Table 7-22, the accuracy of the DUFFNN clustering using the *F-measure* with 10 folds of the test set was 65.93% after just one epoch of training in 13” and 447 milliseconds. Table 7-23 shows the final computed *BMW* vector of the Arcene dataset by the DUFFNN clustering method.

Table 7-23: The *BMW* vector of the Arcene dataset by the DUFFNN method

Vector of the *SW* for the Arcene dataset									
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8	BMW_9	BMW_{10}
0.000105	7.08E-05	0.000128	0.000146	9.25E-05	0.000165	0.000193	9.51E-05	0.000103	8.65E-05
BMW_{11}	BMW_{12}	BMW_{13}	BMW_{14}	BMW_{15}	BMW_{16}	BMW_{17}	BMW_{18}	BMW_{19}	BMW_{20}
8.13E-05	9.38E-05	8.67E-05	8.91E-05	0.000183	8.28E-05	9.52E-05	9.20E-05	0.000109	8.57E-05
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
BMW_{99991}	BMW_{99992}	BMW_{99993}	BMW_{99994}	BMW_{99995}	BMW_{99996}	BMW_{99997}	BMW_{99998}	BMW_{9999}	BMW_{10000}
0.000117	0.000221	1.00E-04	0.000101	9.11E-05	0.00014	0.000156	9.47E-05	9.27E-05	0.000251

The total thresholds of the received data points were computed on the basis of the final *BMW* vector, and the data points were subsequently clustered. As shown in Figure 7-22, two distinct clusters can be recognized.

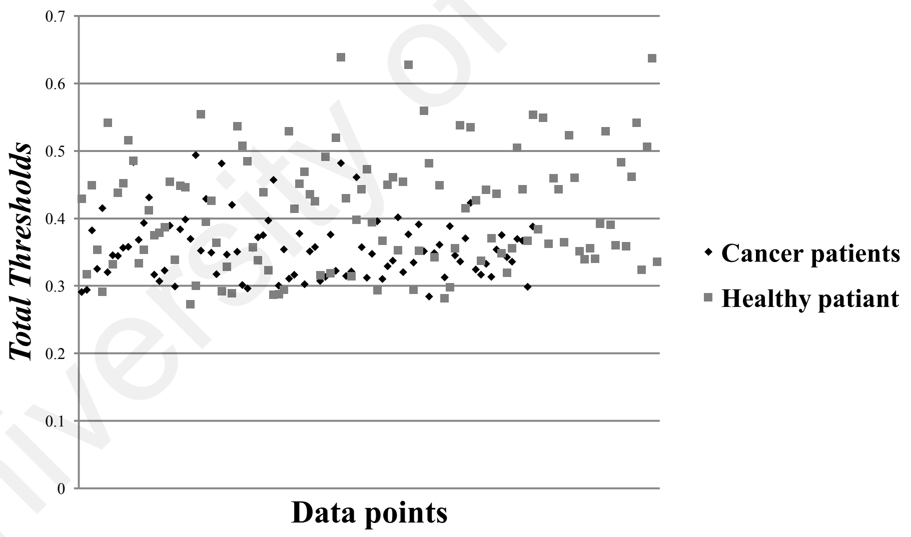


Figure 7-22: The clusters of the DUFFNN method on the Arcene

We compared the results of the proposed DUFFNN and DSFFNN methods with the results of some related methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.005$, $\delta = \varepsilon$ and $\gamma = 0.005$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.25$. Table 8.24 shows the speed of processing on the basis of the number of epochs and the accuracy on the basis of the density of the *CCN* in the Arcene data points.

Table 7-24: Comparison of the clustering results on the Arcene data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	Accuracy by *F-measure %*	Epoch	CPU TIME (milliseconds)
K-means	59.00	-	10	-
SOM	53.00	58.04	140	-
Semi-SOM	64.00	67.86	140	-
DSOM	47.00	52.68	20	43' , 12” and 943
ESOM	48.00	53.57	1	56” and 998
Semi-ESOM	60.50	63.93	1	56” and 998
DUFFNN	62.00	65.93	1	13” and 447
DSFFNN	100	100	1	13” and 447

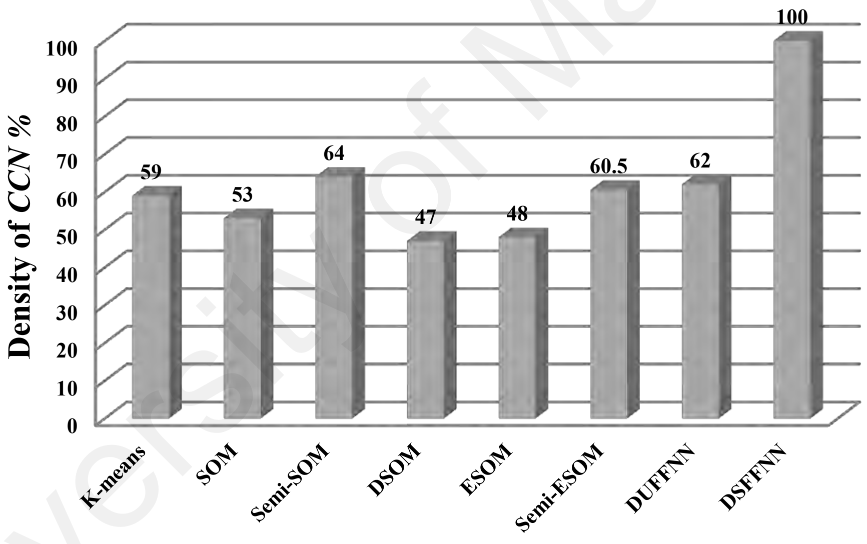


Figure 7-23: Comparison of the clustering density of *CCN %* on the Arcene data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-24 and Figure 7-23, the *K*-means produced 59.00% for the density of the *CCN* after 10 epochs (Sárközy, et al., 2012). The SOM produced 53.00% for the density of the *CCN* and 58.04% using the *F-measure* after 140 epochs. The semi-SOM clustered the data with 64.00% for the density of the *CCN* and 67.86% using the *F-measure*. The DSOM clustered the data points with 47.00% for the density of the *CCN* and 52.68% accuracy by the *F-measure* after 20 epochs in 43' , 12” and 943 milliseconds. The ESOM clustered the data points with 48.00% for the density of the *CCN* and 53.57% accuracy by the *F-measure* after 1 epoch in 56” and 998 milliseconds. The semi-ESOM clustered the data points with 60.50% for the density of the *CCN* and 63.93% accuracy by the *F-measure*. The DUFFNN clustering method clustered this dataset with 62.00% for the density of the *CCN* and 65.93% accuracy by the *F-measure* after one epoch in 13” and 447 milliseconds. The results of the DSFFNN show 100% for the density of the *CCN* and 100% accuracy for the *F-measure* with 1 epoch of training. Recently, Veenu Mangat and Renu Vig (Mangat & Vig, 2014) reported classification of the Arcene dataset by several classification methods such as the *K*-NN. The *K*-NN (*K*=10) was able to classify the Arcene dataset with 77.00% accuracy by the *F-measure* after several epochs and 10 times executing the method. All clustering methods show two distinct clusters for this dataset.

7.2.9 The DUFFNN and DSFFNN Clustering on Yeast

The Yeast dataset from the UCI Repository contains 1484 samples with 8 attributes and 10 classes. Table 7-25 and Figure 7-24 shows the computed *BMW* vector of the Yeast dataset by the DUFFNN clustering method.

Table 7-25: The final computed *BMW* vector from the received Yeast data using the DUFFNN method

Vector of the *SW* for the Yeast dataset							
BMW_1	BMW_2	BMW_3	BMW_4	BMW_5	BMW_6	BMW_7	BMW_8
0.0887	0.0826	0.0682	0.0688	0.1975	0.1973	0.243	0.054

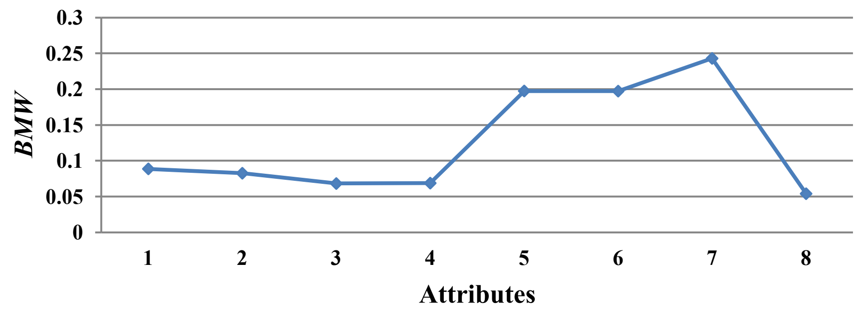


Figure 7-24: The final *BMW* vector of the Yeast dataset by the DUFFNN method

The total thresholds of the received data points were computed on the basis of the final *BMW* vector, and the data points were subsequently clustered. Finally, the clusters that the RUFFNN can recognized are shown in Table 7-26.

Table 7-26: The clusters of the DUFFNN method on the Yeast dataset

	Class CYT	Class ERL	Class EXC	Class ME1	Class ME2	Class ME3	Class MIT	Class NUC	Class POX	Class VAX
Cluster 1	169	0	1	0	2	44	87	157	4	6
Cluster 2	0	3	0	2	1	0	0	0	0	0
Cluster 3	4	0	4	11	7	2	2	5	2	1
Cluster 4	5	0	4	12	10	1	5	3	3	2
Cluster 5	5	0	6	6	7	4	10	9	0	5
Cluster 6	62	0	0	0	0	17	36	45	2	3
Cluster 7	100	0	1	0	2	15	60	61	2	4
Cluster 8	144	0	0	0	2	54	68	148	5	9
Cluster 9	1	0	8	6	0	0	0	1	4	1
Cluster 10	9	0	0	0	1	1	8	10	0	2

As shown in Table 7-26, clustering of the Yeast dataset was so difficult and the clusters often covered and shared members of other near neighbour clusters. We compared the results of the proposed DUFFNN method with the results of some related methods. In the ESOM, we considered the parameters: $\beta = 0.8$, $\varepsilon = 0.005$, $\delta = \varepsilon$ and $\gamma = 0.005$. In the DSOM, we considered the parameters: $\varepsilon = 0.100$ and $\delta = 1.25$. Table 7-27 shows the speed of processing based on the number of epochs and the accuracy based on the density of the *CCN* in the Yeast dataset.

Table 7-27: Comparison of the clustering results on the Yeast data points by the DUFFNN, DSFFNN and some related methods

The clustering method	Density of *CCN %*	*F-measure%*	Epoch	CPU TIME (milliseconds)
DSOM	27.29	24.53	20	11” and 387
ESOM	29.31	17.63	1	37” and 681
Semi-ESOM	36.79	20.72	1	37” and 681
DUFFNN	28.71	27.25	1	1” and 373
DSFFNN	100	100	1	1” and 373

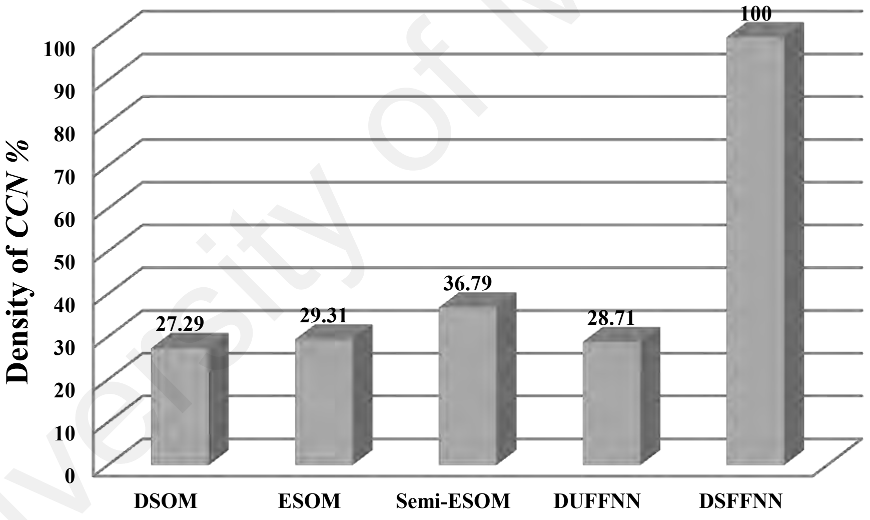


Figure 7-25: Comparison of the clustering density of *CCN %* on the Yeast data points by the DUFFNN, DSFFNN and some related methods

As shown in Table 7-27 and Figure 7-25, the DSOM clustered the Yeast data with 27.29% for the density of the *CCN* and 24.53% accuracy by the *F-measure* after 20 epoch in 11” and 387 milliseconds. The ESOM clustered the Yeast data with 29.31% for the density of the *CCN* and 17.63% accuracy by the *F-measure* after 1 epoch taking 37” and 681 milliseconds. The semi-ESOM clustered this data with 36.79% for the density of the *CCN* and 20.72% accuracy by the *F-measure*. The DUFFNN clustering method clustered this dataset with 28.71% for the density of the *CCN* and 27.25% accuracy by the *F-measure* after 1 epoch just in 1" and 373 milliseconds. The DSFFNN clustering method clustered this dataset with 100% for the density of the *CCN* and also by the *F-measure.* Several literature reported the difficulty of clustering or classification of the Yeast dataset. As Longadge et al. reported, classification of the Yeast dataset by several classification methods such as the *K*-NN (Longadge & Dongre, 2013). The *K*-NN (*K*=3) was able to classify the Yeast dataset with 0.11% accuracy by the *F-measure* after several epochs and times running the method. Also in 2014, Ahirwar reported the *K-means* was able to classify the Yeast dataset with 65.00% accuracy by the *F-measure* after several epochs (Ahirwar, 2014).

7.2.10 The DSFFNN Clustering on the Breast Cancer (UMMC)

Table 7-28 shows the results of the implementation of the proposed DSFFNN method on the UMMC breast cancer data points. The number of cases of each subset, CPU time usage per second for training each subset during one epoch, and the accuracy of the semi-clustering of each subset of the breast cancer dataset based on the *F-measure* with 10 folds of the test dataset using the RSFFNN clustering model are provided in Table 7-28.

Table 7-28: The results of implementation of the DSFFNN for each subset of the breast cancer from UMMC

Year	Density *CCN* (%)	The number of cases in each subset	Epoch	CPU Time usage (milliseconds)	Accuracy of the DSFFNN (*F-measure%*)
1st year	99.15	827	1	1', 05" and 2	99.43
2nd year	98.96	673	1	882	98.69
3rd year	98.93	561	1	501	98.93
4th year	98.18	440	1	252	98.14
5th year	100	355	1	137.39	99.99
6th year	100	270	1	40.72	100
7th year	100	200	1	28.93	100
8th year	100	124	1	25.49	100
9th year	100	56	1	22.95	100

Table 7-28 shows that the training process for each subset of the breast cancer dataset was in one epoch between [22.95, 1', 05" and 2] milliseconds of CPU time; and the accuracy of the DSFFNN by the *F-measure* for the breast cancer sub-datasets was between [98.14% - 100%]. The results of the SOM-BPN and the PCA-BPN are shown in Table 7-29 for every subset of the UMMC breast cancer dataset.

Table 7-29: Comparison of the results of the PCA-BPN, the SOM-BPN and the DSFFNN for each subset of the UMMC breast cancer dataset

Year	PCA-BPN (*F-measure%*)	SOM-BPN (*F-measure%*)	DSFFNN (*F-measure%*)
1st year	76.00	82.00	99.43
2nd year	63.00	72.00	98.69
3rd year	62.00	71.00	98.93
4th year	77.00	78.00	98.14
5th year	83.00	86.00	99.99
6th year	93.00	93.00	100
7th year	98.00	98.00	100
8th year	99.00	99.00	100
9th year	99.00	99.00	100

The results of Table 7-29 shows the accuracy of implementation of the PCA-BPN model by the *F-measure* for the UMMC breast cancer dataset was between [62%-99%], and the accuracies of implementation of the SOM-BPN model by the *F-measure* for each subset of the breast cancer dataset was between [71%- 99%]. However, the DSFFNN clustering method had better results with accuracy between [98.14%-100%] by the *F-measure*.

7.3 Summary

This chapter illustrated the experimental results and evaluation of the DUFFNN and the DUFFNN clustering performances, and compared the proposed methods to several related clustering methods using the various datasets from UCI Repository and the UMMC. For experimentation, the training speed was measured by the number of epochs and the CPU time usage. The accuracy of the clustering methods was measured by employing the *F-measure* with 10 folds of the test set, and also through the number of clusters and the density of the *correctly classified nodes (CCN)*. The results of the experiments showed the superior outcomes of the proposed DUFFNN clustering method. Also, this chapter showed the results of the DSFFNN on the nine datasets from UCI Repository and the breast cancer dataset from the UMMC are more accurate than the results of the DUFFNN clustering method. We discuss and evaluate the proposed methods with more details in Section 8.3 in the next chapter.

CHAPTER 8: CONCLUSIONS AND FUTURE RESEARCH

8.1 Introduction

Looking back, this thesis has presented eight chapters, including this chapter that began from the re-evaluation and exploration of the current online dynamic unsupervised feedforward neural network (ODUFFNN) clustering methods to the idea of developing the dynamic unsupervised feedforward neural network (DUFFNN) by reducing the training process to one epoch. We discussed some related researches of the ODUFFNN clustering, reducing the training process, increasing clustering accuracy and reducing the time complexity and reducing memory complexity. This chapter briefly discussed the summary of the results and findings, achievement of the objectives and contributions of the current study, in order to overcome the problems mentioned in Section 1.2. Finally, we recommend some future researches in order to improve the results of the proposed methods, and to apply the method in different environments.

8.2 Achievement to Objectives of Research

The following are the objectives of this research:

- To review current effective ODUFFNN clustering methods.
- To identify limitations and problems of current effective ODUFFNN clustering methods through the literature and practical investigations.
- To develop a dynamic unsupervised feedforward neural network (DUFFNN) clustering method that is able to:

1) Reduce the training time of clustering during one training epoch
2) Increase the accuracy of clustering
3) Reduce the time complexity of clustering
4) Reduce the memory complexity of clustering

- To evaluate the performance of the proposed DUFFNN method.
- To compare the results of the proposed DUFFNN method performance with rival methods within the scope of this research.

In the next sections, we explain how we achieved the objectives of the research, in order to overcome the mentioned problems in Section 1.2: high training time and low accuracy of clustering, besides, high time complexity and high memory complexity of clustering.

8.3 Summary of the results and finding

The UFFNN clustering methods such as SOM and GNG are inherently distributed parallel processing architectures that can adjust their interconnection weights to learn (Andonie & Kovalerchuk, 2007; Bengio, et al., 2000; Hegland, 2003; Jain, 2010; Rougier & Boniface, 2011). In the online non-stationary data environment such as credit card transactions, the data are often highly massive and continuous, the distribution of data is not known and the data distribution may change over time. The problem of such environments are collection, storage, search, transfer, visualization and analysis of massive noise and the number of data dimensions (Bouchachia, et al., 2007; Hebboul, et al., 2011; Hsu, 2003; Kasabov, 1998; Rougier & Boniface, 2011). In the real online area, the static UFFNN are not suitable to use, however, they are generally considered as the fundamental clustering methods and are adapted/modified to be used in non-stationary environments, and forms the current online dynamic UFFNN (ODUFFNN) clustering methods such as the ESOM and DSOM (Bouchachia, et al., 2007; Hebboul, et al., 2011; Kasabov, 1998; Schaal & Atkeson, 1998). However, current ODUFFNN methods inherit the limitations and problems of the primary UFFNN clustering methods too (Bouchachia, et al., 2007; Hegland, 2003; Kasabov, 1998; Schaal & Atkeson, 1998). We discussed the strategies and topologies of the current effective ODUFFNN methods, and investigated and analysed their limitations, problems and some of the reasons of these problems in Chapter 3, in order to answer the questions, "Q1: What are the strategies, topologies and performances of current affective ODUFFNN clustering methods?" and "Q2: What are the limitations and problems of the current ODUFFNN clustering methods?" in Section 1.4. The ODUFFNN clustering methods should not be rigid; and should be ready to change and update its structure, nodes and connections with each online data entry. Therefore, the ODUFFNN clustering method should handle new data, control noisy data, adapt its algorithm, accommodate and prune data and rules incrementally and adjust itself in a flexible style to new conditions of the environment over time dynamically for processing of both data and knowledge. Relearning is a critical issue in the ODUFFNN clustering method with lifelong and incremental learning. The ODUFFNN clustering methods should train online data fast without relearning and cluster the continuous data in one pass, because there is no capacity to store the whole details of the online data, previous data and the connection of the data points in consequent steps (Bouchachia, et al., 2007; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu, et al., 2013; Schaal & Atkeson, 1998). However, the ODUFFNN clustering methods are not able to accommodate and adjust the data and rules without destroying old data and old knowledge (Bouchachia, et al., 2007; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). In addition, the ODUFFNN method must control time, memory space and accuracy efficiently (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). As mentioned in Section 1.2, current ODUFFNN clustering methods generally suffer from high training time and low accuracy of clustering, also high time complexity and high memory complexity of clustering (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). Essentially, we recognize the reasons of the problems of the current ODUFFNN clustering methods are related to the structure and features of the data, such as the size and dimensions of data, growth of the number of clusters and size of the network during clustering; and the topology and algorithm of the current ODUFFNN clustering method, such as the use of random weights, distance thresholds and parameters for controlling tasks during clustering, and relearning that takes several epochs and CPU time usage. There is a trade-off between training time, clustering accuracy, time complexity and memory complexity, in the ODUFFNN clustering methods, and there is surprisingly and comparatively very few published works dealing with them together in one ODUFFNN clustering model (Hamker, 2001; Hebooul, et al., 2015; Kasabov, 1998; Kulkarni & Mulay, 2013; Liu & Ban, 2015; Liu, et al., 2013). For example, the IGNGU focuses on the strategy of fast training by pruning, however, its action leads to low accuracy of clustering (Hebboul, et al., 2011).

To the best of our knowledge, there are no published works related to the evaluation of current ODUFFNN clustering methods on a standard common datasets for benchmark purposes. In this research, we considered several datasets from the UCI Repository, and implemented, evaluated and compared the results of the ESOM and DSOM as effective examples of the ODUFFNN methods. In addition, this research provides a good range of experimental results of these ODUFFNN methods and the proposed methods, and could potentially act as benchmarks for other researchers working in this field. As shown in Table 8-1, we compare the DUFFNN clustering method with some effective current ODUFFNN clustering methods, based on the experimental results and evaluation in Chapters 6 and 7, in order to answer question, "Q5: How is the performance of the developed DUFFNN clustering method in comparison of results with rival methods?", in Section 1.4.

Table 8-1: Comparison of the DUFFNN clustering method with some current online dynamic unsupervised feedforward neural network clustering methods

	ESOM	ESOINN	DSOM	IGNGU	EDSOM	HI-GNG	DUFFNN
Base patterns	SOM and GNG, Hebbian	GNG	SOM	GNG and Hebbian	SOM and GNG	GNG and Hebbian	RUFFNN
Some bold features (Advantages)	Begin without any node	Control the number and density of each cluster	Improve the formula of updating weights	Train by two layers in parallel	Begin with four connected nodes	Begin without any node	Begin without any node
							Input vectors are not stored during learning
	Update itself with online input data	Initialize code book	Elasticity or Flexibility property	Control density of each cluster and size of the network	Initialize code book [m×4]	Control density of each cluster and size of the network	Update itself with online input data
	The nodes with weak thresholds can be pruned	Prune for controlling noise and weak thresholds		Control noise	The nodes with weak threshold can be pruned	Prune nodes with weak thresholds	The nodes with weak thresholds can be pruned
		Input vectors are not stored during learning		Fast training by pruning			Initialize non-random weights
							No sensitive to the order of the data entry
		New input does not destroy last learned knowledge					Mining *BMW*
							Cluster during one epoch without updating weights
	Cluster during one epoch		Earn *best matching Unit (BMU)*		Earn *best matching unit (BMU)*	Use the enhanced Hebbian learning which makes the method robust to noisy data nodes	Ability to retrieve old data
							Ability to learn the number of clusters
Time Complexity	$O(n^2.m)$	$O(c.n^2.m)$	$O(c.n.m^2)$	$O(c.n^2.m)$	$O(c.n^2.m)$	$O(c.n^2.m)$	$O(n.m)$
Memory Complexity	$O(n^2.m.s_m)$	$O(c.n^2.m.s_m)$	$O(c.n.m^2.s_m)$	$O(c.n^2.m.s_m)$	$O(c.n^2.m.s_m)$	$O(c.n^2.m.s_m)$	$O(n.m.s_m)$

The DUFFNN clustering with incremental lifelong or online learning property is developed for real non stationary environments, based on the structure, features and capabilities of the RUFFNN clustering in order to overcome the mentioned problems in Section 1.2. The DUFFNN clustering is a flexible method and with each online continuous data, immediately updates all nodes, weights and distance thresholds. The proposed DUFFNN method is able to learn the number of clusters, without having any constraint and parameter for controlling the clustering tasks, based on the total thresholds, and generates the clusters after just one epoch of training. As mentioned earlier, the DUFFNN is a flexible model and by changing the *BMW*, immediately re-clusters the current online data node and old nodes dynamically, and subsequently clusters all data nodes based on the new structure of the network without suffering or destroying old data. The DUFFNN clustering method is able to control or delete attributes with weak weights to reduce the data dimensions, and data with solitary thresholds in order to reduce noise. Moreover, the DUFFNN method has the capabilities of increasing clustering accuracy and improving training time in a single epoch clustering without weight updating, and improving time complexity and memory complexity of clustering.

8.4 Contributions of the research

Based on the objectives in Section 1.3, we have considered several works and strategies to overcome the problems and achieve the goals of this research. Originally, the contributions of the research are outlined as follows:

- A developed real unsupervised feedforward neural network (RUFFNN) clustering method
- An improved real semi-supervised feedforward neural network (RSFFNN) clustering method
- A developed dynamic unsupervised feedforward neural network (DUFFNN) clustering method
- An improved dynamic semi-supervised feedforward neural network (DSFFNN) clustering method

A Developed Real Unsupervised Feedforward Neural Network (RUFFNN) Clustering Method: We developed the RUFFNN method to cluster stationary data. The goals of developing the RUFFNN clustering are high accuracy, low training time, low time complexity and low memory complexity. In order to achieve these goals, the RUFFNN clustering method has several strategies. First, the method considers uniform normalized data through data preprocessing. Then, the method generates non random weights by using input data directly. The method prunes attributes with weak weights and reduce the number of dimensions of the data matrix. Subsequently, the thresholds of each record of the data matrix are computed by considering non random weights and normalized data values. The RUFFNN recognizes noisy data by considering solitary thresholds and deleting them. The RUFFNN method clusters the data of a matrix based on the exclusion threshold of each record or data instance. Therefore, the number of clusters and capacity of each cluster is predicted based on the thresholds by the model without any pre-initialization of parameters.

The RUFFNN clusters the matrix of the data during one training epoch, with low CPU time usage, without updating the weights and computing error function. Table 5-4 in Section 5.6 showed, the time complexity and memory complexity of the RUFFNN are $O(n.m)$ and $O(n.m.s_m)$ respectively, which are lower than related methods.

An Improved Real Semi-supervised Feedforward Neural Network (RSFFNN) Clustering Method: We improved the result of the RUFFNN clustering method by assigning a class label to each input instance based on the training set. By using the *K-step* activation function, the model considers the exclusive threshold of each input instance and the related class label. Consequently, based on *K* class labels and exclusive thresholds in the training set, the model expects *K* clusters and for each cluster considers a domain of thresholds. By considering the clusters of results of the RUFFNN method, if there is some input instance with a related threshold in each cluster but without related class label, the model moves this input instance to a related cluster. Therefore, the model updates the number of clusters and the density of each cluster by using class labels. This stage improves the accuracy of clustering.

A Developed Dynamic Unsupervised Feedforward Neural Network (DUFFNN) Clustering Method: The main contribution of the thesis is the DUFFNN clustering method which has incremental lifelong learning suitable for dynamic environments. The goals of developing the DUFFNN clustering are high accuracy, low training time, low time complexity and low memory complexity in the scope of ODUFFNN clustering methods. In order to achieve these goals, the DUFFNN clustering method has several strategies. Dynamically after the entrance of each online input data, the method pre-processes the data by considering uniform normalized online data. Then, the method generates non random weights by using online input data directly. The DUFFNN computes and stores *essential Intelligent Information (EII)* of the current data, such as, the best matching weight (*BMW*) vector. The method is able to control and delete attributes with weak weights to reduce the data dimensions. Consequently, a single layer DUFFNN fetches the *BMW* vector and normalized the current data to generate its exclusive threshold during one epoch of training. If the new *BMW* vector is not equal to the old *BMW*, the single layer DUFFNN retrieves the old data nodes by using the *EII* and updates their *TT* based on the new *BMW.* The model recognizes and manages the data with solitary thresholds in order to reduce noise. Finally, the number of clusters and density of each cluster is updated. Similar to the RUFFNN clustering method, the DUFFNN method predicts the number of clusters and capacity of each cluster based on the thresholds without any pre-initialization of parameters. The DUFFNN clusters the online data in one training epoch, thus reduces CPU time usage, without updating the weights and computing error function. Also, the DUFFNN method is not sensitive to the order of the entrance of the online data. The time complexity and memory complexity of the DUFFNN are *O(n.m)* and *O(n.m.s$_m$)* respectively, which are lower than the mentioned related methods in the scope of this research, as Table 5-4 showed these matters in Section 5.6 . Therefore, the DUFFNN clustering method is able to achieve the objectives of the research appropriately, in answering the questions, "Q3: Is the proposed DUFFNN clustering method appropriate in order to cluster online continuous data dynamically and incrementally?" and "Q4: How can we develop a DUFFNN clustering method and what is the associated algorithm that is able to reduce the training time of clustering, increase the accuracy of clustering, reduce the time complexity and reduce memory complexity of clustering?", in Section 1.4.

An Improved Dynamic Semi-supervised Feedforward Neural Network (DSFFNN) Clustering Method: The DUFFNN clustering is improved by applying class labels as partial supervision, which is called dynamic semi-supervised FFNN (DSFFNN) clustering, to assign a class label to each unlabeled data by considering a linear activation function and the exclusion threshold for more accurate results. The DSFFNN model like the RSFFNN, updates the number of clusters and the density of each cluster by using class labels through the feedback of users, which done dynamically. This stage affects the result of clustering and improves the accuracy of clustering of DUFFNN method.

In Chapter 7, we evaluated and compared the performance of the proposed DUFFNN method with other related methods within the scope of this research, we used various datasets from the university of California at Irvine machine learning repository. Furthermore, we also used the breast cancer dataset from the University of Malaya Medical Centre (UMMC) to predict the survival time by using the DSFFNN clustering method. For experimental purposes, the training time is measured by the number of epochs, the CPU time usage, and time complexity. The accuracy of the clustering methods are measured by employing the *F*-measure with 10 folds of the test set, and also through the number of clusters and the density of the correctly classified nodes (*CCN*) (Andrew, 2014; Chaimontree, et al., 2010; Rendón, et al., 2011; Rendón, et al., 2011; Sung & Mukkamala, 2003). The memory usage is estimated by memory complexity which is computed by using the number of input data, training iterations and clusters; and the densities of the clusters. As part of the experimental results, the accuracy of the DUFFNN clustering method by using an *F - measure* with 10-fold cross-validation was 97.71% of the Breast Cancer, 97.24% of Iris, 73.41% of Spam, 90.52% of SPECT Heart, 86.62% of SPECTF Heart, 52.57% of Musk1, 84.31% of Musk2, 66.07% of Arcene, and 27.25% of Yeast datasets , respectively, in just one epoch of training in a relatively short time, and so the results were improved around 100% by using the DSFFNN clustering method. Furthermore, the accuracy of clustering the breast cancer dataset from the university of Malaya medical centre to predict the survival time, were between 98.14% and 100% by the DSFFNN method showing superior results. Based on question in Section 1.4, "Q5: How is the performance of the developed DUFFNN clustering method in comparison of results with rival methods?", the experimental results, prove that the method is a practical proficiency with the strong support of theoretical background.

8.5 Recommendation and Future Directions

In this research, the results of Chapter 7 presented the robust and flexible properties of the developed DUFFNN clustering method. The following are recommendations for future studies, suggested based on several prospects for investigating the research field:

- To develop the DUFFNN clustering method in different data mining systems such as distributed stream data mining, time series data mining, world wide web mining, graph mining and multi-rational data mining.
- To apply the DUFFNN clustering method in order to cluster higher dimensional data with the higher number of classes in the big data environment.
- To implement the hardwares of the RUFFNN, RSFFNN, DUFFNN, and DSFFNN clustering methods, in order to apply in different technological environments.

As mentioned earlier, there are no published works related to the evaluation of current ODUFFNN clustering methods on a standard common datasets for benchmark purposes. Also, the selected datasets are remarkable, and most conventional methods do not satisfactorily cluster these datasets because of their features. In this research, we considered nine different datasets from the UCI Repository, and implemented, evaluated and compared several effective current ODUFFNN clustering methods. In addition, this research provides a good range of experimental results of these ODUFFNN methods and the proposed methods. We hope this research could potentially act as benchmarks for other researchers working in this field.

REFERENCES

Ahirwar, G. (2014). A Novel K means Clustering Algorithm for Large Datasets Based on Divide and Conquer Technique. Pradnyesh. J. Bhisikar(IJCSIT) International Journal of Computer Science and Information Technologies, 5(1), 301-305.

Aleksander, I., & Morton, H. (1990). An introduction to neural computing (Vol. 240): Chapman and Hall London.

Amigó, E., Gonzalo, J., Artiles, J., & Verdejo, F. (2009). A comparison of extrinsic clustering evaluation metrics based on formal constraints. Information retrieval, 12(4), 461-486.

Andonie, R., & Kovalerchuk, B. (2007). Neural Networks for Data Mining: Constrains and Open Problems. Ellensburg, WA: Central Washington University, Computer Science Department.

Andrew, P. (2014, 20 April 2014). Semantic Search Art, from http://semanticsearchart.com/downloadsdatacorpus.html

Andrews, N. O., & Fox, E. A. (2007). Recent developments in document clustering. Computer Science, Virginia Tech, Tech. Rep.

Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. Paper presented at the ACM Sigmod Record.

Asadi, R., Mustapha, N., & Sulaiman, N. (2009). Training Process Reduction Based On Potential Weights Linear Analysis To Accelarate Back Propagation Network. Arxiv preprint arXiv:0908.1453.

Asuncion, A., & Newman, D. (2007). UCI Machine Learning Repository. Irvine,CA: University of California, School of Information andComputer Science. Accessed at http://www.ics.uci.edu/~mlearn/MLRepository.

Barnett, V. (1994). Lewis, 1, 1994, Outliers in Statistical Data: Chichester, John Wiley.

Bengio, Y. (2013). Deep Learning of Representations: Looking Forward. arXiv preprint arXiv:1305.0445.

Bengio, Y., Buhmann, J. M., Embrechts, M., & Zurada, J. M. (2000). Neural networks for data mining and knowledge discovery [Special Issue]. (Vol. 100): IEEE Transactions on Neural Networks.

Bose, N. K., & Liang, P. (1996). Neural network fundamentals with Graphs, Algorithms, and Applications: New York: McGraw–Hill.

Bouchachia, A., Gabrys, B., & Sahel, Z. (2007). Overview of some incremental learning algorithms. Paper presented at the Proc. Fuzzy Systems Conf. Fuzz-IEEE.

Brachman, R. J., & Anand, T. (1994). The Process of Knowledge Discovery in Databases: A First Sketch. Paper presented at the KDD Workshop.

Brown, M., An, P. E., Harris, C. J., & Wang, H. (1993). How Biased is Your Multi-Layered Perceptron?

Camastra, F., & Verri, A. (2005). A novel kernel method for clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence,, 27(5), 801-805.

Chaimontree, S., Atkinson, K., & Coenen, F. (2010). Multi-agent based clustering: Towards generic multi-agent data mining Advances in Data Mining. Applications and Theoretical Aspects (pp. 115-127): Springer.

Chakraborty, R. C. (2010). Fundamentals of Neural Networks. Soft Computing, 7-14.

Chattopadhyay, M., Pranab, K., & Mazumdar, S. (2011). Principal component analysis and self-organizing map for visual clustering of machine-part cell formation in cellular manufacturing system. Paper presented at the Systems Research Forum.

Christen, P. (2005). Probabilistic data generation for deduplication and data linkage Intelligent Data Engineering and Automated Learning-IDEAL 2005 (pp. 109-116): Springer.

Costa, J. A. F., & Oliveira, R. S. (2007). Cluster analysis using growing neural gas and graph partitioning. Paper presented at the Neural Networks, 2007. IJCNN 2007. International Joint Conference on.

Craven, M. W., & Shavlik, J. W. (1997). Using neural networks for data mining. Future generation computer systems, 13(2), 211-229.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS), 2(4), 303-314.

Daffertshofer, A., Lamoth, C. J. C., Meijer, O. G., & Beek, P. J. (2004). PCA in studying coordination and variability: a tutorial. Clinical Biomechanics, 19(4), 415-428.

Dasarathy, B. V. (1990). Nearest neighbor pattern classification techniques. Los Alamitos, CA: IEEE Computer Society Press.

Deconinck, S. (2010). Artificial intelligence a modern approach.

DeMers, D., & Cottrell, G. (1993). Non-linear dimensionality reduction. Advances in neural information processing systems, 580-580.

Demuth, H., Beale, M., & Hagan, M. (2008). Neural Network Toolbox TM 6: User's Guide. Natick, MA: Math Works.

Deng, D., & Kasabov, N. (2003). On-line pattern analysis by evolving self-organizing maps. Neurocomputing, 51, 87-103.

Doszkocs, T. E., Reggia, J., & Lin, X. (1990). Connectionist models and information retrieval. Annual review of information science and technology, 25, 209-262.

Drago, G. P., & Ridella, S. (1992). Statistically controlled activation weight initialization (SCAWI). Neural Networks, IEEE Transactions on, 3(4), 627-631.

Du, K. L. (2010). Clustering: A neural network approach. Neural Networks, 23(1), 89-107.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? The Journal of Machine Learning Research, 11, 625-660.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. Paper presented at the Kdd.

Ester, M., Kriegel, H. P., & Xu, X. (1995). A database interface for clustering in large spatial databases. Paper presented at the KDD.

Estévez, P. A., Tesmer, M., Perez, C. A., & Zurada, J. M. (2009). Normalized mutual information feature selection. Neural Networks, IEEE Transactions on, 20(2), 189-201.

Fern, X. Z., & Brodley, C. E. (2004). Solving cluster ensemble problems by bipartite graph partitioning. Paper presented at the Proceedings of the twenty-first international conference on Machine learning.

Fernandez-Redondo, M., & Hernandez-Espinosa, C. (2000). A comparison among weight initialization methods for multilayer feedforward networks. Paper presented at the Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on.

Fernández-Redondo, M., & Hernandez-Espinosa, C. (2001). Weight initialization methods for multilayer feedforward. Paper presented at the ESANN'2001 proceedings-European Symposium on Artificial Neural Networks.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. Machine Learning, 2(2), 139-172.

Fisher, R. (1950). The Use of Multiple Measurements in Taxonomic Problems: Contributions to Mathematical Statistics (Vol. (Original work published 1936)): JNew York: Wiley.

Fowlkes, E. B., & Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. Journal of the American statistical association, 78(383), 553-569.

Fritzke, B. (1995). A growing neural gas network learns topologies.

Fritzke, B. (1997). Some competitive learning methods. Dresden: Dresden University of Technology, Artificial Intelligence Institute.

Funahashi, K. I. (1989). On the approximate realization of continuous mappings by neural networks. Neural Networks, 2(3), 183-192.

Furao, S., & Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. Neural Networks, 19(1), 90-106.

Furao, S., Ogura, T., & Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Networks, 20(8), 893-903.

Gabrys, B. (2012). Adaptive Preprocessing for Streaming Data.

Galhardas, H., Florescu, D., Shasha, D., Simon, E., & Saita, C. (2001). Declarative data cleaning: Language, model, and algorithms.

Ganesh, M., Srivastava, J., & Richardson, T. (1996). Mining entity-identification rules for database integration. Paper presented at the Proceedings of the Second International Conference on Data Mining and Knowledge Discovery.

Germano, T. (1999). Self Organizing Maps. Accessed at http://davis.wpi.edu/~matt/courses/soms.

Goebel, M., & Gruenwald, L. (1999). A survey of data mining and knowledge discovery software tools. ACM SIGKDD Explorations Newsletter, 1(1), 20-33.

Goroshin, R., & LeCun, Y. (2013). Saturating Auto-Encoder. arXiv preprint arXiv:1301.3577.

Granda, W. V. (2003). Strategies for Clustering, Classifying, Integrating, Standardizing and Visualizing Microarray Gene Expression Data A Beginner’s Guide to Microarrays (pp. 277-340): Springer.

Gui, V., Vasiu, R., & Bojković, Z. (2001). A new operator for image enhancement. Facta universitatis-series: Electronics and Energetics, 14(1), 109-117.

Guyon, I. (2003). Design of experiments of the NIPS 2003 variable selection benchmark. Paper presented at the NIPS 2003 workshop on feature extraction and feature selection.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. The Journal of Machine Learning Research, 3, 1157-1182.

Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2001). On clustering validation techniques. Journal of Intelligent Information Systems, 17(2-3), 107-145.

Hamker, F. H. (2001). Life-long learning Cell Structures--continuously learning without catastrophic interference. Neural Networks, 14(4-5), 551-573.

Han, J., & Kamber, M. (2006). Data Mining, Southeast Asia Edition: Concepts and Techniques. San Francisco, CA.: Morgan kaufmann.

Han, J., & Kamber, M. (2011). Pei. Data Mining Concepts and Techniques: The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann Publishers.

Haykin, S. (1994). Neural networks: a comprehensive foundation: Prentice Hall PTR.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation.

Haykin, S., & Network, N. (2004). A comprehensive foundation. Neural Networks, 2.

Hazlina, H., Sameem, A., Nur Aishah, M., & Yip, C. (2004). Back Propagation Neural Network for the Pronosis of Breast Cancer: Comparison on Different Training Algorithms. Proceedings of the Second International Coriference on Artificial Intelligence in Engineering & Technology, Sabah, Malaysia, 445-449.

Hebb, D. O. (1949). The organization of behavior: A neuropsychological approach. New York: Wiley, 1, 143-150.

Hebboul, A., Hacini, M., & Hachouf, F. (2011). An incremental parallel neural network for unsupervised classification. Paper presented at the Proc. 7th Int. Workshop on Systems, Signal Processing Systems and Their Applications (WOSSPA).

Hebooul, A., Hachouf, F., & Boulemnadjel, A. (2015). A new Incremental Neural Network for Simultaneous Clustering and Classification. Neurocomputing.

Hegland, M. (2003). Data mining–challenges, models, methods and algorithms. Canberra, Australia: Australia National University, ANU Data Mining Group.

Herrmann, L., & Ultsch, A. (2007). Label propagation for semi-supervised learning in self-organizing maps. Proceedings of the 6th WSOM.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504.

Honkela, T. (1998). Description of Kohonen's Self-Organizing Map. Accessed at http://www.cis.hut.fi/~tho/thesis.

Hsu, J. (2003). Data Mining and Business Intelligence: Tools, Technologies. Business Intelligence in the Digital Economy: Opportunities, Limitations and Risks: Opportunities, Limitations and Risks, 141.

Hubert, L., & Arabie, P. (1985). Comparing partitions. Journal of classification, 2(1), 193-218.

Jacquier, E., Kane, A., & Marcus, A. J. (2003). Geometric or arithmetic mean: A reconsideration. Financial Analysts Journal, 46-53.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. Pattern Recognition Letters, 31(8), 651-666.

Jain, A. K., & Dubes, R. C. (1988). Algorithms for clustering data (Vol. 6): Prentice hall Englewood Cliffs.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. ACM computing surveys (CSUR), 31(3), 264-323.

Jean, J. S., & Wang, J. (1994). Weight smoothing to improve network generalization. IEEE Transactions on Neural Networks, 5(5), 752-763.

Jiang, Y., & Ren, J. (2011). Eigenvalue sensitive feature selection. Paper presented at the Proceedings of the 28th International Conference on Machine Learning (ICML-11).

Jolliffe, I. T. (1986). Principal Component Analysis Springer Series in Statistics (pp. 1-7): Springer New York.

Jolliffe, I. T. (2002). Principal Component Analysis: New York: Springer–Verlag.

Kamiya, Y., Ishii, T., Furao, S., & Hasegawa, O. (2007). An online semi-supervised clustering algorithm based on a self-organizing incremental neural network. Paper presented at the Proc. Int. Joint Conf. Neural Networks (IJCNN), IEEE.

Kamyshanska, H., & Memisevic, R. (2013). On autoencoder scoring.

Kantardzic, M. (2011). Data mining: concepts, models, methods, and algorithms: New York: Wiley–Interscience.

Kasabov, N. K. (1998). ECOS: Evolving Connectionist Systems and the ECO Learning Paradigm. Paper presented at the Proc. 5th Int. Conf. Neural Information Processing, ICONIP'98, Kitakyushu, Japan.

Kaski, S. (2009). Self-organizing maps.

Kaufman, L., & Rousseeuw, P. (1987). Clustering by means of medoids: North-Holland.

Kaufmann, L., & Rousseeuw, P. J. (1990). Finding groups in data. New York: J. Wiley & Sons.

Keeni, K., Nakayama, K., & Shimodaira, H. (1999). A training scheme for pattern classification using multi-layer feed-forward neural networks. Paper presented at the Computational Intelligence and Multimedia Applications, 1999. ICCIMA'99. Proceedings. Third International Conference on.

Kemp, R. A., MacAulay, C., Garner, D., & Palcic, B. (1997). Detection of malignancy associated changes in cervical cell nuclei using feed-forward neural networks. Journal of the European Society for Analytical Cellular Pathology, 14(1), 31-40.

Kim, Y. K., & Ra, J. B. (1991). Weight value initialization for improving training speed in the backpropagation network. Paper presented at the Neural Networks, 1991. 1991 IEEE International Joint Conference on.

Knorr, E. M., Ng, R. T., & Tucakov, V. (2000). Distance-based outliers: algorithms and applications. The VLDB Journal, 8(3-4), 237-253.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological cybernetics, 43(1), 59-69.

Kohonen, T. (1997). Self-organizing maps (Springer Series in Information Sciences). Berlin: Springer–Verlag, 30, 22-25.

Kohonen, T. (2000). Self-Organization Maps (3rd ed.): Berlin: Springer–Verlag.

Kulkarni, P. A., & Mulay, P. (2013). Evolve systems using incremental clustering approach. Evolving Systems, 4(2), 71-85.

Kuncheva, L. I. (2005). Using diversity measures for generating error-correcting output codes in classifier ensembles. Pattern Recognition Letters, 26(1), 83-90.

Kurgan, L. A., Cios, K. J., Tadeusiewicz, R., Ogiela, M., & Goodenday, L. S. (2001). Knowledge discovery approach to automated cardiac SPECT diagnosis. Artificial Intelligence in Medicine, 23(2), 149-169.

L., K., & T., D. (2006). Hebbian Learning, Principal Component Analysis, and Independent Component Analysis. 15-486/782: Artificial Neural Networks, from http://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/slides/hebbpca.pdf

Labib, K., & Vemuri, V. R. (2006). An application of principal component analysis to the detection and visualization of computer network attacks. Paper presented at the Annales des télécommunications.

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. Journal of Machine Learning Research, 10, 1-40.

Larochelle, H., Mandel, M., Pascanu, R., & Bengio, Y. (2012). Learning Algorithms for the Classification Restricted Boltzmann Machine. Journal of Machine Learning Research, 13, 643-669.

Larson, J. W., Hegland, M., Harding, B., Roberts, S., Stals, L., Rendell, A. P., . . . Nobes, R. (2013). Fault-tolerant grid-based solvers: Combining concepts from sparse grids and mapreduce. Procedia Computer Science, 18, 130-139.

LeCun, Y., Bottou, L., Orr, G., & Müller, K. (1998). Efficient backprop. Neural networks: Tricks of the trade, 546-546.

Li, G., Alnuweiri, H., Wu, Y., & Li, H. (1993). Acceleration of back propagation through initial weight pre-training with delta rule. Paper presented at the Neural Networks, 1993., IEEE International Conference on.

Lin, L., Osan, R., & Tsien, J. Z. (2006). Organizing principles of real-time memory encoding: neural clique assemblies and universal neural codes. TRENDS in Neurosciences, 29(1), 48-57.

Linde, Y., Buzo, A., & Gray, R. (1980). An algorithm for vector quantizer design. IEEE Transactions on Communications, 28(1), 84-95.

Lindsay, R. S., Funahashi, T., Hanson, R. L., Matsuzawa, Y., Tanaka, S., Tataranni, P. A., . . . Krakoff, J. (2002). Adiponectin and development of type 2 diabetes in the Pima Indian population. Lancet, 360(9326), 57-58.

Lippmann, R. P. (1987). An introduction to computing with neural nets. ASSP Magazine, IEEE, 4(2), 4-22.

Liu, H., & Ban, X. J. (2015). Clustering by growing incremental self-organizing neural network. Expert Systems with Applications, 42(11), 4965-4981.

Liu, H., Kurihara, M., Oyama, S., & Sato, H. (2013). An incremental self-organizing neural network based on enhanced competitive Hebbian learning. Paper presented at the Neural Networks (IJCNN), The 2013 International Joint Conference on.

Longadge, R., & Dongre, S. (2013). Class Imbalance Problem in Data Mining Review. arXiv preprint arXiv:1305.1707.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. Paper presented at the Proceedings of the fifth Berkeley symposium on mathematical statistics and probability.

Maiorana, F., Mastorakis, N. E., Poulos, M., Mladenov, V., Bojkovic, Z., Simian, D., . . . Udriste, C. (2008). Performance improvements of a Kohonen self organizing classification algorithm on sparse data sets. Paper presented at the WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering.

Maletic, J. I., & Marcus, A. J. (2000). Data cleansing: Beyond integrity analysis. Paper presented at the Proceedings of the Conference on Information Quality.

Mangat, V., & Vig, R. (2014). Novel associative classifier based on dynamic adaptive PSO: Application to determining candidates for thoracic surgery. Expert Systems with Applications, 41(18), 8234-8244.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval (Vol. 1): Cambridge university press Cambridge.

Marcus, A. J., Maletic, J. I., & Lin, K. I. (2001). Ordinal association rules for error identification in data sets. Paper presented at the Proceedings of the tenth international conference on Information and knowledge management.

Marina, M., D., H. . (2001). An Experimental Comparison of Model-Based Clustering Methods. . Machine Learning 42(1/2), 9-29.

Martinetz, T. M. (1993). Competitive Hebbian learning rule forms perfectly topology preserving maps ICANN'93 (pp. 427-434): Springer.

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). Neural-gas' network for vector quantization and its application to time-series prediction. IEEE Transactions on Neural Networks, 4(4), 558-569.

McClelland, J. L., Thomas, A. G., McCandliss, B. D., & Fiez, J. A. (1999). Understanding failures of learning: Hebbian learning, competition for representational space, and some preliminary experimental data. Progress in brain research, 121, 75-80.

McCloskey, S. (2000). Neural Networks and Machine Learning Accessed at http://www.cim.mcgill.ca/~scott/RIT/research_project.html (pp. 755).

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biology, 5(4), 115-133.

Mercer, D. P. (2003). Clustering large datasets. Linacre College.

Milligan, G. W., & Cooper, M. C. (1986). A study of the comparability of external criteria for hierarchical cluster analysis. Multivariate Behavioral Research, 21(4), 441-458.

Minsky, M., & Seymour, P. (1969). Perceptrons.

Mitchell, T. M. (1997). Machine learning. 1997. Burr Ridge, IL: McGraw Hill, 45.

Murtagh, F. (1983). A survey of recent advances in hierarchical clustering algorithms. The Computer Journal, 26(4), 354-359.

Niknam, T., Firouzi, B. B., & Nayeripour, M. (2008). An efficient hybrid evolutionary algorithm for cluster analysis. Paper presented at the World Applied Sciences Journal.

Nilsson, N. J. (1998). Artificial Intelligence: A New Synthesis: A New Synthesis: Elsevier.

Oh, M., & Park, H. M. (2011). Preprocessing of independent vector analysis using feed-forward network for robust speech recognition. Paper presented at the Proc. Neural Information Processing Conf.

Özbay, Y., Ceylan, R., & Karlik, B. (2006). A fuzzy clustering neural network architecture for classification of ECG arrhythmias. Computers in Biology and Medicine, 36(4), 376-388.

Papadimitriou, C. H. (2003). Computational complexity: John Wiley and Sons Ltd.

Pavel, B. (2002). Survey of clustering data mining techniques. San Jose, CA: Accrue Software.

Peng, J. M., & Lin, Z. (1999). A non-interior continuation method for generalized linear complementarity problems. Mathematical Programming, 86(3), 533-563.

Prudent, Y., & Ennaji, A. (2000). Extraction incrémentale de la topologie des données. Laboratoire PSI, Université et INSA de Rouen, F-76821, France.

Prudent, Y., & Ennaji, A. (2005). An incremental growing neural gas learns topologies. Paper presented at the Proc. IEEE Int. Joint Conf. Neural Networks, IJCNN'05.

Rendón, E., Abundez, I., Arizmendi, A., & Quiroz, E. M. (2011). Internal versus External cluster validation indexes. International Journal of computers and communications, 5(1), 27-34.

Rendón, E., Abundez, I., Gutierrez, C., & DÍAZ, S. (2011). A comparison of internal and external cluster validation indexes. Paper presented at the Proceedings of the 2011 American conference.

Rohlf, F. J. (1974). Methods of comparing classifications. Annual Review of Ecology and Systematics, 101-113.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Rougier, N., & Boniface, Y. (2011). Dynamic self-organising map. Neurocomputing, 74(11), 1840-1847.

Sárközy, G. N., Song, F., Szemerédi, E., & Trivedi, S. (2012). A Practical Regularity Partitioning Algorithm and its Applications in Clustering. arXiv preprint arXiv:1209.6540.

Satyanarayana, A., & Acquaviva, V. (2014). Enhanced cobweb clustering for identifying analog galaxies in astrophysics. Paper presented at the Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on.

Schaal, S., & Atkeson, C. G. (1998). Constructive incremental learning from only local information. Neural computation, 10(8), 2047-2084.

Shavlik, J. W., & Dietterich, T. G. (1990). Readings in machine learning: Morgan Kaufmann.

Sheikholeslami, G., Chatterjee, S., & Zhang, A. (1998). Wavecluster: A multi-resolution clustering approach for very large spatial databases. Paper presented at the VLDB.

Shen, F., Yu, H., Sakurai, K., & Hasegawa, O. (2011). An incremental online semi-supervised active learning algorithm based on self-organizing incremental neural network. Neural Computing and Applications, 20(7), 1061-1074.

Shimodaira, H. (1994). A weight value initialization method for improving learning performance of the backpropagation algorithm in neural networks. Paper presented at the Tools with Artificial Intelligence, 1994. Proceedings., Sixth International Conference on.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory.

Steinley, D. (2004). Properties of the Hubert-Arable Adjusted Rand Index. Psychological methods, 9(3), 386.

Studholme, C., Hill, D. L. G., & Hawkes, D. J. (1999). An overlap invariant entropy measure of 3D medical image alignment. Pattern Recognition, 32(1), 71-86.

Sung, A. H., & Mukkamala, S. (2003). Identifying important features for intrusion detection using support vector machines and neural networks. Paper presented at the Applications and the Internet, 2003. Proceedings. 2003 Symposium on.

Tong, X., Qi, L., Wu, F., & Zhou, H. (2010). A smoothing method for solving portfolio optimization with CVaR and applications in allocation of generation asset. Applied Mathematics and Computation, 216(6), 1723-1740.

Ultsch, A., & Siemon, H. P. (1990). Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis. Proc. Int. Neural Networks Conf., 305-308.

Van der Maaten, L. J., Postma, E. O., & Van den Herik, H. J. (2009). Dimensionality reduction: A comparative review. Journal of Machine Learning Research, 10(1-41), 66-71.

Van Rijsbergen, C. J. (1979). Information Retrieval. Dept. of Computer Science, University of Glasgow. URL: citeseer. ist. psu. edu/vanrijsbergen79information. html.

Vandesompele, J., De Preter, K., Pattyn, F., Poppe, B., Van Roy, N., De Paepe, A., & Speleman, F. (2002). Accurate normalization of real-time quantitative RT-PCR data by geometric averaging of multiple internal control genes. Genome biology, 3(7), research0034.

Wang, T., Yu, X., Alahakoon, D., & Fei, S. (2013). An enhancing dynamic self-organizing map for data clustering. Paper presented at the Control and Automation (ICCA), 2013 10th IEEE International Conference on.

Wang, W., Yang, J., & Muntz, R. (1997). STING: A statistical information grid approach to spatial data mining. Paper presented at the VLDB.

Werbos, P. (1974). Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD Thesis. Harvard University.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits.

Wolberg, W. H., & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Proceedings of the National Academy of Sciences, 87(23), 9193-9196.

Xianguang, G. (1998). Application of Improved Entropy Method in Evaluation of Economic Result [J]. Systems Engineering-Theory & Practice, 12.

Yoon, H.-S., Bae, C.-S., & Min, B.-W. (1995). Neural networks using modified initial connection strengths by the importance of feature elements. Paper presented at the Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on.

Zhang, X. M., Chen, Y. Q., Ansari, N., & Shi, Y. Q. (2004). Mini-max initialization for function approximation. Neurocomputing, 57, 389-409.

Zhao, Y., & Karypis, G. (2004). Empirical and theoretical comparisons of selected criterion functions for document clustering. Machine Learning, 55(3), 311-331.

Ziarko, W., & Shan, N. (1994). KDD-R: A comprehensive system for knowledge discovery in databases using rough sets. Paper presented at the Proc. 3rd Int. Workshop Rough Sets Soft Comput. RSSC'94.

Ziegel, E. R. (2002). Statistical inference. Technometrics, 44(4).

LIST OF PUBLICATIONS AND PAPERS PRESENTED

ISI journals

Asadi, R., Asadi, M., & Kareem, S. A. (2014). An efficient semisupervised feedforward neural network clustering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*, 1-15. The Cambridge University Press, University of Southern California, USA. (Q1)

Asadi, R., Kareem, S. A., Asadi, M., & Asadi, S. (2015). A Single-Layer Semi-Supervised Feed Forward Neural Network Clustering Method. *Malaysian Journal of Computer Science (MJCS)*, *28*, 3. The publication of University of Malaya, Malaysia. (Q4)

Asadi, R., Kareem, S. A., Asadi, M., & Asadi, S. (2015). A dynamic semisupervised feedforward neural network clustering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), (Accepted September 2015.* The Cambridge University Press, University of Southern California, USA. (Q1)

Asadi, R., Kareem, S. A., Asadi, M., & Asadi, S. (2015). An Unsupervised Feed Forward Neural Network Method for Efficient Clustering. The International Arab Journal of Information Technology (IAJIT), *(Accepted September 2015).* Zarqa University, Jordan. (Q2)

Scopus and General journals

Asadi, R., Mustapha, N., Sulaiman, N., & Shiri, N. (2009). New supervised multi layer feed forward neural network model to accelerate classification with high accuracy. *European Journal of Scientific Research*, *33*(1), 163-178.

Asadi, R., Mustapha, N., & Sulaiman, N. (2009). A framework for intelligent multi agent system based neural network classification model. *arXiv preprint arXiv:0910.2029*.

Asadi, R., Mustapha, N., & Sulaiman, N. (2009). Training process reduction based on potential weights linear analysis to accelarate back propagation network. *arXiv preprint arXiv:0908.1453*.

Asadi, R. (2009). *Preprocessing And Pretraining Of Multilayer Feed Forward Neural Network* (Doctoral dissertation, Universiti Putra Malaysia).

Asadi, R., H. Sabah Hasan, et al. (2014). "Review of current Online Dynamic Unsupervised Feed Forward Neural Network classification." International Journal of Artificial Intelligence and Neural Networks (IJAINN) 4(2): 12.

Conference Papers and Presentations

Asadi, R., Kareem, S. A., Hasan, S. H. (2014). Review of current Online Dynamic Unsupervised Feed Forward Neural Network classification. Computer Science and Electronics Engineering (CSEE) Kuala Lumpur, Malaysia.

Asadi, R., & Kareem, S. A. (2014, June). Review of feed forward neural network classification preprocessing techniques. In *PROCEEDINGS OF THE 3RD INTERNATIONAL CONFERENCE ON MATHEMATICAL SCIENCES* (Vol. 1602, pp. 567-573). AIP Publishing.

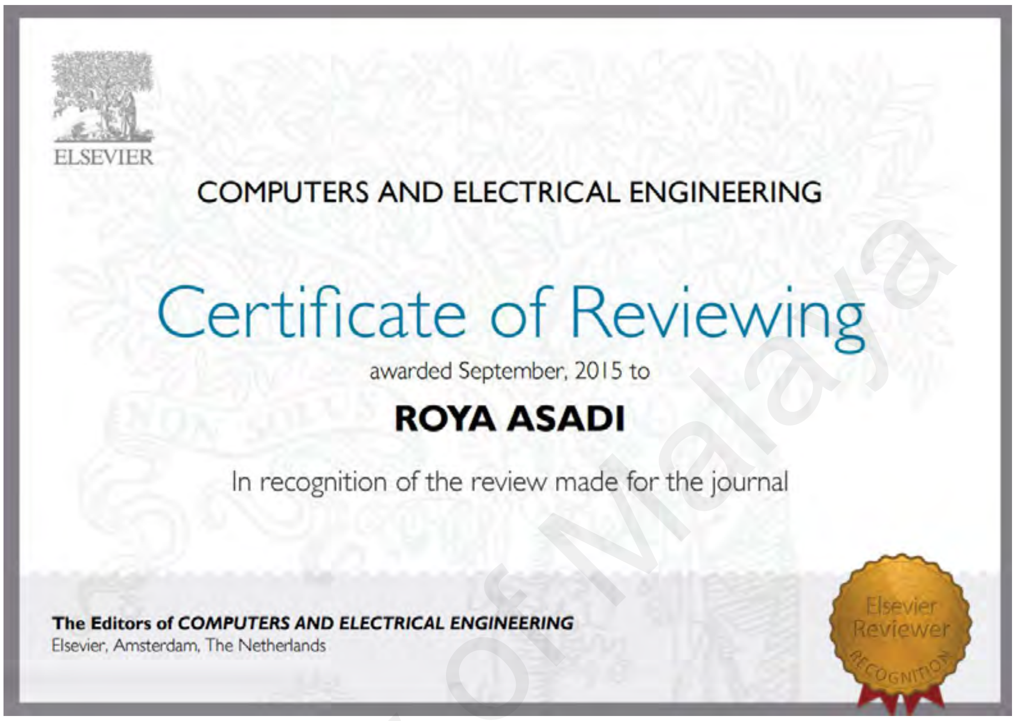
Asadi, R., Kareem, S. A., & Asadi, S. (2015). Assemble Intelligent Multi Agent System Based Feed-Forward Neural Network clustering.

Asadi, R., Abdul-Kareem, S. (2011). Preprocessing techniques in multi-layer feed forward neural network classification. PGRES-2011, Faculty of Computer Science and Information Technology, University of Malaya (UM) Conference, Kuala Lumpur, Malaysia.

Asadi,. R., Abdul-Kareem, S. (2013). A survey of feed forward neural network classification preprocessing techniques. PGRES-2013, Faculty of Computer Science and Information Technology, University of Malaya (UM) Conference, Kuala Lumpur, Malaysia.

Reviewer of some Journals and Conferences

Journal of Computers & Electrical Engineering- Elsevier (CAEE). Having the ELSEVIER award because of contribution and manuscript review in the last two years for *Computers and Electrical Engineering*.



ELSEVIER

COMPUTERS AND ELECTRICAL ENGINEERING

Certificate of Reviewing

awarded September, 2015 to

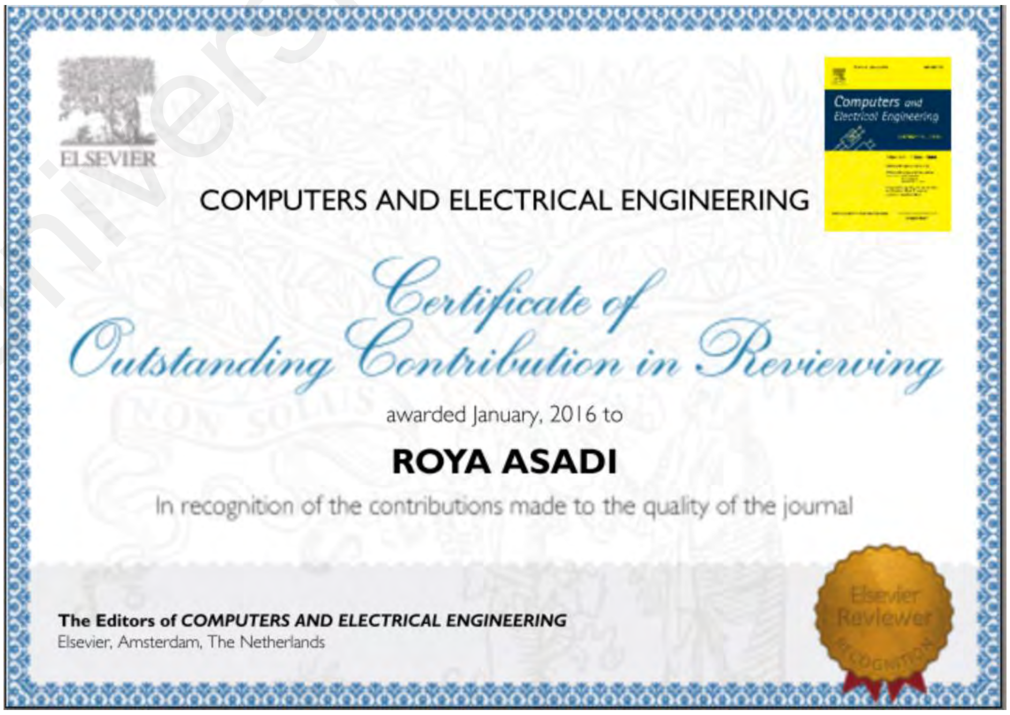
ROYA ASADI

In recognition of the review made for the journal

The Editors of *COMPUTERS AND ELECTRICAL ENGINEERING*
Elsevier, Amsterdam, The Netherlands

Elsevier Reviewer RECOGNITION

Having the ELSEVIER award because of contribution and manuscript review in the last three years for Computers and Electrical Engineering, as 10 percent Top Reviewers.



ELSEVIER

COMPUTERS AND ELECTRICAL ENGINEERING

Certificate of Outstanding Contribution in Reviewing

awarded January, 2016 to

ROYA ASADI

In recognition of the contributions made to the quality of the journal

The Editors of *COMPUTERS AND ELECTRICAL ENGINEERING*
Elsevier, Amsterdam, The Netherlands

Elsevier Reviewer RECOGNITION

Journal of IEEE Transactions on Neural Networks and Learning Systems (TNNLS).

The International Joint Conference on Neural Networks - (IJCNN-2014), Organized and Sponsored by INNS and IEEE-CIS.

The International Joint Conference on Neural Networks - (IJCNN-2016), Organized and Sponsored by INNS and IEEE-CIS.

Also, as invited reviewer of some other journals and conferences, such as *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM),* journal of Cambridge University Press.

APPENDIX A

Algorithms

```
PCA (X; L)
Input: Database X, database of input values;
Output: Matrix L, Transformed database of X;

Begin
Let row number: n;
Let column number: p;
Let Matrixcov: matrix n×p;
// Computing vertical mean and deviation in each column

Let $\mu_p$ = Mean of values vectors in column p;

Let $devn_p$ = deviation of values vectors from $\mu_p$ for each row n in column p;

// Computing the Cov of each two columns

Let $p' = p$;

    Forall  columns of Matrix $X_p$ do

        Forall  columns of Matrix $X_{p'}$  do

            Forall  rows of Matrix $X_n$  do

            $Matrixcov(p,p') = \sum (dev_{np} . dev_{np'}) / (n-1)$ ;

//computing the eigenvalues and eigenvectors by Convergence of Power Iteration,
$X_p=A.X_{p-i}$

Let $A_p=eigenvalue(Matrixcov(p,p'))$ ;
Let $Eg_p(p,p')=eigenvector(Matrixcov(p,p'))$ ;
// Choosing component : eigenvectors with highest eigenvalues
// Forming feature vectors(Sort eigen...)
Let $A_p$ =Sort descending ($A_p$) ;
//Deriving the new datasets: finaldata=rowfeaturevectore*rowdatabase

Forall  $p'$
    Forall  columns of Matrix $L_p$ do
        Forall  rows of Matrix $L_n$ do
        $L(n, p) = L(n, p) + D(n,p) * Eg_m(p,p')$ ;
```

Figure 3-1A: Principal Component Analysis (PCA) algorithm

```
SOM ( )
Input:  Data Vector $X$;
Output: Clusters of dataset ;

Begin
Let $t$ : Time;
Let $N^t$:Node set at time $t$;
Let $n$ : Current number of nodes in $N^t$;
Let $m$ : Current number of weight in $W_i^t$ for node $i$;
Let $W_i$ : For each node i, a vector weight initialed randomly;
Let $BMU$ : Best matching unit or winner node;
Let $p$ : Position of node;
Let $T$: Threshold;
Let $\varepsilon_i$ : The initial learning rate;
Let $\varepsilon_f$ : The final learning rate;
Let $\delta(t)$ : The initial neighbourhood width;
Let $\delta_f$ : The final neighbourhood width;
Let $\alpha(\delta)$ : The neighbourhood function;
Let $\varepsilon(t), \alpha(\delta)$: Rate of learning and Control distance from the $BMU$ ;
Let $p_i$ : The codeword of node i ;
Let $p_j$ : The code word for next position ($j$) ;
Let $X$: Input vector;
While the termination condition is not satisfied
{
Input a new vector $X$;
// Which vector of weight matrix for $X$ is more similar to vector $X$
  Forall $i$=1 to $n$
$d(X, W) = \|X - W_{ij}\| < T_i$ ;
$BMU$ = the winner or the Best Matching Unit with the lowest $T_i$;
  Finding the neighbours of winner in order;
  // Update the nodes in the neighbourhood of the BMU by pulling them closer to the input vector by using "The neighbourhood function"
Forall neighbours of winner node
            {
            // Usually, the SOM has $\delta_f << \delta_i$  and $\varepsilon_f << \varepsilon_i$
              $\varepsilon(t) = \varepsilon_i(\varepsilon_f/\varepsilon_i)^{t/t_f}$ ;

              $\delta(t) = \delta_i(\delta_f/\delta_i)^{t/t_f}$ ;
            $\alpha(\delta) = e^{-\|p_i - p_j\|^2/2\delta(t)^2}$ ;
            ;
            $W_{new} = W_i + \varepsilon(t) \cdot \alpha(\delta) \cdot (X - W_i)$;
            }
}  Repeat
```

Figure 3-2A: Algorithm of the self-organizing map (SOM)

```
GNG ( )
Input:  DataVector $X$;
Output: Clusters of dataset ;
```

```
Begin
Let $t$ : Time;
Let $W_i$ : vector node weight;
Let $C$ : Interconnection set of edges;
Let $X$: Input vector $X$;
Let age, edge, Error: three parameters for age, connection between two nodes and
computing distance error;
Let $E$: Error;
Let $S$, $T$, $U$, $V$: nodes;
Initialize two random nodes $S$ and $T$;
$Age_{max}$ , $Error_{max}$ = Max threshold for age and Max amount for the Error;
Let $\alpha$ , $\beta$ = parameters for decreasing error;
Set edge, age, Error = 0;
While termination condition is not satisfied
{
Input a new vector $X$;
// Finding first and second nodes closet to vector $X$
      First node = $|| W_s - X ||^2$;
      Second node = $|| W_t - X ||^2$;

If node $S$ is winner node and nodes $n$ are neighbour nodes of winner node
// $e_w$ , $e_n \in [0, 1]$
            $E_s = E_s + || W_s - X ||^2$ ;
            $W_s = W_s + e_w (X - W_s )$;
            $W_n = W_n + e_n (X - W_n )$;

Forall edges from node $S$ to its neighbours
          age = age +1;

If edge between $S$ and $T$ are not connect
          {
          Create c(S, T);
          age =age +1;
          }

Forall nodes with $age>age_{max}$
                  {
                  Delete edge;
                  Delete nodes without edge;
                  }

Forall nodes $V$ and neighbour nodes $U$ with $Error>Error_{max}$
                  {
                  Add new node $r$ between two nodes $V$ and $U$
                  $W_r = (W_u + W_v ) /2$ ;
                  Delete $edge_{uv}$ ;
                  Add $edge_{ur}$ and $edge_{rv}$ ;
```

```
                // Decrease Errors of $E_u, E_v$
                $E_u = \alpha . E_u$ ;
                $E_v = \alpha . E_v$ ;
                Set $E_r = E_u$ ;
                }
Forall nodes
        $E_j = E_j - \beta . Error_i$ ;

} Repeat
```

Figure 3-3A: Algorithm of growing neural gas (GNG)

ESOM ( )
Input: DataVector *X*;
Output: Clusters of dataset ;

```
Begin
Let $t$ : Time;
Let $N_t$: Node set at time $t$;
Let $m$ : Current number of nodes in $N_t$;
Let $W_i$: Each prototype node $W_i \in N_t$ as a dimensional vector d for $i=1,...,m$;
Let $C$ : Interconnection set;
Let $P, \varepsilon, \delta, \gamma, T_p$ : Parameter set, $P=\{\varepsilon, \delta, \gamma, T_p\}$, $\varepsilon$ is distance threshold, $\delta$ controls the spread of neighbourhood, $\gamma$ is a small constant learning rate; $T_p$ steps of learning time;
While termination condition is not satisfied
{
Input a new $X$;

If ($N=0$ or none of the existing nodes matches the input vector $X$ within a distance threshold $\varepsilon$)
      Forall $i=1$ to $m$
      {
        $d(W_i, X) = || W_i - X ||$

   If $|| W_i - X|| > \varepsilon$
         {
         // Create new node in the network
            $W_{m+1} = X ;$
            $N=N \cup W_{m+1}$ ;
            $m=m+1$;
         // Connect the new node with its two nearest neighbours $W_{n1}$ , $W_{n2}$
            If there are not any connection
                 $C_{t+1} = C_t \cup C(X, W_{n1}) \cup C(X, W_{n2}) \cup C(W_{n1}, W_{n2})$;
         }
      Else
         {
          // In one epoch, update the matching node and each of its neighbours, denoted as $W$,
          // according to their distances to the input vector $X$, modified each node and usually
          // set $\delta = \varepsilon$
                // Compute activation function
                  $D= -2 |X-Wi|^2 / \varepsilon^2$;
                 $\alpha_i(X) = e^D$;
                //   Compute neighbourhood rank related to $k$ neighbours
                 $h_{i,winner\ index}(X)= \alpha_i(X)/\sum_k \alpha_k(X)$ ;
                $\Delta W= \gamma\, h_{i,winner\ index}(X)(X-W_i)$ ;
           }

// Reset the strength of connections between the winner (or the newly created node) and its
// neighbours. The connection strength $s(i; j)$, for the connection $C(w_i;w_j)$, and $\beta=0.8$ as
// forgetting constant:
$S(i,j) = \beta S(i,j) + (1-\beta)\alpha_i(X)\alpha_j(X)$;
Prune weakness connection and isolated nodes;
} Repeat
```

Figure 3-4A: Algorithm of evolving self-organizing map (ESOM)

ESOINN ( )
Input: DataVector *X*;
Output: Clusters of dataset ;

```
Begin
Let $A$: node set $A$, used to store nodes
Let $C$: connection set or edge set, used to store connection between nodes
Let $a_1, a_2$ = first and second nearest nodes (or winners)
Let $N_i$= the set of direct topological neighbour nodes of node $i$
Let $N_A$= number of nodes in $A$
Let $W_i$= the n-dimensional weight vector of node $i$
Let $T_i$ = similarity threshold of node $i$
Let $M_i$ = local accumulated number of signals of node $i$; the number is updated when
node $i$ is the winner
Let $age_{(i,j)}$ = age of edge that connects node $i$ and node $j$.
Let $\gamma$ = the number of learning iteration
Let $h_i$ = mean accumulated point (density) of node $i$
Let $p_i$ = predefined parameter to take the place of number and use the mean of the
accumulated point of node i to describe the density of that node
Let $A_{max}$ = apex density of subset $A$
Let $B_{max}$ = apex density of subset $B$
Let $age_{max}$ = predefined max value for age
Let $c_1, c_2$ = control parameters $c_1$ for two neighbour nodes and $c_2$ for one neighbour
nodes to control the deletion,
$( 0 < c <=1 )$
Let $path$ = link between two nodes with a series of nodes
Initialization: set $A$ with two nodes of weight vectors chosen randomly from input
pattern; Add connection of $A \times A$ to empty set;

While termination condition is not satisfied
{
Input new pattern $X$;
        // Compute $T_i$ distance threshold
                        If node $i$ has $j$ neighbour nodes
                        $T_i = max_{j \in Ni} || W_i - W_j ||$;
                        Else (no neighbour)
                                // $N$ is the set of all $j$ nodes
                        $T_i = min_{j \in N\backslash(i)} || W_i - W_j ||$;
         Compute $T_{a1}$ and $Ta_2$ based on $T_i$;
          $a_1$= argmin || $X$ - $W_a$ ||;  $a_2$ = argmin || $X$ - $W_a$ || ;
          If  $a_1 > T_{a1}$  or  $a_2 > T_{a2}$
                       {
                         // Node $i$ is new,
                        Add node $i$ to set $A$
                         // Do while for input new pattern $X$
                         Exit loop;
                       }
 Increase the age of all edges linked with by 1;

 // Judge if it is necessary to build a connection between $a_1, a_2$
   If the $a_1$ or $a_2$ is a new node (it is unknown which subclass the node belongs)
```

```
                    connect two nodes with an edge; $age = 0$;
    If the $a_1$ or $a_2$ belong to the same subclass
                    connect two nodes with an edge; $age = 0$;
    If the $a_1$ belongs to subclass $A$,
                     the $a_2$ belongs to subclass $B$;
                    If $min( h_{a1} , h_{a2} ) > \alpha_A A_{max}$ Or $min( h_{a1} , h_{a2} ) > \alpha_B B_{max}$ is satisfied
                         {
                           connect the two nodes; $age = 0$;
                                combine subclasses $A$ and $B$;
                         }
                    Else
                         {
                           don't connect the two nodes;
                                If a connection exists between the two nodes
                                      remove the connection;
                         }

 // Update the density of the winner
        $h_i = 1/N \sum_{j=1}^{n}(\sum_{k=1}^{\gamma} p_i)$;
 // Add 1 to local accumulated number of signal $M_{a1}$
        $M_{a1} (t+1) = M_{a1} (t) +1$;
// Adapt the weight vectors of the winner and its direct topological neighbours by a fraction $\varepsilon_1(t)$ and $\varepsilon_2(t)$ of the total distance to the input signal,
// Adopt the learning rate over time by $\varepsilon_1(t)=1/t$ and $\varepsilon_2(t)=1/100t$

Updating weights by using
        {
        $\Delta W_{a1} = \varepsilon_1 (M_{a1})(X - W_{a1})$
        $\Delta W_i = \varepsilon_2 (M_{a1})(X - W_i)$ for all direct neighbours $i$ of $a_1$
        }
For all edges
        If $ages > age_{max}$
        Remove such edges;

If the number of input signals generated so far is an integer multiple of parameter $\gamma$
  Update the subclass label of every node by: Separating a complex class into subclasses;

// We call a node an peak of a subclass if the node has a local maximum density.
For all peaks in the complex class
         {  Give such peaks different labels;
           Classify all other nodes with the same subclass label as their peaks;
         }
// Such nodes lie in the overlapped area if the connected nodes have different subclass labels
// Delete nodes resulting from noise as follows:
//$N_A$ is the number of nodes in node set $A$;    $( 0 < c <=1 )$

          For all nodes in set $A$
                    If node $a$ has two neighbours and $h_a < c_1 \sum_{j=1}^{N_A} h_j / N_A$
                         remove the node $a$;
          For all nodes in set $A$
```

```
                    If node ahas one neighbour, and $h_a < c_2\sum_{j=1}^{N_A} h_j / N_A$
                              remove node $a$;
              For all nodes in set $A$
                       If node $a$ has no neighbour
                              remove node $a$;

If the learning process is finished
     {
     // Classify nodes to different classes
        Initialize all nodes as unclassified;
        Randomly choose one unclassified node i from node set $A$;
        Mark node $i$ as classified and label it as class $C_i$;
        For all unclassified nodes in set $A$ those are connected to node $i$ with a “path”
        Mark these nodes as classified and label them as the same class as node $i$;
     // Report the number of classes, output the prototype vectors of every class;
     // Stop the learning process;
     }
} Repeat
```

Figure 3-5A: The algorithm of enhanced self organizing incremental neural network for online unsupervised learning

$*\ note$:

In ESOINN, the parameter of “point” is used as the mean of the accumulated points of a node to describe the density of that data node. First, the mean distance of node i or d_i from its neighbours must be computed as in Equations (3.1A) and (3.2A):

$$\bar{d_\iota} = {}^{1}\!/_{m} \sum_{n}^{j=1} \| W_i - W_j \| \tag{3.1A}$$

Where, m is the number of the neighbours of node i, W_i is the weight vector of node i, while W_j is the weight vector of node i neighbours.

$$p_i = \text{point of node } i = \begin{cases} 1/(1+\bar{d_\iota})^2 & \text{, if node } i \text{ is a winner;} \\ 0 & \text{, if node } i \text{ is not a winner;} \end{cases} \tag{3.2A}$$

The sum of the data points during the period of learning is equal to $\sum_{j=1}^{n}(\sum_{k=1}^{\gamma} p_i)$, and parameter γ is the number of input signals during one learning period; n includes learning period times. The mean accumulated points (density) of node i or h_i is shown in Equation (3.3A):

$H_i = 1/N \sum_{j=1}^{n} (\sum_{k=1}^{\gamma} p_i)$ (3.3A)

The ESOINN is able to detect the overlapped clusters and separate them into the different sub-clusters with lower densities by using Equation (3.4A), then, the connections between the data nodes in the sub-clusters are renewed. Therefore, if *A* and *B* are the subclasses, A_{max} and B_{max} in the algorithm refer to the peak density of subclass *A* and the peak density of subclass *B*. Both the subclasses of *A* and *B* can be combined in one subclass through the following conditions of Equation (3.4A):

$$\text{If } min(h_{winner}, h_{secondwinner}) > \alpha_A A_{max} \quad \text{Or} \quad min(h_{winner}, h_{secondwinner}) > \alpha_B B_{max} \tag{3.4A}$$

The data nodes of the winner and the second winner lie in the overlapped area between the subclasses A and B. Parameter α is measured in the domain of [0, 1] which can be calculated by using the threshold function as in Equation (3.5A) and (3.6A):

$$\alpha_A = \begin{cases} 0 \quad if \ 2mean_A >= A_{max} \\ 0.5 \quad if \ 3mean_A >= A_{max} > 2mean_A \\ 1 \quad if \quad A_{max} > 3mean_A \end{cases} \tag{3.5A}$$

$$\boldsymbol{mean_A} = \mathbf{1}/\boldsymbol{N_A} \sum_{\mathrm{i\ A}} \mathbf{h_i} \tag{3.6A}$$

Where N_A is the number of nodes in subclass *A*.

```
**IGNGU** ( )
Input:  DataVector $X$;
Output: Clusters of dataset ;
```

```
Begin
Let $t$ : Time
Let $W_i$ : Node feature vector
Let $T$ : Threshold
Let $C$ : Class label of every feature vector in first layer or second layer
Let $type_i$= type of feature vector in first layer or second layer which can be active or
inactive
Let $M_i$ = activation number
Let $N_a$= the number of active nodes
Let $X$: Input vector $X$
Let $age, edge$: three parameters for age, connection between two nodes
Let $d_w$ : within-cluster distance
Let $d_b$ : between-cluster distance
$agemax$ = Max threshold for age
Set $edge, age$ = 0;
While termination condition is not satisfied
{
Input a new vector $X$;
// Finding first and second nodes closet to vector $X$, first winner $g_1$, second winner $g_2$
If not satisfy $(||X–W_i|| <= T_i )$ and there are not at least two nodes which satisfy this
condition (empty network)
        {
        Add new node $i$;
        $M_i = 0$;
        $W_i = X$;
        }
If in first layer, network is not empty (there are at least two nodes which satisfy the
condition of $|| X – W_i || <= T_i$)
                $T_i = || X – W_{g1} ||$;
If $g_1$ is first winner that satisfy $( || X – W_i || <= T_i )$
        If this is first layer
                $T_i = || X – W_{g1} ||$;

        Find $g_2$ second winner
                If $g_2$ does not exist or not satisfy $( || X – W_i || <= T_i )$
                        {
                        Add new node $j$;
                        $W_j = X$;
                        Add $edge\ g1$,$node\ j$; $age\ of\ edge g1$, node $j=0$;
                        In first layer: $T_j = || X – W_j ||$;
                        }
If satisfy $( || X – W_i || <= T_i )$ and there are at least two nodes which satisfy this
condition
  // Adaptive rates of winner node and neighbour nodes:  $\varepsilon_{g1} = 1/t$; $\varepsilon_v = (1/100\times t)$
    Updating weights by using
        {
        $\Delta W_{g1}(t) = \varepsilon_{g1} (X - W_{g1})$;
        Forall $v$ neighbour nodes of $g_1$
                $\Delta W_v(t) = \varepsilon_v (X - W_v)$;
```

```
            }
Forall ages of nodes
        If $age_i > age_{max}$
                {
                Delete $edge$;
                Delete nodes without edge;
                }
Forall first layer winner and its neighbour nodes
        {
        $Type_i = active$;
        $M_i = M_i + 1$;
        }
Forall first layer first winner and second winner nodes
        $T_{g1} = max_{c \varepsilon neibor node g1} || W_c - W_{g1} ||$;
        $T_{g2} = max_{c \varepsilon neibor node g2} || W_c - W_{g2} ||$;

// For removed to eliminate noise and to avoid exceeding the number of nodes
// $0 < c < 1$
If $M_i < c \sum_{j=1}^{Na} {M_j}/{N_a}$
        Delete nodes with this condition for $M_i$ ;
Forall $d_w$ within-cluster distance
        $d_w = {1}/{N_c} \sum_{\{i,j\} \epsilon C} \|W_i - W_j\|$ ;
Forall $d_b$ between-cluster distance
        $d_b(C_i, C_j) = min_{i \epsilon C_1, j \epsilon C_2} \|W_i - W_j\|$;
} Repeat
```

Figure 3-6A: Incremental growing with neural gas utility parameter (IGNGU) algorithm

```
EDSOM ( )
Input:  DataVector $X$;
Output: Clusters of dataset ;

Begin
Let $t$ : Time;
Let $N_t$: Node set at time $t$;
Let $m$ : Current number of nodes in $N_t$;
Let $W_i$: Each prototype node $W_i \in N_t$ as a dimensional vector d for $i=1,\dots,m$;
Let $C$ : Interconnection set;
Let $E$ : A set of edge connecting the nodes;
Let winning node: The best matching unit toward the input value data;
Let $\varepsilon, \delta, \gamma, T_p$ :  $\varepsilon$ is distance threshold, $\delta$ controls the spread of neighbourhood, $\gamma$ is a
small constant learning rate; $T_p$ steps of learning time;
While termination condition is not satisfied
{
Initial four connected data nodes, and $N$=4;
Initial a [m×4] matrix of the random weights;
Input a new $X$;
   If $\| W_i - X \| < \varepsilon$
        {
        Add $X$;
        // Update the winner and its direct neighbours according to the distance to $X$,
        // We usually consider $\delta = \varepsilon$ ,

        $\Delta W = \gamma e^{\left( d^2(W_i, W_{winner}) / 2\delta^2 \right)} (W_i - X)$
        }
   Else {
         // Create new node in the network
           $W_{m+1} = X$ ;
           $N = N \cup W_{m+1}$ ;
           $m=m+1$;
          If winner node is a boundary node, connect the new node to winner node,
          Else if it   is not boundary node, connect to secound winner node;
        }
 Prune weakness connection and isolated nodes;

}  Repeat
```

Figure 3-7A: Algorithm of enhancing dynamic self-organizing map (EDSOM)

```
HI-GNG ( )
Input:  DataVector $X$;
Output: Clusters of data nodes which their maturity-level are larger than the boundary maturity-level;
-----------------------------------------------------------------------------------------------

Begin
Let $t$ : Time;
Let $N_t$: Node set at time $t$;
Let $m$ : Current number of nodes in $N_t$;
Let $W_i$: Each prototype node $W_i \in N_t$ as a dimensional vector d for $i=1,\dots,m$;
Let $C$ : Interconnection set;
Let $\Delta s$ : An integer value for updating connection-strength;
Let winning node: The best matching unit toward the input value data;
Let $a_f, a_n, \delta, \varepsilon, T_p$ : $a_f, a_n$ are small constant learning rates which are integer values
between (0,1), $\varepsilon$ is distance threshold, $\delta$ controls the spread of neighbourhood, $T_p$ steps
of learning time;

While termination condition is not satisfied
{
Initial the network without any data node, and $N$=0;

Input a new $X$;

Search for first winner and second winner nodes by considering $|| W_i - X ||< \varepsilon$
{
If ($N$=0) or  (first winner node is larger than $\varepsilon$) or (seocond winner nodes is larger than $\varepsilon$)
        {
        Create two new nodes in the network;
        Set their weights by:

          $W_{first\ winner} = X$;
          $W_{second\ winner}$ is randomly computed by $|| W_{first\ winner} - W_{second\ winner} ||< \frac{\varepsilon}{10}$;
          $N=N \cup (W_{first\ winner}$ and $W_{second\ winner})$;
          $m=m+2$;
          Create connection between first winner node and second winner node;
          Connection-strength (first winner node , second winner node) ← 1;
          }
  Continue;
  }
// Update the weight vectors of the first nearest node and its neighbours:
   {
   $W_{first\ winner} \leftarrow W_{first\ winner} + a_f(X - W_{first\ winner})$ ;
   $W_{neighbour\ of\ first\ winner} \leftarrow W_{neighbour\ of\ first\ winner} + a_n(X - W_{neighbour\ of\ first\ winner})$ ;
   Increase the maturity level values of first winner and second winner by 1;
   }

If there is connection between first node and second node,
   Then update connection-strength(first and second nodes) by adding $\Delta s$;
If there is not connection between first node and second node,
    Then create a connection and increase connection-strength(first and second nodes) by 1;

Reduce the connection-strength of the connections between first winner node and its neighbours;

Remove the connections whose connection-strength is zero and if this make some nodes isolated, also remove the isolated nodes;

If the number of iterations is so far an integer multiple of the pre-initialized maximum value of maturity-level,
    Then Reduce the connection-strength of all existing connections;

} Repeat
```

Figure 3-8A: Algorithm of enhanced incremental growing neural gas (HI-GNG)

APPENDICES B

Datasets Features

Breast Cancer Wisconsin Dataset

- Number of Instances: 699 and after cleaning 683
- Number of Attributes: 11 (10 Integer + 1 binary class)
- The attribute Information :
  1. Sample code number: ID number
  2. Clump Thickness: 1 - 10
  3. Uniformity of Cell Size: 1 - 10
  4. Uniformity of Cell Shape: 1 - 10
  5. Marginal Adhesion: 1 - 10
  6. Single Epithelial Cell Size: 1 - 10
  7. Bare Nuclei: 1 - 10
  8. Bland Chromatin: 1 - 10
  9. Normal Nucleoli: 1 - 10
  10. Mitoses: 1 - 10
  11. Class: (2 for benign, 4 for malignant)
- Missing Attribute Values: Yes
- Class Distribution:

Class	# Instances
Benign	444
malignant	239

- **Sample of the Breast Cancer Wisconsin Dataset**

Id	A1	A2	A3	A4	A5	A6	A7	A8	A9	Class
1000025	5	1	1	1	2	1	3	1	1	2
1002945	5	4	4	5	7	10	3	2	1	2
1015425	3	1	1	1	2	2	3	1	1	2
1016277	6	8	8	1	3	4	3	7	1	2
1017023	4	1	1	3	2	1	3	1	1	2
1017122	8	10	10	8	7	10	9	7	1	4
1018099	1	1	1	1	2	10	3	1	1	2
1018561	2	1	2	1	2	1	3	1	1	2
1033078	2	1	1	1	2	1	1	1	5	2
1033078	4	2	1	1	2	1	2	1	1	2
1035283	1	1	1	1	1	1	3	1	1	2
1036172	2	1	1	1	2	1	2	1	1	2
1041801	5	3	3	3	2	3	4	4	1	4
1043999	1	1	1	1	2	3	3	1	1	2
1044572	8	7	5	10	7	9	5	5	4	4
1047630	7	4	6	4	6	1	4	3	1	4
1048672	4	1	1	1	2	1	2	1	1	2
1049815	4	1	1	1	2	1	3	1	1	2
1050670	10	7	7	6	4	10	4	1	2	4
1050718	6	1	1	1	2	1	3	1	1	2
1054590	7	3	2	10	5	10	5	4	4	4
1054593	10	5	5	3	6	7	7	10	1	4
1056784	3	1	1	1	2	1	2	1	1	2
1059552	1	1	1	1	2	1	3	1	1	2
1065726	5	2	3	4	2	7	3	6	1	4
1066373	3	2	1	1	1	1	2	1	1	2
1066979	5	1	1	1	2	1	2	1	1	2
1067444	2	1	1	1	2	1	2	1	1	2
1070935	1	1	3	1	2	1	1	1	1	2
...	...	...	...	...	...	...	...	...	...	...

Iris Dataset

- Number of Instances: 150
- Number of Attributes: 5 (4 Real + 1 trinary class)
- The attribute Information :
    1. Sepal length in cm
    2. Sepal width in cm
    3. Petal length in cm
    4. Petal width in cm
    5. Class: (Iris Setosa, Iris Versicolour , Iris Virginica)
- Missing Attribute Values: None
- Class Distribution:

Class	# Instances
Iris Setosa	50
Iris Versicolour	50
Iris Virginica	50

- **Sample of the Iris Dataset**

A1	A2	A3	A4	Class
5.6	2.8	4.9	2	Iris-virginica
5.7	2.5	5	2	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
6.3	2.5	5	1.9	Iris-virginica
6.7	3.1	4.4	1.4	Iris-versicolor
6	2.9	4.5	1.5	Iris-versicolor
6	2.2	5	1.5	Iris-virginica
6.6	3	4.4	1.4	Iris-versicolor
6.1	2.9	4.7	1.4	Iris-versicolor
4.9	2.5	4.5	1.7	Iris-virginica
6.1	3	4.6	1.4	Iris-versicolor
5.6	3	4.5	1.5	Iris-versicolor
5.4	3	4.5	1.5	Iris-versicolor
6.6	2.9	4.6	1.3	Iris-versicolor
5.9	3	4.2	1.5	Iris-versicolor
6.2	2.2	4.5	1.5	Iris-versicolor
6.4	2.9	4.3	1.3	Iris-versicolor
6.2	2.9	4.3	1.3	Iris-versicolor
6.1	2.8	4.7	1.2	Iris-versicolor
5.7	2.8	4.5	1.3	Iris-versicolor
6.3	2.3	4.4	1.3	Iris-versicolor
5.7	2.9	4.2	1.3	Iris-versicolor
5.6	3	4.1	1.3	Iris-versicolor
5.5	4.2	1.4	0.2	Iris-setosa
5	2.3	3.3	1	Iris-versicolor
5	2	3.5	1	Iris-versicolor
5.8	4	1.2	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
5.7	3.8	1.7	0.3	Iris-setosa
...	...	...	...	...

Spambase Dataset

- Number of Instances: 4601
- Number of Attributes: 57 (56 Integer, Real + 1 binary class)
- The attribute Information :

48 continuous real [0,100] attributes of type word_freq_WORD
= percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.

6 continuous real [0,100] attributes of type char_freq_CHAR]
= percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences)/total characters in e-mail

1 continuous real [1,...] attribute of type capital_run_length_average
= average length of uninterrupted sequences of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_longest
= length of longest uninterrupted sequence of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_total
= sum of length of uninterrupted sequences of capital letters
= total number of capital letters in the e-mail

1 nominal {0,1} class attribute of type spam
= denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

- Missing Attribute Values: Yes

- **Sample of the Spambase Dataset**

Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	...
1	0	0	0	0	0	1.29	0	0.64	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	3.19	0	0	0	0	0	0	0	...
1	0	0	1.35	1.35	0	0	0	1.35	0	0	0	0	0	0	0	...
1	0	1.03	0	0	1.03	0	1.03	0.51	0	0.51	0	1.03	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.29	...
1	0	0	0.68	0	0	0	0	1.36	0	0	0.68	0.68	0	0	0	...
1	0.1	0.2	1.01	0	0.8	0.8	0.5	0	0.8	0.1	0.3	0.7	0.3	0	1.61	...
1	0	0	0.66	0	0.33	0	0.33	0.33	1.33	2	0	0.66	0	0.33	1	...
1	0	0	0	0	0	0.23	0	0	0	0	0	0.46	0	0	0	...
1	0.39	0	0	0	0	0.39	0.79	0	0	0.39	0	0.79	0	0	0	...
1	0	0	0.77	0	0.38	0.38	0.38	0	0	0.77	0.38	0.38	0	0	0	...
1	0	0	0	0	0.53	0	0.53	0	0.53	0	0	1.07	0	0	0	...
1	0	0.31	0.42	0	0	0.1	0	0.52	0.21	0.52	0	0.52	0.63	0.1	0.1	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0.85	0	0	0	0	1.7	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	2.04	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0.24	0	0	0	...
0	0.9	0	0	0	0.9	0	0	0	0	0	0	1.8	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	1.85	0	0	0	...
0	0.08	0	0.08	0	0	0.08	0	0.49	0	0	0.08	1.48	0.08	0.08	0	...
0	0	0	0	0	1.85	0	0	0	0	0	0	0	0	0	0	...
0	0	0	1.44	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	5.88	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

SPECT Heart Dataset

- Number of Instances: 267
- Number of Attributes: 23 (22 binary + 1 binary class)
- The attribute Information :
    1. OVERALL_DIAGNOSIS: 0,1 (class attribute, binary)
    2. F1: 0,1 (the partial diagnosis 1, binary)
    3. F2: 0,1 (the partial diagnosis 2, binary)
    4. F3: 0,1 (the partial diagnosis 3, binary)
    5. F4: 0,1 (the partial diagnosis 4, binary)
    6. F5: 0,1 (the partial diagnosis 5, binary)
    7. F6: 0,1 (the partial diagnosis 6, binary)
    8. F7: 0,1 (the partial diagnosis 7, binary)
    9. F8: 0,1 (the partial diagnosis 8, binary)
    10. F9: 0,1 (the partial diagnosis 9, binary)
    11. F10: 0,1 (the partial diagnosis 10, binary)
    12. F11: 0,1 (the partial diagnosis 11, binary)
    13. F12: 0,1 (the partial diagnosis 12, binary)
    14. F13: 0,1 (the partial diagnosis 13, binary)
    15. F14: 0,1 (the partial diagnosis 14, binary)
    16. F15: 0,1 (the partial diagnosis 15, binary)
    17. F16: 0,1 (the partial diagnosis 16, binary)
    18. F17: 0,1 (the partial diagnosis 17, binary)
    19. F18: 0,1 (the partial diagnosis 18, binary)
    20. F19: 0,1 (the partial diagnosis 19, binary)
    21. F20: 0,1 (the partial diagnosis 20, binary)
    22. F21: 0,1 (the partial diagnosis 21, binary)
    23. F22: 0,1 (the partial diagnosis 22, binary)

- Data set is divided into:
  training data ("SPECT.train" 80 instances)
  testing data ("SPECT.test" 187 instances)
- Missing Attribute Values: None
- Class Distribution:

Class	# examples
0	55
1	212

Training dataset

Class	# examples

0	40
1	40

Testing dataset

Class	# examples
0	15
1	172

- **Sample of the SPECT Heart Dataset:**

OVERALL_DIAGNOSIS	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	...
1	0	0	0	1	0	0	1	0	1	0	0	1	0	1	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	1	1	0	0	1	1	1	0	0	1	1	1	0	0	...
1	0	1	1	1	0	0	1	1	1	0	1	1	1	0	...
1	1	0	1	1	1	0	0	1	1	1	0	0	1	1	...
1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	...
1	1	1	1	0	1	0	1	1	1	1	0	1	1	1	...
1	1	0	0	1	0	0	0	0	0	1	1	0	0	0	...
1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	...
1	1	0	1	1	0	0	0	1	1	1	0	0	1	1	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	...
0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	...
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	...
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	...
0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	...
0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	1	0	0	0	1	0	1	0	0	1	0	1	0	0	...
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

SPECTF Heart Dataset

- Number of Instances: 267
- Number of Attributes: 45 (44 continuous + 1 binary class)
- The attribute Information :
  1. OVERALL_DIAGNOSIS: 0,1 (class attribute, binary)
  2. F1R: continuous (count in ROI (region of interest) 1 in rest)
  3. F1S: continuous (count in ROI 1 in stress)
  4. F2R: continuous (count in ROI 2 in rest)
  5. F2S: continuous (count in ROI 2 in stress)
  6. F3R: continuous (count in ROI 3 in rest)
  7. F3S: continuous (count in ROI 3 in stress)
  8. F4R: continuous (count in ROI 4 in rest)
  9. F4S: continuous (count in ROI 4 in stress)
  10. F5R: continuous (count in ROI 5 in rest)
  11. F5S: continuous (count in ROI 5 in stress)
  12. F6R: continuous (count in ROI 6 in rest)
  13. F6S: continuous (count in ROI 6 in stress)
  14. F7R: continuous (count in ROI 7 in rest)
  15. F7S: continuous (count in ROI 7 in stress)
  16. F8R: continuous (count in ROI 8 in rest)
  17. F8S: continuous (count in ROI 8 in stress)
  18. F9R: continuous (count in ROI 9 in rest)
  19. F9S: continuous (count in ROI 9 in stress)
  20. F10R: continuous (count in ROI 10 in rest)
  21. F10S: continuous (count in ROI 10 in stress)
  22. F11R: continuous (count in ROI 11 in rest)
  23. F11S: continuous (count in ROI 11 in stress)
  24. F12R: continuous (count in ROI 12 in rest)
  25. F12S: continuous (count in ROI 12 in stress)
  26. F13R: continuous (count in ROI 13 in rest)
  27. F13S: continuous (count in ROI 13 in stress)
  28. F14R: continuous (count in ROI 14 in rest)
  29. F14S: continuous (count in ROI 14 in stress)
  30. F15R: continuous (count in ROI 15 in rest)
  31. F15S: continuous (count in ROI 15 in stress)
  32. F16R: continuous (count in ROI 16 in rest)
  33. F16S: continuous (count in ROI 16 in stress)
  34. F17R: continuous (count in ROI 17 in rest)
  35. F17S: continuous (count in ROI 17 in stress)
  36. F18R: continuous (count in ROI 18 in rest)
  37. F18S: continuous (count in ROI 18 in stress)
  38. F19R: continuous (count in ROI 19 in rest)
  39. F19S: continuous (count in ROI 19 in stress)
  40. F20R: continuous (count in ROI 20 in rest)
  41. F20S: continuous (count in ROI 20 in stress)
  42. F21R: continuous (count in ROI 21 in rest)
  43. F21S: continuous (count in ROI 21 in stress)
  44. F22R: continuous (count in ROI 22 in rest)
  45. F22S: continuous (count in ROI 22 in stress)

All continuous attributes have integer values from the 0 to 100

- Dataset is divided into:

Training data ("SPECTF.train" 80 instances)
Testing data ("SPECTF.test" 187 instances)

- Missing Attribute Values: None
- Class Distribution:

Class	# examples
0	55
1	212

Training dataset

Class	# examples
0	40
1	40

Testing dataset

Class	# examples
0	15
1	172

- **Sample of the SPECTF Heart Dataset**

OVERALL_DIAGNOSIS	F1R	F1S	F2R	F2S	F3R	F3S	F4R	F4S	F5R	F5S	F6R	F6S	F7R	F7S	…
1	59	52	70	67	73	66	72	61	58	52	72	71	70	77	…
1	72	62	69	67	78	82	74	65	69	63	70	70	72	74	…
1	71	62	70	64	67	64	79	65	70	69	72	71	68	65	…
1	69	71	70	78	61	63	67	65	59	59	66	69	71	75	…
1	70	66	61	66	61	58	69	69	72	68	62	71	71	71	…
1	57	69	68	75	69	74	73	71	57	61	72	74	73	69	…
1	69	66	62	75	67	71	72	76	69	70	66	69	71	80	…
1	61	60	60	62	64	72	68	67	74	68	76	70	74	71	…
1	65	62	67	68	65	67	71	71	64	56	73	72	68	69	…
1	74	73	72	79	66	61	76	66	65	64	78	74	62	57	…
1	70	69	60	62	58	60	71	77	69	69	73	68	68	70	…
1	67	66	65	77	66	70	72	72	72	67	76	72	73	76	…
1	76	69	78	73	68	67	75	70	77	70	79	73	79	75	…
0	69	64	73	72	49	70	66	71	57	56	64	62	76	74	…
0	70	64	68	67	76	68	76	69	67	64	69	65	62	65	…
0	65	68	70	78	65	72	72	74	64	69	71	73	72	68	…
0	64	53	74	70	65	63	70	70	57	57	64	64	73	74	…
0	70	71	71	74	68	66	72	70	66	69	74	72	70	69	…
0	72	70	75	80	73	70	76	73	66	56	72	70	73	75	…
0	70	75	72	72	67	71	71	78	63	67	73	76	71	74	…
0	59	57	67	71	66	68	68	70	56	62	77	61	67	71	…
0	67	64	73	75	77	77	74	70	65	62	74	75	65	67	…
0	68	65	72	72	47	74	76	74	67	66	71	69	69	67	…
0	66	54	69	66	69	69	75	72	63	62	68	66	68	70	…
1	75	71	54	51	53	50	68	69	46	55	11	12	43	48	…
1	77	61	62	68	62	58	72	68	77	71	76	77	72	75	…
1	75	72	75	79	72	68	79	77	69	66	73	77	67	73	…
1	78	76	71	72	65	71	75	74	70	64	65	76	65	73	…
1	69	68	75	74	78	72	75	72	61	57	72	71	75	72	…
1	72	66	75	67	61	59	64	63	61	67	75	76	66	48	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Musk1 (Musk version 1) Dataset

- Number of Instances: 476
- Number of Attributes: 167 (166 Integer + 1 binary class)
- The attribute Information :

molecule_name: Symbolic name of each molecule. Musks have names such as MUSK-188. Non-musks have names such as NON-MUSK-jp13.
conformation_name: Symbolic name of each conformation. These have the format MOL_ISO+CONF, where MOL is the molecule number, ISO is the stereoisomer number (usually 1), and CONF is the conformation number.
f1 through f162: These are "distance features" along rays (see paper cited above). The distances are measured in hundredths of Angstroms. The distances may be negative or positive, since they are actually measured relative to an origin placed along each ray. The origin was defined by a "consensus musk" surface that is no longer used. Hence, any experiments with the data should treat these feature values as lying on an arbitrary continuous scale. In particular, the algorithm should not make any use of the zero point or the sign of each feature value.
f163: This is the distance of the oxygen atom in the molecule to a designated point in 3-space. This is also called OXY-DIS.
f164: OXY-X: X-displacement from the designated point.
f165: OXY-Y: Y-displacement from the designated point.
f166: OXY-Z: Z-displacement from the designated point.
Class: Musk and Non-Musk

- Missing Attribute Values: None
- Class Distribution:

Class	# Instances
Musk	207
Non-Musk	269

- **Sample of the Musk1 (Musk version 1) Dataset**

Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	...
MUSK-188	42	-198	-109	-75	-117	11	23	-88	-28	-27	-232	-212	-66	-286	...
MUSK-188	42	-191	-142	-65	-117	55	49	-170	-45	5	-325	-115	-107	-281	...
MUSK-188	42	-191	-142	-75	-117	11	49	-161	-45	-28	-278	-115	-67	-274	...
MUSK-188	42	-198	-110	-65	-117	55	23	-95	-28	5	-301	-212	-107	-280	...
MUSK-190	42	-198	-102	-75	-117	10	24	-87	-28	-28	-233	-212	-67	-286	...
MUSK-190	42	-191	-142	-65	-117	55	49	-170	-45	6	-324	-114	-106	-280	...
MUSK-190	42	-190	-142	-75	-117	12	49	-161	-45	-29	-277	-114	-68	-274	...
MUSK-190	42	-199	-102	-65	-117	55	23	-94	-29	6	-299	-212	-106	-280	...
MUSK-211	40	-173	-142	13	-116	-7	50	-171	-44	-103	-321	-117	-242	-286	...
MUSK-211	44	-159	-63	-74	-117	17	5	-114	-31	-33	-287	-243	-73	-314	...
MUSK-212	42	-170	-63	-65	-117	58	11	-136	-33	7	-320	-247	-105	-315	...
MUSK-212	41	-95	-61	-75	-117	15	30	-164	-12	-25	-254	-204	-67	-294	...
MUSK-212	45	-199	-108	13	-117	-6	24	-96	-26	-102	-296	-217	-242	-286	...
MUSK-213	41	90	-141	12	-116	-8	49	-169	-44	-103	-322	-118	-243	-288	...
MUSK-213	70	-30	-61	-73	-117	11	12	-118	-32	-27	-284	-252	-71	-317	...
MUSK-213	85	-158	-63	-74	-117	18	5	-114	-31	-32	-287	-243	-72	-314	...
MUSK-213	50	-192	-143	34	214	55	50	-173	-44	-8	-317	-116	-81	-282	...
MUSK-219	46	-194	-148	34	-117	55	53	-200	-45	-8	-321	-253	-83	-329	...
MUSK-219	47	-102	-60	-113	-117	-127	35	-166	-14	-32	-265	-198	-86	-292	...
MUSK-224	47	-197	-144	33	-117	60	65	-44	-28	-10	-195	-110	-101	-190	...
MUSK-224	48	-100	-58	-78	-117	-65	43	-86	-4	-46	-190	-187	-149	-202	...
MUSK-227	43	-192	-151	-80	-117	-71	41	6	-45	-49	-168	-99	-144	-204	...
MUSK-227	40	-198	-160	-69	-117	27	35	-61	-31	21	-104	-95	-92	-186	...
MUSK-228	49	-197	-145	28	-117	-87	63	-13	-27	-173	-53	27	-95	-90	...
MUSK-228	38	-168	-157	31	-117	-94	42	40	-32	-106	-56	-57	-50	-94	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Musk2 (Musk version 2) Dataset

- Number of Instances: 6598
- Number of Attributes: 167 (166 Integer + 1 binary class)
- The attribute Information :

molecule_name: Symbolic name of each molecule. Musks have names such as MUSK-188. Non-musks have names such as NON-MUSK-jp13.
conformation_name: Symbolic name of each conformation. These have the format MOL_ISO+CONF, where MOL is the molecule number, ISO is the stereoisomer number (usually 1), and CONF is the conformation number.
f1 through f162: These are "distance features" along rays (see paper cited above). The distances are measured in hundredths of Angstroms. The distances may be negative or positive, since they are actually measured relative to an origin placed along each ray. The origin was defined by a "consensus musk" surface that is no longer used. Hence, any experiments with the data should treat these feature values as lying on an arbitrary continuous scale. In particular, the algorithm should not make any use of the zero point or the sign of each feature value.
f163: This is the distance of the oxygen atom in the molecule to a designated point in 3-space. This is also called OXY-DIS.
f164: OXY-X: X-displacement from the designated point.
f165: OXY-Y: Y-displacement from the designated point.
f166: OXY-Z: Z-displacement from the designated point.
class: 0 => non-musk, 1 => musk
Class: Musk and Non-Musk

- Missing Attribute Values: None
- Class Distribution:
  Entire data

Class	# Instances
Musk	1017
Non-Musk	5581

- **Sample of the Musk2 (Musk version 2) Dataset**

Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	…
MUSK-211	46	-108	-60	-69	-117	49	38	-161	-8	5	-323	-220	-113	-299	…
MUSK-211	41	-188	-145	22	-117	-6	57	-171	-39	-100	-319	-111	-228	-281	…
MUSK-211	46	-194	-145	28	-117	73	57	-168	-39	-22	-319	-111	-104	-283	…
MUSK-211	41	-188	-145	22	-117	-7	57	-170	-39	-99	-319	-111	-228	-282	…
MUSK-211	41	-188	-145	22	-117	-7	57	-170	-39	-99	-319	-111	-228	-282	…
MUSK-211	46	-194	-145	28	-117	72	57	-168	-39	-22	-319	-112	-104	-284	…
MUSK-211	47	-199	-106	28	-117	73	27	-104	-22	-23	-269	-210	-105	-285	…
MUSK-211	41	-199	-101	22	-117	-6	26	-99	-21	-101	-293	-213	-229	-285	…
MUSK-211	41	-199	-101	22	-117	-6	26	-100	-21	-101	-293	-213	-229	-285	…
MUSK-211	47	-199	-106	28	-117	73	27	-104	-22	-23	-269	-210	-105	-285	…
MUSK-211	41	-199	-101	22	-117	-6	26	-100	-21	-101	-293	-213	-229	-285	…
MUSK-211	44	-131	-62	-71	-117	54	20	-142	-19	10	-317	-254	-98	-310	…
MUSK-211	44	-130	-62	-71	-117	54	20	-142	-19	10	-317	-255	-97	-310	…
MUSK-211	44	-129	-62	-71	-117	54	20	-141	-19	10	-317	-255	-98	-310	…
MUSK-211	46	-107	-60	-77	-117	10	38	-160	-8	-18	-261	-220	-65	-298	…
MUSK-211	44	-131	-62	-77	-117	12	20	-126	-19	-24	-285	-255	-68	-310	…
MUSK-211	44	-132	-62	-76	-117	11	20	-126	-19	-25	-285	-255	-68	-310	…
MUSK-211	46	-107	-60	-77	-117	10	38	-160	-8	-18	-261	-220	-65	-299	…
MUSK-211	46	-107	-60	-78	-117	10	38	-160	-8	-18	-261	-220	-65	-299	…
MUSK-212	33	-158	-62	-71	-117	50	26	-146	-18	8	-320	-259	-103	-311	…
MUSK-212	33	-158	-61	-77	-117	9	26	-131	-18	-22	-284	-259	-66	-312	…
MUSK-212	32	-92	-60	-78	-117	10	38	-160	-5	-20	-263	-230	-65	-300	…
MUSK-212	33	-104	-62	-78	-117	9	32	-141	-10	-21	-268	-251	-65	-305	…
MUSK-212	33	-93	-60	-99	-117	73	37	-161	-5	17	-268	-229	-40	-298	…
MUSK-212	33	-93	-60	-99	-117	73	37	-161	-5	17	-268	-229	-39	-298	…
MUSK-212	33	-158	-62	-51	-117	-56	26	-129	-19	-52	-319	-258	-241	-311	…
MUSK-212	33	-159	-62	-99	-117	73	26	-147	-17	17	-276	-259	-38	-312	…
MUSK-212	32	-92	-60	-51	-117	-57	38	-154	-6	-53	-319	-230	-243	-299	…
MUSK-212	32	-93	-60	-50	-117	-57	38	-154	-6	-53	-319	-230	-244	-299	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Arcene Dataset

- Number of Instances: 900
- Number of Attributes: 10001 (10000 Real + 1 binary class)
- Source:

Original owners: The data were obtained from two sources: The National Cancer Institute (NCI) and the Eastern Virginia Medical School (EVMS). All the data consist of mass-spectra obtained with the SELDI technique. The samples include patients with cancer (ovarian or prostate cancer), and healthy or control patients.

Donor of database: This version of the database was prepared for the NIPS 2003 variable and feature selection benchmark by Isabelle Guyon, 955 Creston Road, Berkeley, CA 94708, USA (isabelle '@' clopinet.com).

- Data set information:

ARCENE was obtained by merging three mass-spectrometry datasets to obtain enough training and test data for a benchmark. The original features indicate the abundance of proteins in human sera having a given mass value. Based on those features one must separate cancer patients from healthy patients. We added a number of distractor feature called 'probes' having no predictive power. The order of the features and patterns were randomized.

ARCENE	Positive ex.	Negative ex.	Total
Training set	44	56	100
Validation set	44	56	100
Test set	310	390	700
All	398	502	900

Number of variables/features/attributes:
Real: 7000
Probes: 3000
Total: 10000
This dataset is one of five datasets used in the NIPS 2003 feature selection challenge.
Our website [Web Link] is still open for post-challenge submissions. Information about other related challenges are found at:[Web Link]. The CLOP package includes sample code to process these data: [Web Link].

All details about the preparation of the data are found in our technical report: Design of experiments for the NIPS 2003 variable selection benchmark, Isabelle Guyon, July 2003, [Web Link] (also included in the dataset archive). Such information was made available only after the end of the challenge.

The data are split into training, validation, and test set. Target values are provided only for the 2 first sets. Test set performance results are obtained by submitting prediction results to: [Web Link].

The data are in the following format:
dataname.param: Parameters and statistics about the data
dataname.feat: Identities of the features (withheld, to avoid biasing feature selection).
dataname_train.data: Training set (a coma delimited regular matrix, patterns in lines, features in columns).

dataname_valid.data: Validation set.
dataname_test.data: Test set.
dataname_train.labels: Labels (truth values of the classes) for training examples.
dataname_valid.labels: Validation set labels (withheld during the benchmark, but provided now).
dataname_test.labels: Test set labels (withheld, so the data can still be use as a benchmark).

- Attribute Information:

The attribute information is not provided to avoid biasing the feature selection process.

- Class: Patients with cancer (ovarian, prostate cancer) and healthy patients
- Missing Attribute Values: N/A

  Class Distribution: 10

Class	# Instances
Patients with cancer (ovarian, prostate cancer)	88
Healthy patients	112

- **Sample of the Arcene Dataset**

Class	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14
1	0	71	0	95	0	538	404	20	0	0	0	0	17	...
-1	0	41	82	165	60	554	379	0	71	0	0	0	0	...
1	0	0	1	40	0	451	402	0	0	0	0	0	15	...
1	0	56	44	275	14	511	470	0	0	0	0	0	0	...
-1	105	0	141	348	0	268	329	0	0	1	0	0	23	...
-1	38	62	0	251	75	515	0	9	85	300	0	52	0	...
1	76	80	236	213	0	324	361	21	0	0	0	0	116	...
-1	47	4	207	222	0	323	130	0	0	32	0	0	0	...
-1	0	17	5	82	0	395	471	0	0	0	0	0	32	...
-1	38	113	186	456	86	261	129	0	82	62	132	0	0	...
-1	0	0	102	111	0	553	400	0	34	0	0	0	0	...
1	17	26	0	299	0	501	0	0	17	144	98	0	0	...
-1	0	81	42	179	34	492	447	0	14	0	0	0	65	...
1	106	6	116	438	0	246	410	0	0	0	0	0	34	...
-1	7	6	0	199	0	440	0	0	39	363	10	0	0	...
1	71	34	184	341	0	296	266	0	0	0	0	0	50	...
-1	13	58	17	488	38	339	177	0	30	125	109	0	74	...
-1	0	0	0	543	64	453	0	0	150	177	166	0	0	...
-1	132	0	171	394	0	247	129	0	0	0	0	0	0	...
-1	3	46	0	426	0	533	0	16	113	165	103	0	0	...
-1	29	21	0	366	50	531	0	0	146	152	64	0	11	...
-1	168	63	151	470	0	315	86	6	0	29	20	0	77	...
-1	0	1	43	97	28	477	420	0	33	0	15	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Yeast Dataset

- Number of Instances: 1484
- Number of Attributes: 9 (8 predictive, 1 name)
- Source:

Original owners:
Title: Protein Localization Sites
Creator and Maintainer:
Kenta Nakai
Institue of Molecular and Cellular Biology
Osaka, University
1-3 Yamada-oka, Suita 565 Japan
nakai@imcb.osaka-u.ac.jp
http://www.imcb.osaka-u.ac.jp/nakai/psort.html
Donor: Paul Horton (paulh@cs.berkeley.edu)
Date: September, 1996
See also: ecoli database

Past Usage:
Reference: "A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins", Paul Horton & Kenta Nakai, Intelligent Systems in Molecular Biology, 109-115. St. Louis, USA 1996.
Results: 55% for Yeast data with an ad hoc structured probability model. Also similar accuracy for Binary Decision Tree and Bayesian Classifier methods applied by the same authors in unpublished results.

Predicted Attribute: Localization site of protein. (non-numeric).

The references below describe a predecessor to this dataset and its development. They also give results (not cross-validated) for classification by a rule-based expert system with that version of the dataset.

Reference: "Expert Sytem for Predicting Protein Localization Sites in Gram-Negative Bacteria", Kenta Nakai & Minoru Kanehisa, PROTEINS: Structure, Function, and Genetics 11:95-110, 1991.

Reference: "A Knowledge Base for Predicting Protein Localization Sites in Eukaryotic Cells", Kenta Nakai & Minoru Kanehisa, Genomics 14:897-911, 1992.

Attribute Information.
1. Sequence Name: Accession number for the SWISS-PROT database
2. mcg: McGeoch's method for signal sequence recognition.
3. gvh: von Heijne's method for signal sequence recognition.
4. alm: Score of the ALOM membrane spanning region prediction program.
5. mit: Score of discriminant analysis of the amino acid content of the N-terminal region (20 residues long) of mitochondrial and non-mitochondrial proteins.
6. erl: Presence of "HDEL" substring (thought to act as a signal for

retention in the endoplasmic reticulum lumen). Binary attribute.

7. pox: Peroxisomal targeting signal in the C-terminus.
8. vac: Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins.
9. nuc: Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins.

8. Missing Attribute Values: None.

9. Class Distribution. The class is the localization site. Please see Nakai and Kanehisa referenced above for more details.

CYT (cytosolic or cytoskeletal)	463
NUC (nuclear)	429
MIT (mitochondrial)	244
ME3 (membrane protein, no N-terminal signal)	163
ME2 (membrane protein, uncleaved signal)	51
ME1 (membrane protein, cleaved signal)	44
EXC (extracellular)	37
VAC (vacuolar)	30
POX (peroxisomal)	20
ERL (endoplasmic reticulum lumen)	5

- **Sample of the Yeast Dataset**

Class	A1	A2	A3	A4	A5	A6	A7	A8	A9
ABP1_YEAST	0.4	0.39	0.6	0.15	0.5	0	0.58	0.3	CYT
ACE1_YEAST	0.43	0.39	0.54	0.21	0.5	0	0.53	0.27	NUC
ACE2_YEAST	0.42	0.37	0.59	0.2	0.5	0	0.52	0.29	NUC
ACH1_YEAST	0.4	0.42	0.57	0.35	0.5	0	0.53	0.25	CYT
ACON_YEAST	0.6	0.4	0.52	0.46	0.5	0	0.53	0.22	MIT
ACR1_YEAST	0.66	0.55	0.45	0.19	0.5	0	0.46	0.22	MIT
ACT_YEAST	0.46	0.44	0.52	0.11	0.5	0	0.5	0.22	CYT
ACT2_YEAST	0.47	0.39	0.5	0.11	0.5	0	0.49	0.4	CYT
ACT3_YEAST	0.58	0.47	0.54	0.11	0.5	0	0.51	0.26	NUC
ACT5_YEAST	0.5	0.34	0.55	0.21	0.5	0	0.49	0.22	NUC
ADA2_YEAST	0.61	0.6	0.55	0.21	0.5	0	0.5	0.25	NUC
C1TC_YEAST	0.45	0.4	0.5	0.16	0.5	0	0.5	0.22	CYT
PUR4_YEAST	0.43	0.44	0.48	0.22	0.5	0	0.51	0.22	CYT
PUR3_YEAST	0.73	0.63	0.42	0.3	0.5	0	0.49	0.22	CYT
ADH1_YEAST	0.43	0.53	0.52	0.13	0.5	0	0.55	0.22	CYT
ADH2_YEAST	0.46	0.53	0.52	0.15	0.5	0	0.58	0.22	CYT
ADH3_YEAST	0.51	0.51	0.52	0.51	0.5	0	0.54	0.22	MIT
ADH4_YEAST	0.59	0.45	0.53	0.19	0.5	0	0.59	0.27	CYT
KAD1_YEAST	0.57	0.47	0.6	0.18	0.5	0	0.51	0.22	CYT
KAD2_YEAST	0.63	0.67	0.57	0.24	0.5	0	0.49	0.22	MIT
ADP1_YEAST	0.8	0.88	0.36	0.39	0.5	0	0.56	0.33	ME1
ADR1_YEAST	0.53	0.54	0.43	0.1	0.5	0	0.57	0.32	NUC
ABF2_YEAST	0.55	0.5	0.66	0.36	0.5	0	0.49	0.22	MIT
...	...	...	...	...	...	...	...	...	...

Breast Cancer Dataset from the UMMC

- Number of Instances: 827
- Number of Attributes:14 (13 countinues+ 1 binary class)
- The attribute Information :
  1. AGE: Patient's age in year at time first diagnosis
  2. RACE: Ethnicity (Chinese, Malay, Indian and Others)
  3. STG: Stage (how far cancer has spread anatomically)
  4. T: tumours type (the extent of the primary tumours)
  5. N: Lymph node type (amount of regional lymph node involvement)
  6. M: Metastatic (presence or absence)
  7. LN: Number of node involved
  8. ER: Estrogen receptor (negative or positive)
  9. GD: tumours grade
  10. PT: Primary treatment (type of surgery performed)
  11. AC: Adjuvant Chemotherapy
  12. AR: Adjuvant Radiotherapy
  13. AT: Adjuvant Tamoxifen
  14. Class: Alive - Dead
- Missing Attribute Values: None
- Class Distribution:

Class	# Instances
Alive	437
Dead	246

- The number of data in each subset:
  Subset 1: 827 (Class 0: 790, Class 1: 37)
  Subset 2: 673 (Class 0: 604, Class 1: 69)
  Subset 3: 561 (Class 0: 462, Class 1: 99)
  Subset 4: 440 (Class 0: 323, Class 1: 117)
  Subset 5: 355 (Class 0: 244, Class 1: 111)
  Subset 6: 270 (Class 0: 172, Class 1: 98)
  Subset 7: 200 (Class 0: 118, Class 1: 82)
  Subset 8: 124 (Class 0: 63, Class 1: 61)
  Subset 9: 56 (Class 0: 26, Class 1: 30)

- **Sample of the Breast Cancer Dataset from the UMMC**

Year	RACE	AGE	DOD	DODeath	Status	STG	T	N	M	LN	ER	GD	P2	AC	AR	AT
1993	3	66	May-93		0	1	1	0	0	0	2	4	3	0	1	1
1993	3	66	Sep-93		1	3	2	1	0	1	3	4	3	0	0	1
1993	3	43	Dec-93		1	2	2	0	0	0	2	4	1	0	1	1
1993	2	30	Jul-93	20/07/1995	1	3	2	1	0	2	2	4	1	1	1	0
1993	1	65	Aug-93	15/11/1995	1	7	2	0	1	0	2	4	1	0	0	0
1993	2	41	Jun-93		0	4	3	1	0	6	2	4	3	1	1	1
1993	1	55	Jun-93	03/08/2001	1	3	2	1	0	0	2	4	3	1	1	1
1993	1	36	Jul-93		1	5	3	1	0	1	3	4	3	1	0	1
1993	1	51	Dec-92	28/03/1994	1	3	2	1	0	1	3	4	3	0	1	1
1993	1	58	Feb-93		0	3	2	1	0	5	2	4	3	1	1	1
1993	1	42	Oct-93		0	4	3	1	0	3	2	4	3	1	1	1
1993	1	31	Feb-93	28/05/1996	1	2	1	1	0	4	2	4	1	1	1	1
1993	1	30	May-93		0	1	1	0	0	0	2	1	3	0	0	1
1993	1	48	Nov-93	30/06/1994	1	7	4	0	1	0	2	4	3	1	0	1
1993	3	72	Sep-93		0	2	2	0	0	0	2	4	1	0	0	1
1993	4	64	Nov-93	04/04/2000	1	3	2	1	0	10	3	4	3	1	0	1
1993	3	54	Nov-93	20/11/1996	1	2	2	0	0	0	2	4	3	0	0	1
1993	1	74	Feb-93	30/06/1997	1	2	2	0	0	0	2	4	3	0	0	1
1993	1	59	Mar-93		0	2	2	0	0	0	2	4	3	0	0	1
1993	1	42	Oct-93		0	3	2	1	0	2	2	4	1	1	1	1
1993	1	49	Apr-93	11/06/1997	1	2	2	0	0	0	3	4	3	0	0	1
1993	1	46	Mar-93		0	2	2	0	0	0	2	4	3	0	0	1
1993	1	66	Dec-93		0	3	2	1	0	3	2	2	3	0	0	1
1993	1	45	Feb-93	31/12/2001	1	2	2	0	0	0	2	4	1	0	1	1
1993	4	70	Absc		0	3	2	1	0	2	2	3	3	0	1	1
1993	1	44	Feb-93		0	3	2	1	0	2	2	4	3	1	0	1
1993	1	47	Jan-93		1	3	2	1	0	3	2	4	3	1	0	1
1993	1	46	Mar-93	21/02/1996	1	5	4	1	0	0	2	4	3	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...