An Investigation of Clustering Algorithms and Soft Computing Approaches for Pattern Recognition

A Thesis Submitted to Assam University
In Partial Fulfillment of the Requirements
for the Degree of

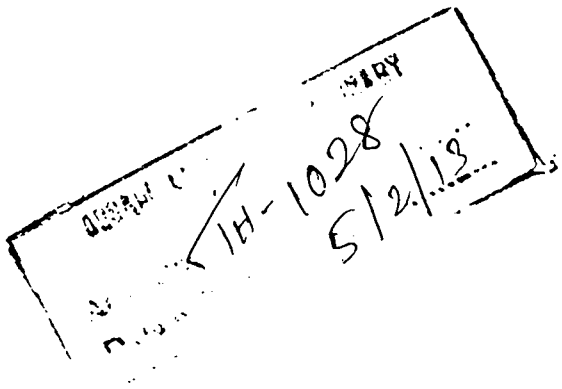
Doctor of Philosophy
In Computer Science

By

Alok Chakrabarty
Ph. D. Registration No. - PhD/543/2007

Department of Computer Science
School Of Physical Sciences
Assam University Silchar
Assam, India PIN-788011

July 2010



DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF PHYSICAL SCIENCES
ASSAM UNIVERSITY SILCHAR
(A CENTRAL UNIVERSITY CONSTITUTED UNDER ACT XIII OF 1989)
Silchar-788011, Assam, India

Date: 27-7-2010

CERTIFICATE

Certified that the thesis entitled "An Investigation of Clustering Algorithms and Soft Computing Approaches for Pattern Recognition" submitted by Shri Alok Chakrabarty, Regn. No. PhD/543/2007 of 2007-08 for award of the Degree of Doctor of Philosophy in Computer Science is the outcome of a bona fide research work. This work has not been submitted previously for any other degree of this or any other university. It is further certified that the candidate has compiled this thesis fulfilling all the formalities as per the requirements of Assam University. I recommend that the thesis may be placed before the examiners for consideration of award of the degree of this university.

Bipul Syam Purkayastha

(Dr. Bipul Syam Purkayastha)
Supervisor & Associate Professor
Department of Computer Science,
Assam University, Silchar

DECLARATION

I, Alok Chakrabarty, bearing Registration Number PhD/543/2007 dated 22-11-2007, hereby declare that the subject matter of the thesis entitled "An Investigation of Clustering Algorithms and Soft Computing Approaches for Pattern Recognition" is the record of works done by me and that the contents of the thesis did not form the basis for award of any other degree to me or to anybody else to the best of my knowledge. The thesis has not been submitted in any other University / Institute.

This thesis is being submitted to Assam University for the degree of Doctor of Philosophy in Computer Science.

Alok Chakrabarty

Place: Silchar
Date: 27-7-2010

(Alok Chakrabarty)
Research Scholar

To my parents

Shri Anil Kumar Chakrabarty & Shrimati Anita Chakrabarty

and my sisters

Madhuchanda Chakrabarty and Maitreyee Chakrabarty

ACKNOWLEDGEMENTS

I address my sincere gratitude to the Almighty for the inspiration and stamina generated within me to undertake the challenge and complete this assignment.

I am indebted to my supervisor, Dr. Bipul Syam Purkayastha for his immense contribution in my whole PhD work and express my sincere thanks and gratitude to him. He, as my guide, provided me constant encouragement and support through inspirations and suggestions. I am also thankful to Prof. D. Biswas, Dean, School of Physical Sciences, Mr. P. K. Deva Sarma, Head, Department of Computer Science and to Dr. (Mrs.) Shahin Ara Begum, Reader, Department of Computer Science, for their valuable support and suggestions. I am grateful to Prof. T. Bhattacharjee, Hon'ble Vice Chancellor, Assam University Silchar for his inspiring advices. I also express my sincere thanks to all my teachers, staff members and all of them who helped me directly or indirectly to accomplish my research work.

I convey my sincere gratitude to Prof. Pushpak Bhattacharyya of IIT Bombay for his precious help and encouragement. He helped me a lot in understanding the importance of WordNet and its use in text clustering.

I am very thankful to the editors of ICFAI Journal of Information Technology, and AUIJST Journal for publication of my research papers in their esteemed Journals. I am also thankful to the editors' committee of IEEE Advanced Computing Conference 2009, of Global WordNet Conference, 2010 and of National Conference on Current Trends in Computer Science 2010, for publication of my research papers in their esteemed conference proceedings.

I must thank Mr. Prodipto Das, Assistant Professor, Department of Computer Science, Assam University Silchar and my co-researcher, for his valuable suggestions and help.

Last but not the least; I would like to express my gratefulness to my family members especially to my beloved parents Shri Anil Kumar Chakrabarty and Smt. Anita Chakrabarty, who have given me all the support, opportunity and encouragement in completing the work. I also thank my two sisters Miss Madhuchanda and Miss Maitreyee for their continuous support, care and encouragement.

Date: 27-7-2010

Alokchakrabarty

(ALOK CHAKRABARTY)

Table of Contents

List of Figures

List of Tables

Abstract

Pattern recognition is a scientific discipline which is becoming increasingly important in this age of information technology due to the rapid increase in the dependency on machine intelligence based systems for decision making, for which pattern recognition is vital. As a field of research, pattern recognition has been evolving since the early 1950s, in close connection with the emergence and evolution of computer technology. The goal of pattern recognition is the classification of objects into meaningful classes by the discovery of the inherent structure in the given set of objects. The traditional computing approach to pattern recognition is a straightforward yes/no process. It is not able to utilize imprecision, ambiguity and uncertainty in decision making. This presents a big opportunity for the application of soft computing approaches in pattern recognition, which aim at achieving tractability, robustness and close resemblance to human like decision making so as to obtain low cost solutions. Pattern recognition is also done using clustering, which is a key unsupervised technique for discovering the inherent structure in any given data set. Investigation on application of soft computing techniques like fuzzy logic to clustering, for better pattern recognition, is also a lucrative research problem. Thus in this thesis, the effectiveness of various clustering algorithms and soft computing approaches have been investigated from a pattern recognition point of view.

The research work initiated with a study on pattern recognition and on its paradigms and various approaches. Then a study was done on some prominent soft computing approaches and on their applications in pattern recognition. Some clustering algorithms were then investigated to evaluate their performance and to understand their shortcomings. A study and investigation was also done on various existing clustering validity assessment indices. The major outcomes of these studies include the obtainment of two new clustering techniques and enhanced forms of two validity indices. A software package, named 'PatternWiz', was then developed, by integrating together the implementations of all the clustering algorithms and soft computing approaches investigated in the research. All the experimental results that are presented in this thesis were obtained using this software package, PatternWiz, on a modern computer with 3 GB RAM and Intel Core 2 Duo 2 GHz processor.

CHAPTER 1

Introduction

This thesis contributes to the subject area of Pattern Recognition. It is focused on the study and analysis of Clustering algorithms and Soft Computing approaches from a pattern recognition perspective and also on the possible enhancements of existing Clustering algorithms and existing clustering Validity assessment Indices.

Pattern recognition is the assignment of a physical object or event to one of several pre-specified categories. It is the act of taking in raw data and taking an action based on the category of the pattern [1].

A pattern is a type of theme of recurring events or objects, sometimes referred to as elements of a set, which repeat in a predictable manner. Watanabe [2] defines a pattern as, "As opposite of a chaos; it is an entity, vaguely defined, that could be given a name." For example, a pattern could be the similarity trend in sepal lengths, sepal widths, petal lengths and petal widths of different flowers of same species, a mathematical correlation or trend in numbers, an image of a printed character, or a set of terms that 'characterize' each text document in a collection of text documents.

Clustering is a fundamental technique in pattern recognition. It is a key technique for discovering the inherent structure in any given data set. It partitions a given data set into subsets (clusters), so that the data in each subset share some common trait - often proximity according to some defined distance measure. The goal of any clustering technique is to discover 'natural' groupings in a set of patterns, points, or objects, without prior knowledge of any class labels. Formally, clustering is an unsupervised process of grouping a given set of unlabeled patterns into a number of clusters such that similar patterns are assigned to one cluster [3] [4] [5].

Soft computing refers to a collection of methodologies in computer science, artificial intelligence, machine learning and some engineering disciplines which work synergistically and provides in one form or another, flexible information processing capabilities for handling real life ambiguous situations. Soft computing approaches aim at achieving tractability, robustness, low cost solutions, and close resemblance to human like decision making. [6] [7].

1.1 Motivation

Pattern recognition is a very active research area, which overlaps with various other research fields such as Machine Learning, Artificial Intelligence, Data Mining, Probability Theory, Algebra and Calculus. Rigorous researches in the field of pattern recognition have been going on from decades, but there is still more demand of research. Though pattern recognition problems are rather simple for human beings with ordinary intelligence, the man-made machine, 'computer', with so immense computing power, has still not become intelligent enough to solve these problems efficiently. This is because in traditional computing, computers use conventional arithmetic algorithms to detect whether a given pattern matches an existing one or not, which is a straightforward yes/no process [8]. In traditional computing, the primary considerations are precision, certainty and rigor; this is why traditional computing is also called 'hard' computing [9]. Thus the traditional hard computing approach to pattern recognition is not suitable for many real life problems involving uncertainty, imprecision and noisy data. Such problems thus present a great opportunity for the application of soft computing approaches, which can tolerate noise and imprecision and can handle real life ambiguous situations quite well, in order to respond to unknown patterns in a better way. As such, a study of soft computing approaches is included in the present research.

The motivation here for focus on clustering is the fact that clustering is a key process in Pattern Recognition and machine learning. In fact, clustering is a primary goal of pattern recognition [10]. In addition, clustering algorithms are used in many applications, such as image segmentation, data mining, statistical data analysis, etc.

Therefore, finding efficient clustering algorithms and clustering validity assessment indices is very important for researchers in many different disciplines [11].

1.2 Objectives

The primary objectives of this thesis can be summarized as follows:

1) To study the paradigms and different approaches to pattern recognition.
2) To study some prominent soft computing approaches and their applications in pattern recognition.
3) To study and investigate some existing hard and soft computing based clustering algorithms.
4) To develop an efficient hard clustering algorithm based on an existing popular hard computing based clustering algorithm by identifying its shortcomings.
5) To develop an efficient hybrid soft computing based clustering algorithm.
6) To study and propose enhancements over some existing clustering validity assessment indices, which are used for validating the results obtained by a clustering algorithm.
7) To develop a software package to aid researchers working in the area of pattern recognition to study and compare the working of different clustering algorithms and soft computing approaches against standard benchmark data and other custom data.

1.3 Methodology

1) A general overview of pattern recognition paradigms and approaches was first obtained.
2) Study of some prominent soft computing approaches and their applications in pattern recognition was then done.
3) Some standard hard and soft computing based clustering algorithms were then investigated to evaluate their performance and to understand their shortcomings.

4) Since clustering is unsupervised, so a study was also done on various existing validity assessment indices used for validation of clustering results.
5) Based on the study on clustering algorithms, new clustering techniques were then proposed by addressing some shortcomings of the algorithms investigated.
6) Based on the study on validity assessment indices, a common enhancement, applicable to two different forms of an existing clustering validity assessment index, was then proposed, resulting in two enhanced validity indices.
7) A software package was then developed by integrating together the implementations of all the clustering algorithms and soft computing approaches investigated in the research, so as to obtain a valuable research aid for researchers working in the area of pattern recognition.

1.4 Main Contributions

The main contributions of this thesis are:

1) Studies and analysis of clustering algorithms and soft computing approaches for pattern recognition.
2) Studies and analysis of clustering validity assessment indices.
3) Development of an efficient deterministic clustering algorithm, HBDKM, based on the classical K-means clustering algorithm.
4) Development of an efficient hybrid fuzzy clustering algorithm, DpsFCM.
5) Formulation of the *Hybrid split* rule for *k*-d tree partitioning.
6) Discovery of an enhancement, applicable to two well known clustering validity assessment indices, *PBM* index (for hard clustering) and *PBMF* index (for fuzzy logic based clustering).
7) The development of a software package PatternWiz to aid researchers working in the area of pattern recognition to study and compare the working of different clustering algorithms and soft computing approaches against standard benchmark data and other custom data.

1.5 Thesis Outline

Chapter 2: Basics of Pattern Recognition. This chapter presents an overview of pattern recognition, its different paradigms and standard approaches. It also presents some applications of pattern recognition. Further, it also presents a brief discussion on pattern recognition on data from some different domains.

Chapter 3: Soft Computing Approaches. This chapter presents a discussion on some of the prominent soft computing approaches and their applications in pattern recognition. An experiment on the use of an artificial neural network in pattern recognition on images is also presented in this chapter.

Chapter 4: Clustering Algorithms and Validity Indices. This chapter begins with a brief overview on clustering, similarity measures and different taxonomical representations of clustering. It then presents a discussion on some prominent clustering algorithms and some popular clustering validity assessment indices. Further, the chapter also presents some experiments on the applications of clustering algorithms for pattern recognition on data from different domains.

Chapter 5: An Efficient Deterministic K-means Clustering Algorithm. This chapter presents an efficient deterministic version of the widely used K-means clustering algorithm. The chapter also presents experimental results, obtained by applying the proposed algorithm on a number of benchmark data sets, to illustrate its efficiency over the K-means algorithm.

Chapter 6: An Efficient Deterministic psFCM Clustering Algorithm. This chapter presents an efficient deterministic version of the psFCM clustering algorithm. The chapter also presents experimental results, obtained by applying the proposed algorithm on a number of benchmark data sets, to illustrate its efficiency over the pshFCM, psFCM and FCM clustering algorithms.

Chapter 7: Enhancement on the PBM Validity Index. This short chapter presents a proposed enhancement, applicable on both the hard and fuzzy forms of the well-known and highly efficient *PBM* validity Index. Experimental results are presented to show the better performance of the proposed validity indices over the existing hard and fuzzy forms of the *PBM* validity index.

Chapter 8: PatternWiz - A software package for pattern recognition. This chapter presents the features and working of PatternWiz, a software package developed in MATLAB 7, as one of the primary objective of the research.

Chapter 9: Conclusions. Finally, this chapter presents the conclusion. Summary of the works and contributions are outlined along with a discussion on scope for future research work.

CHAPTER 2

Basics of Pattern Recognition

This chapter presents a discussion on some of the fundamental concepts of pattern recognition, viz. representation of a pattern, composition of a typical pattern recognition system, pattern recognition paradigms and well known approaches to pattern recognition. Further, the chapter also presents in brief some applications of pattern recognition in different problem domains. Brief discussion on pattern recognition on data from some different domains is also presented.

2.1 Overview

The ability to recognize and classify patterns is one of the most fundamental characteristics of human intelligence. Humans possess highly developed sophisticated skills for sensing their environment and taking actions according to their observation, e.g., deducing trends in a given series of numbers, understanding images, recognizing a face, recognizing digits, understanding and classifying text, understanding spoken words etc. For modern age computing it is important to extend similar capabilities to machines also, and for this the machines must have pattern recognition capabilities.

Pattern recognition is a scientific discipline whose aim is to use machine intelligence for classification of a given set of objects into meaningful classes by discovering the inherent structure in the input objects based on either a priori knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually vectors of measurements or observations defining points in an appropriate multidimensional feature space.

A diagrammatic representation of a pattern **x** in a d-dimensional feature space X is shown in Figure 2.1. The feature space X is also referred to as a data set

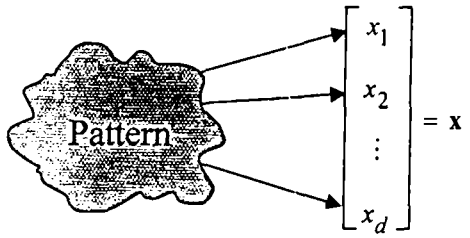


Figure 2.1: A pattern as a d-dimensional feature vector x in feature space X

Pattern recognition is an integral part in most machine intelligence based systems built for decision making. As a field of research, pattern recognition has been evolving since the early 1950s, in close connection with the emergence and evolution of computer technology [12] [13] [14].

2.2 The Pattern Recognition System

There are three fundamental problems in pattern recognition. The first is the *sensing problem* which is concerned with the representation of input data obtained by measurements on objects that are to be recognized. The second is the *feature extraction problem* which is concerned with the extraction of characteristic features from the input data. The features should be characterizing attributes so that the pattern classes can be well discriminated. The third problem is the *classification* or *discrimination problem* which involves the determination of optimal decision procedures for the classification of given patterns. Thus the design of a pattern recognition system essentially involves the following three aspects:

1) Data acquisition and preprocessing.
2) Data representation.
3) Decision making.

The problem domain dictates the choice of sensor, preprocessing technique, representation scheme, and the decision making model [10] [14].

Any typical pattern recognition system is principally consisted of [13] [1]:

- A *sensor and preprocessing module* that gathers the observations to be classified or described and optionally does any preprocessing like noise removal, filtering, normalization, outlier removal.
- A *feature extraction mechanism* that computes numeric or symbolic information from the observations.
- A *classification or description scheme* (consisting of a classifier and a learning algorithm) that does the actual job of classification or description of observations, relying on the extracted features.

The classification or description scheme is usually based on an a priori information availability of a set of patterns that have already been classified or described. This set of patterns is termed the *training set*, and the resulting learning strategy is characterized as *supervised learning*. Learning can also be unsupervised, in the sense that the system is not given an a priori labeling of patterns, instead it itself establishes the classes based on the statistical regularities of the patterns. This is called *unsupervised learning*.

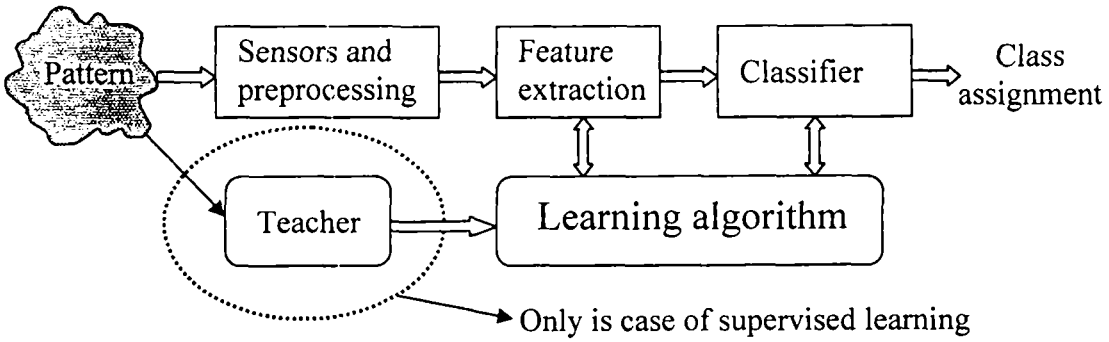


Figure 2.2: Components of a typical pattern recognition system

A pattern recognition system can optionally have a teacher module also, which is present only is case of supervised learning. It provides class information of a pattern from a training set to the learning algorithm which refines the classification accuracy of the classifer based on the class information obtained from the training set (in case of supervised learning) and based on the class information obtained from already classified patterns.

Thus in terms of information availability and hence learning strategy, two broad general paradigms for pattern recognition can be identified which are supervised and unsupervised [15]. The two paradigms are described in the following section.

2.3 Pattern Recognition Paradigms

The two broad pattern recognition paradigms namely supervised and unsupervised pattern recognition are presented below:

2.3.1 Supervised Pattern Recognition

In supervised pattern recognition or classification the recognizer or classifier is trained by a suitable learning technique for deducing a discrimination or classification function from the training data. Using this knowledge base the classifier then tries to identify input patterns in the supplied data as members of different predefined classes.

A classifier is typically represented as a set of discriminant functions

$$f_k(\mathbf{x}): X \rightarrow \Re,\ k = 1,2,\ldots,K,$$

where $\Re$ is the set of real numbers and K is the number of classes. A classifier obtains the class identifer value 'κ' for a pattern $\mathbf{x}$ in X. The constant κ can take one of K possible values, corresponding to the K classes, for each pattern $\mathbf{x}$ in X.

A classifier assigns a feature vector $\mathbf{x}$ to the i^{th} class if $f_i(\mathbf{x}) > f_j(\mathbf{x}), \forall i \neq j$

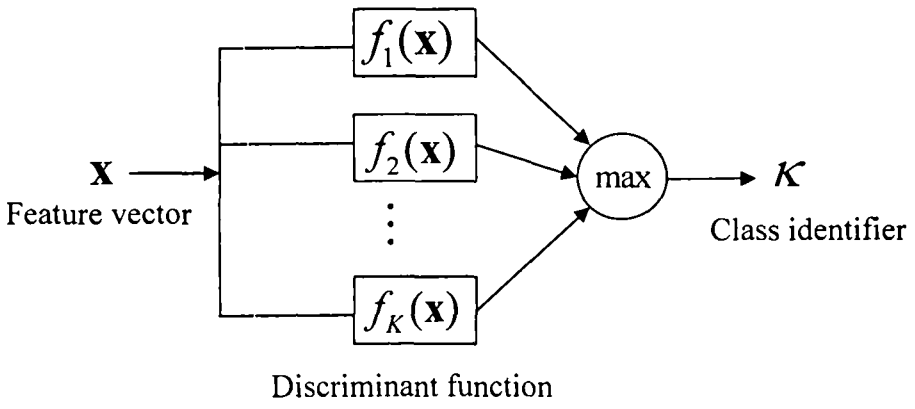


Figure 2.3: A typical classifier

2.3.2 Unsupervised Pattern Recognition

In unsupervised pattern recognition or classification the recognizer or classifier has no explicit target outputs or environmental evaluations associated with each input. It classifies the input patterns into several hitherto unknown classes based on the similarity of patterns and by learning the statistical structure of the overall collection of input patterns. Clustering is a type of unsupervised pattern recognition.

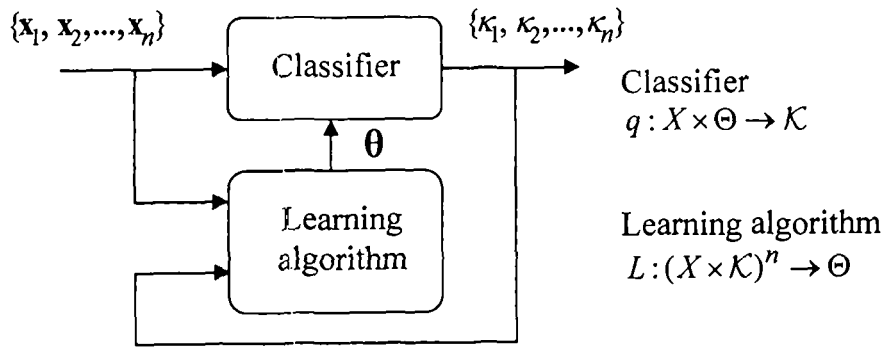


Figure 2.4: Unsupervised pattern recognition of *n* input patterns

In Figure 2.4, the output $\{\kappa_1, \kappa_2,\ldots,\kappa_n\}$ is the n-dimensional class information vector corresponding to the n input patterns $\{\mathbf{x}_1, \mathbf{x}_2,\ldots,\mathbf{x}_n\}$ in X. θ denotes the set of K feature vectors which are identified in each iteration of classification as class representatives of the K classes. Θ is the set of all θ's. $\mathcal{K}$ is the set of all κ_j's.

2.4 Approaches for Pattern Recognition

There are five main approaches to pattern recognition which are described below [16]:

2.4.1 Template Matching

Template matching based approach is one of the simplest and earliest approaches to pattern recognition. Matching is a generic operation in pattern recognition which is used to determine the similarity between two entities (points, curves, or shapes) of the

same type. In template matching, the patterns to be recognized are directly compared or matched against a few stored examples or templates (prototype of the pattern), that are representative of the underlying classes. The similarity measure, often a correlation, may be optimized based on the available training set. Often, the template itself is learned from the training set. Template matching is mostly used for recognition of patterns in images. Typically a template is a 2-dimensional shape in case of images. Template matching is computationally demanding, but the availability of modern age faster processors has now made this approach more feasible. The strict template matching mentioned above, while effective in some application domains, has a number of disadvantages. For example, it would fail if the patterns are distorted due to the imaging process, view point change, or large intra class variations among the patterns. Because of the large variations often encountered in the examples, template matching is not the most effective approach to pattern recognition.

The template matching process on an image is shown in Figure 2.5. The process moves a template image to all possible positions in a larger source image and computes a numerical index that indicates how well the template matches the portion of the image in that position. Match is done on a pixel-by-pixel basis.

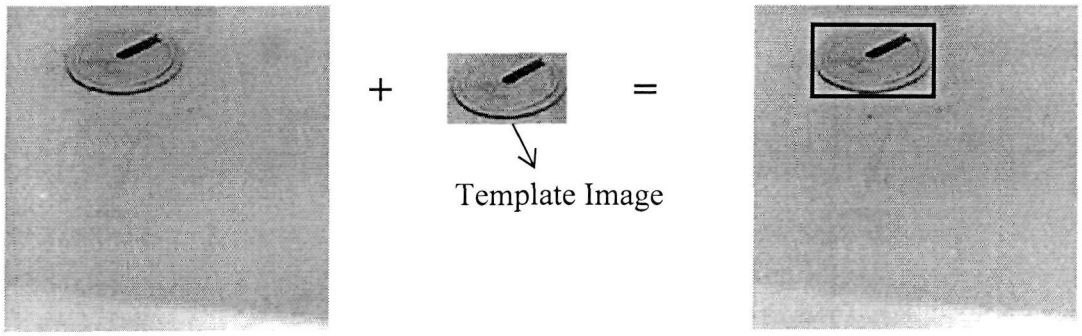


Figure 2.5: Template matching on an image

2.4.2 Geometrical Classification

In geometrical classification the pattern classes are represented by regions in the representation space defined by simple functions such that the training examples are classified as correctly as possible. Suppose, the average value of (height, weight) of

an adult women is (5'5", 55 kg) and that of an adult men is (5'11", 70 kg). A simple geometric woman vs. man classifier using (height, weight) as a two-dimensional representation may simplistically divide the representation space into two triangular regions. So, a person with (height, weight) = (5'2", 54 kg) will be classified by this classifier as a woman. However such a classification may not always be correct, like in this case if the (height, weight) pair (5'2", 54 kg) happens to be of an adult man [16].

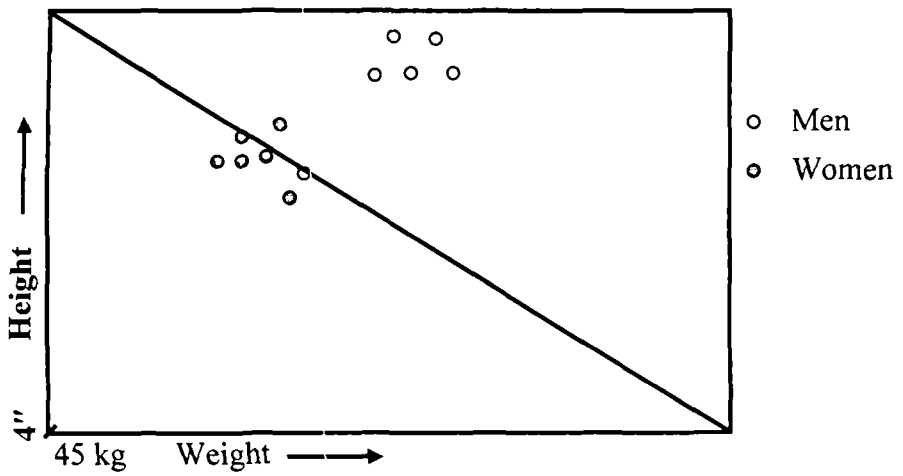


Figure 2.6: Geometrical classification

2.4.3 Statistical Approach

In the statistical approach, each pattern is represented in terms of d features or measurements and is viewed as a point in a d-dimensional space. The goal is to choose those features that allow pattern vectors belonging to different categories to occupy compact and disjoint regions in the d-dimensional feature space. The effectiveness of the representation space (feature set) is determined by how well patterns from different classes can be separated. The data points or patterns are assumed to be drawn from a probability distribution, where each pattern has a certain probability of belonging to a class. Given a set of training patterns from each class, the objective is to establish decision boundaries in the feature space which separate patterns belonging to different classes. In the statistical decision theoretic approach, the decision boundaries are determined by the probability distributions of the patterns

belonging to each class, which must either be specified or learned. The individual patterns are placed into different classes based on quantitative information on one or more characteristics inherent to the patterns (referred to as traits, features, etc) and based on a training set of previously labeled patterns [17] [18]. Continuing with the preceding example, a statistical classifier may estimate the statistical distribution of the two features, namely, height and weight of the two classes of interest (women and men) from known samples. At any coordinate or point in the representation space, one could estimate the probability of it being a man or a woman; depending upon which probability is higher, one could determine the class of an entity. This method differs from the geometrical method in that the classes are not pre-defined in terms of any regular shapes in the representation space.

2.4.4 Syntactic Approach

In many pattern recognition problems involving complex patterns, it is more appropriate to adopt a hierarchical perspective where patterns are viewed as being constructed from simpler sub patterns in a hierarchical fashion [19] [20]. The simplest elementary sub patterns to be recognized are called *primitives* and the given complex pattern is represented in terms of the interrelationships between these primitives. This helps in dividing the recognition task into easier subtask of first identifying sub patterns and only then the actual patterns.

The syntactic approach follows the above mentioned strategy, i.e., in the syntactic approach, a complex pattern (e.g., animal) is described in terms of component patterns (e.g., hair and head, or, torso and limbs) and their relationship (e.g., articulated joints). Each object to be recognized can be represented by a variable-cardinality set of symbolic, nominal features. This allows for representing pattern structures, taking into account more complex interrelationships between attributes than is possible in the case of flat, numerical feature vectors of fixed dimensionality, that are used in statistical pattern recognition approach.

In syntactic pattern recognition, a formal analogy is drawn between the structure of patterns and the syntax of a language. The patterns are viewed as sentences belonging to the language, the primitives are viewed as the alphabet of the language,

and the sentences are generated according to a grammar. Thus, a large collection of complex patterns can be described by a small number of primitives and grammatical rules. The grammar for each pattern class must be inferred from the available training samples. Strategies for learning such a language (defining the structure) from examples are problematic and may lead to many difficulties which are primarily concerned with the segmentation of noisy patterns (to detect the primitives) and the inference of the grammar from training data [21] [22].

Syntactic pattern recognition can be used instead of statistical pattern recognition if there is a clear structure in the patterns. The approach is intuitively appealing because, in addition to classification, this approach also provides a description of how the given pattern is constructed from the primitives. For example, syntactic pattern recognition can be used to find out what objects are present in an image. Furthermore, structural methods are strong in finding a correspondence mapping between two images of an object [23].

2.4.5 Neural Networks

The neural network approach applies biological concepts to computers to recognize patterns. Typically, a neural network or an artificial neural network (ANN) is a self-adaptive trainable system that is able to learn to resolve complex problems based on available knowledge. A set of available data is supplied to the system so that it finds the most adapted function among an allowed class of functions that matches the input. The approach is a very attractive since it requires minimum a priori knowledge, and with enough layers and neurons, an ANN can create any complex decision region [24] [25]. More details on neural network based pattern recognition are presented in chapter 3 on Soft Computing Approaches.

2.5 Applications

Pattern recognition has a wide range of applications. It is used in many areas of science and engineering that studies the structure of observations. The following are

some of the problem domains where pattern recognition finds its applications:

(i) Bioinformatics: In this domain, pattern recognition is used for sequence analysis of input DNA/protein sequences. The pattern classes consist of known types of genes/patterns.

(ii) Biometric recognition: Typical application of pattern recognition in this domain may involve personal identification of authorized users for access control by their iris, face, fingerprint and heart beat recognition.

(iii) Data Mining: In this domain pattern recognition involves searching of meaningful patterns in input set of data points represented in a multidimensional space. The goal is to obtain compact and well separated classes.

(iv) Document Classification: Document classification is particularly useful in searching documents of interest, especially on the internet. Pattern recognition in this case involves the classification of documents based on the semantic categories of the texts in the documents e.g. financial, business, sports, media etc.

(v) Document Image Analysis: Pattern recognition in this domain typically concerns with the extraction of alphanumeric characters or words from an image of a document supplied as input. It is particularly useful for reading machine for blind persons.

(vi) Industrial automation: Typical application of pattern recognition in this domain is the inspection of printed circuit board to identify the defective/ non-defective nature of the product by analyzing input intensity images or range images.

(vii) Multimedia Database Retrieval: In this domain a typical application of pattern application can be the classification of video clips into meaningful video genres like action, comedy, dialogue, tragedy etc.

(viii) Remote Sensing: In this vast domain a typical application of pattern recognition can be the forecasting of crop yield based on the classification of input multispectral images into different classes of land use categories and growth pattern of crops.

(ix) Speech Recognition: A typical application of pattern recognition in this domain may be the speech to text conversion by the classification of input speech waveform into pattern classes of spoken words.

Other than the above domains discussed, pattern recognition also has ample applications in the domain of Agriculture, Artificial Intelligence, Computer-Aided Diagnosis, Computer-Aided Manufacturing, Ethnology, Financial and Stock Market Analysis, Forensics, Marketing, Medical Imaging, Psychology, Robotics, Surveillance, Signal Analysis etc [10] [13] [24].

2.6 Pattern Recognition in different Data Domains

A brief discussion on pattern recognition on data from some different domains is now presented:

2.6.1 Number Theory

Number theory is the branch of pure mathematics concerned with the properties of numbers in general, and integers in particular. Numbers can depict interesting and useful patterns. Pattern recognition on numbers can be viewed as pattern recognition on a set of single-featured (1-dimensional) patterns. That is each pattern (a number) in the input data (sequence of numbers) has only one feature that is the value of the number itself. The recognition in case of number patterns thus involves the deduction of certain mathematical rule or formula that is followed by a subset of numbers (the recognized class) in the input sequence. The deduced or learnt rule can then be used to generate the other successive numbers (patterns) of the recognized class of numbers. In this subsection, a discussion is presented regarding the pattern recognition on one such fascinating class of numbers that has intrigued many mathematicians and researchers from centuries. It is the class of *Prime Numbers*.

Prime numbers or simply primes are a special class of numbers which are being studied from thousands of years. The study of the distribution of prime numbers is a recurring and productive theme in number theory. The identification and generation of

prime numbers is central to present day cryptographic systems for information security. Prime numbers are also used for hash tables and pseudorandom number generators. Further, prime number generation is also used as a benchmark for comparing the hardware performance and the capabilities of compilers. Use of a generalized prime-number-based matrix strategy facilitates efficient iconic indexing of symbolic pictures [26] [27].

Formally, a prime number (or simply a prime) is a natural number that has exactly two distinct natural number divisors: 1 and the number itself. The first 25 prime numbers are:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

The first known algorithm to generate prime numbers was formulated around 240 B.C. by the Egyptian Librarian and Mathematician Eratosthenes. After that, rigorous researches have been done by many researchers and mathematicians for the deduction of some trend in their identification and generation, but till date, a series, a formula or a conjecture to generate all prime numbers deterministically from 2 to a desired limit N remains a confounded mystery of the most unpredictable nature in number theory. Though subsets of the class of prime numbers depict a number of patterns or trend in them, but no unique and universally applicable trend is depicted by all the members in the sifted class or set 'Primes', the class of all prime numbers. Thus till date, the recognition or classification of prime numbers from the input set of natural numbers is done based on a superset of the set 'Primes' called 'Pseudoprimes' using the sieve approach (inspired by the algorithm by Eratosthenes) [27] [28].

The sieve approach is based on the Sieve theory, which is a set of general techniques in number theory, designed to count, or more realistically to estimate the size of sifted sets of integers. One of the successful approaches in sieve theory is to approximate a specific sifted set of numbers (e.g. the set of prime numbers) by another simpler set (e.g. the set of almost prime numbers), which is typically somewhat larger than the original set, and easier to analyze. The set of primes up to some prescribed limit N is an elemental example of a sifted set. And the elemental example of a sieve is the Sieve of Eratosthenes [29].

Any sieve approach to recognize primes basically consists of two steps:

Step 1): Generating a pseudo-prime set of numbers that are not multiples of the first k prime numbers.

Step 2): Sieving the set to remove all the non-primes.

The sieve of Eratosthenes algorithm is the first known prime generating algorithm that uses this sieve approach. Theoretically one can also have the k^{th} extension of this algorithm, where k is for the k^{th} prime number. In the k^{th} extension of this algorithm the initial pseudo-prime set of numbers will not contain the multiples of the first k prime numbers [27].

The sifted set 'Pseudoprimes' is composed of a special kind of number called a *pseudoprime* which literally means a false prime, i.e., an integer which shares a property common to all prime numbers but which may not actually be a prime. The set 'Pseudoprimes' can be viewed as an approximation of the set 'Primes', which is used in the sieve approach of prime generation because it is easier to analyze and for this set some trends indeed exist that is globally applicable to all the members of the set 'Pseudoprimes'. One such trend (or rule) is the 'Wheel sieve' trend.

The wheel sieve consists of repeatedly executing the above two steps of sieve approach phase by phase. In each phase it generates a set of numbers W_{k+1} based on a earlier set of numbers W_k and then removing the non-primes in the newly generated set W_{k+1}. W_k is defined as $R(\pi_k)$, where $R(y) = \{z \mid 1 \le z < y \text{ and } \gcd(z, y) = 1\}$, π_k = product of primes from 2 to P_k, P_k = the k^{th} prime and gcd(m, n) = the greatest common divisor of m and n. $|W_{k+1}| > |W_k|$. W_k is called the k^{th} wheel [30].

The $(k+1)^{th}$ wheel W_{k+1} that is generated using the k^{th} wheel W_k is defined as:

$$W_{k+1} = W_k \cup \{q\pi_k + b \mid q \in \{1, 2, ..., P_{k+1} - 1\} \text{ and } b \in W_k\}$$

The sieve introduced by the wheel sieve method consists basically in using the prime number P_{k+1} to sieve the new wheel W_{k+1} by generating all multiples of P_{k+1} and removing them from W_{k+1}. The rest of the non-primes in W_{k+1} are then eliminated using any other sieving process. This is done so, as the positions of non-primes and their intervals in the wheel W_{k+1} does not follow any sequence like in the case of any k^{th} extension of Sieve of Eratosthenes mentioned above. Rather in the case of wheel

sieve (Figure 2.7) non-primes are removed by writing the numbers of each wheel W_i around circles (wheels) in a specific manner. Numbers in the innermost wheel are then radially coincided with their multiples in similar positions as themselves in the outer wheels, forming spokes of primes and their multiples. Thus the name 'wheel sieve'.

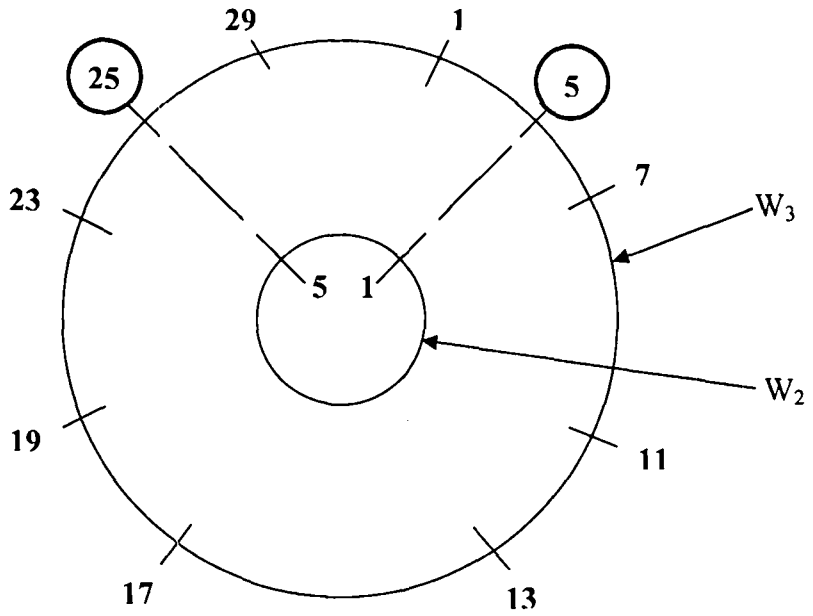


Figure 2.7: Wheel within wheel! W_2 having its multiples in W_3, situated radially

2.6.2 Images

An image (from Latin *imago*) is an artifact, for example a two-dimensional picture, which has a similar appearance to some subject — usually a physical object. For a computer, an image or more formally a digital image is a representation of a two-dimensional image using ones and zeros. It is a rectangular grid of pixels which has a definite height and a definite width counted in pixels [31] [32].

From the pattern recognition point of view an image can be considered as an aggregate of some specific objects or regions of interests, for example an image of human face may be considered as a collection of objects like eyes, mouth, ears, nose, hair etc. An example is shown next:

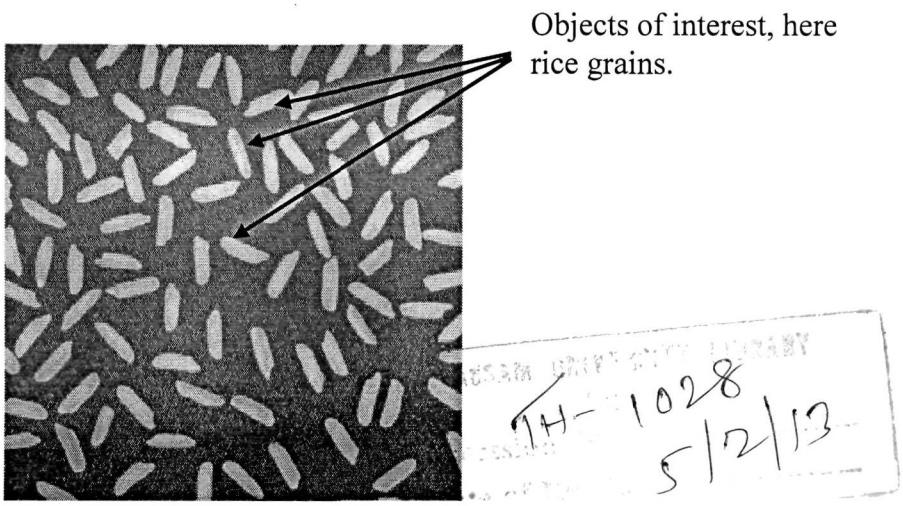


Figure 2.8: Rice grains as patterns (objects of interest) in an image

The classical problem of pattern recognition on images or simply image recognition is thus concerned with determining whether or not the image data contains some specific object or feature such as simple geometric objects (e.g., a polygon), human faces, printed or hand-written characters, and in specific situations, objects typically described in terms of well-defined illumination, background, and pose relative to the camera. Other different varieties of image recognition may include:

- Demarcating regions of interest (like face or tumor) in a given image.
- Clustering an image based on RGB pixel value differences or on the basis of pixel intensities.
- Optical Character Recognition (identifying printed or handwritten text characters in images).
- Edge Detection (identifying edges, i.e., points in an image at which the image brightness changes sharply or more formally has discontinuities).
- Content-based image retrieval (finding all images in a larger set of images which have a specific content, like finding all images similar to image *X*, or say finding all images containing houses with no cars).
- Pose estimation (estimating the position or orientation of a specific object relative to the camera. An example application for this technique would be assisting a robot arm in retrieving objects from a conveyor belt in an assembly line situation).

Demonstration of optical character recognition of printed and handwritten digits as a type of image recognition is presented in chapter 3.

2.6.3 Text Documents

In this age of information technology, a huge amount of information needs to be dealt with everyday, and with the growth of the World Wide Web and the information society, more and more information is getting available and accessible rapidly. Most of the information that is dealt with is stored in the form of text documents. The main problem is how to find the truly relevant information among these huge and large data sources. Most of the applications to search through these large data sources, like a search engine, obtain a large number of irrelevant results and a small number of relevant pages that meet the user requirements. Pattern recognition on text documents, also sometimes referred to as Document Classification, thus involves providing an effective and intuitive navigation mechanism to organize a large amount of retrieval results by grouping the text documents into a small number of meaningful classes by recognizing patterns in the text documents with an aim to provide conceptual views of document classes [33] [34]. Ideally, document classification should involve the discovery of semantic, lexical and ontological relations in the texts of a given sets of documents. Such classification may be very useful in applications like email filtering for filtering out spam e-mails.

Document classification is, however, mostly done using clustering. Most of the current methods for document clustering are based on the similarity between the text sources. The similarity measures work on the syntactical relationships between these sources but neglect the semantic information in them. The collection of documents is used to first generate a matrix using the popular Vector Space Model, in which each document is represented as a vector of term weights of words in the collection, regardless of the order of words [35]. After the generation of the matrix, different well known clustering algorithms are applied on it to obtain the document classes. Many well known methods of text clustering, however, have two main problems:

1) First, they don't consider semantically related words/terms and ignore important relationships between terms like synonyms, antonyms, hypernyms

hyponyms, etc in the documents. For example, they treat {movie, film and cinema} as altogether different terms even though these words have very similar meaning. The usual clustering techniques cannot identify the semantic relations between such set of words. The problem may lead to a very low relevance score for relevant documents because the documents do not always contain the same forms of words/terms.

2) Second, the vector representations of documents tend to use all the words/terms in the documents after removing the stop-words (an, and, any, are, as, at, be, the etc) regardless of their meaning and semantic relations with other words in the collection of documents. This leads to thousands of dimensions in the vector representation of documents. However, it is well known that only a very small number of words/terms in documents have distinguishable power to classify documents [36] and become the key elements of text summaries. Those words/terms are normally the concepts in the domain related to the documents.

Recent research on the above problems have revealed that both the problems can be solved to a great extent by using WordNet lexical categories and WordNet ontology in order to create a well structured document vector space whose low dimensionality allows common clustering algorithms to perform well [33] [34] [37] [38] [39].

A WordNet is a lexical reference system for a language, like English, whose design is inspired by the current psycholinguistic theories of human lexical memory [40]. In a WordNet, nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct lexical concept or sense. These synsets are interlinked by means of conceptual-semantic and lexical relations. With each synset, WordNet provides a short and general definition for that sense. As it stores the lexical information in terms of word meanings whose organization conforms to the current psycholinguistic theories of human lexical memory, so it can also be termed as a lexicon based on psycholinguistic principles [41].

Example demonstration of document classification using clustering is presented in chapter 4.

2.7 Chapter Summary

In this chapter, a discussion on some fundamental concepts of pattern recognition, viz., representation of a pattern, composition of a typical pattern recognition system, supervised and unsupervised pattern recognition paradigms and well known approaches to pattern recognition like template matching, geometrical classification, statistical pattern recognition, syntactic pattern recognition and artificial neural network based pattern recognition is presented. Further the chapter also presented in brief, some applications of pattern recognition in different problem domains. Brief discussion of pattern recognition on data from number theory domain, image domain and textual domain is also presented in this chapter.

CHAPTER 3

Soft Computing Approaches

This chapter presents a brief discussion on some of the prominent soft computing approaches and their applications in pattern recognition. The chapter also presents a demonstration of optical character recognition of printed and handwritten digits as a type of image recognition using Artificial Neural Networks.

3.1 Overview

Soft Computing is a term that was coined by Lotfi Zadeh [6] to describe a collection of methodologies that aims to simulate human like decision making by exploiting the tolerance for impression, uncertainty, approximate reasoning and partial truth to achieve tractability, robustness and low-cost solutions. The principal notion of soft computing is that, precision and certainty carry a cost and as such whenever possible imprecision and uncertainty should be exploited as in the case of human intelligence. The role model for soft computing is the human mind and it intends at a formalization of the cognitive processes humans employ so effectively in the performance of daily tasks. Thus the soft computing approaches provide a strong foundation for the conception and design of high Machine Intelligence Quotient systems and therefore form the basis of future generation computing systems [7] [9].

As per wide acceptance, the most prominent components of soft computing are Fuzzy Logic (FL), Artificial Neural Networks (ANN), Genetic Algorithms (GA) and Probabilistic Reasoning (PR) with the latter subsuming belief networks, chaos theory and parts of learning theory [42] [6], where FL deals with imprecision arising from

vagueness, ANN provides the machinery for learning and adaptation, GA deals with optimization and searching and PR provides the means for dealing with uncertain information. Recently Rough Sets (RS) have also been proposed as a soft computing technique which provides means for dealing with uncertainty arising from limited discernibility of objects [7]. It is important to note that although there are substantial areas of overlap between FL, ANN, GA and PR, in general the components are complementary rather than competitive. For this reason, it is at times quite advantageous to use them in combination rather than exclusively.

In the present research the main focus was given on two of the prominent soft computing approaches mentioned above, namely Fuzzy Logic and Artificial Neural Networks. The two approaches are now presented.

3.2 Fuzzy Logic

Fuzzy logic is logic system, which is based on the Fuzzy Set Theory introduced by Lotfi Zadeh in 1965 [43]. In fuzzy set theory the membership of elements in a set is described with the help of a membership function valued in the real unit interval [0, 1]. Thus the elements of a set in fuzzy set theory can have degrees of membership. For example, if X denotes a set of group of people and A denotes a fuzzy set of old people in X then the elements of the set A (which will be a subset of X) is characterized by assigning to each element **x** of X a real value, such as 0.62, in the range of 0 to 1, as the degree of membership of **x** in A. A membership degree of 0 will indicate that the person **x** in X is 'not at all' old and a value of 1 will indicate that **x** is a 'perfectly old' person, any value in between 0 and 1 will indicate the varying degree of oldness of **x**. In contrast, in classical set theory the membership of elements in a set is assessed in binary terms according to a bivalent condition – an element either belongs or does not belong to the set. Fuzzy sets thus generalize classical sets, since the indicator functions (*a function defined on a set X that indicates membership of an element in a subset A of X, having the value* 1 *for all elements of A and the value* 0 *for all elements of X not in A*) of classical sets are special cases of the membership functions of fuzzy sets, as the latter takes only the values 0 or 1 [44] [45]. In fuzzy set

theory the classical bivalent sets are usually called *crisp* sets.

The fuzzy logic, based on the concept of the fuzzy sets, is thus very appropriate to deal with vague (imprecise) propositions like "this person is old", "this woman is very beautiful", "this boy is too tall" etc which are very difficult to deal with, using classical logic.

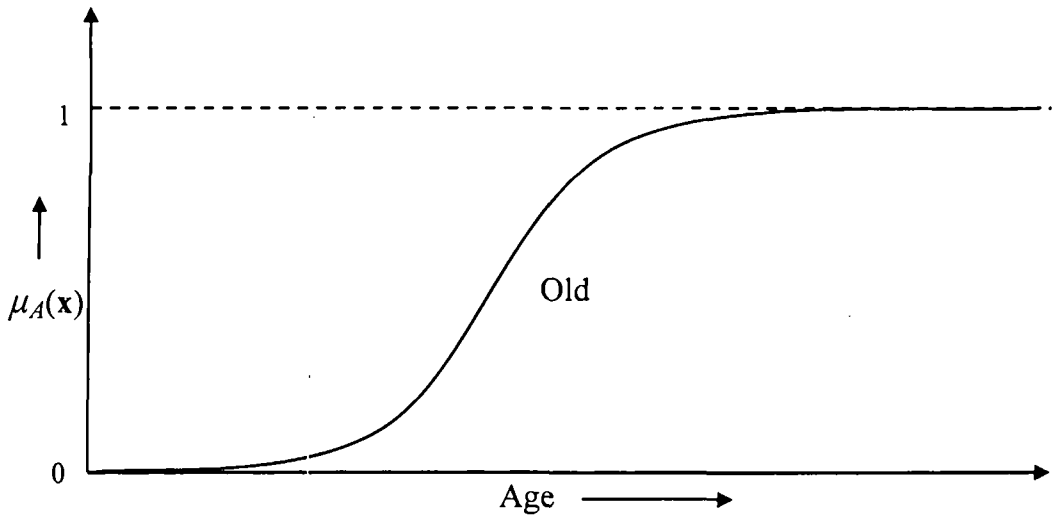


Figure 3.1: Fuzzy set representation of an old person

Figure 3.1 illustrates the use of fuzzy sets to represent the notion of an old person and also depicts the granulation of age. The number, $\mu_A(\mathbf{x})$ represents the degree of membership of $\mathbf{x}$ in the fuzzy set A. A membership function to calculate $\mu_A(\mathbf{x})$ is of the form:

$$\mu_A : X \rightarrow [0,1]$$

According to Zadeh, fuzzy logic is used in two different senses [6] [46] namely *broad* and *narrow*. Fuzzy logic in the *broad* sense serves mainly as an apparatus for fuzzy control and analysis of vagueness in several application domains. In this sense it is one of the major techniques of soft computing. Its use in this sense offers tolerance to imprecision (vagueness) and suboptimality, so that quick, simple and sufficiently good solutions are obtained for problems involving real-life ambiguous situations. Fuzzy logic in the *narrow* sense is a logical system that aims at a formalization of approximate reasoning. In this sense fuzzy logic may be viewed as a generalization of

the traditional multivalued logical systems, e.g., Lukasiewicz's logic. However, its agenda remains quite different from the traditional logic systems because the concept of use of a logic system for approximate reasoning is not a part of traditional multivalued logic systems.

Fuzzy sets and fuzzy logic constitute the oldest and most reported soft computing paradigm [47]. Fuzzy logic has been applied to many fields, from control theory to artificial intelligence. Research on application of fuzzy logic and fuzzy set theory in the field of supervised pattern recognition was first reported in the year 1966 in the seminal note of Bellman *et al.* [48]. After that fuzzy logic has been increasingly used to improve many conventional methods for pattern recognition. An example of this is the introduction of fuzzy logic based clustering algorithms like the popular Fuzzy c-means clustering algorithm. Experiments on the application of fuzzy logic based pattern recognition on numeric and image data is presented in chapter 4. The experiments employ the Fuzzy c-means clustering algorithm.

3.3 Artificial Neural Networks

An Artificial Neural Network is a computational system that tries to mimic the learning process of biological neurons in order to achieve human-like ability of information processing. They offer a remarkable ability to derive meaning from complicated or imprecise data. Thus, they can be used to extract patterns and detect trends that are too complex to be noticed by other computer techniques. Artificial neural networks are made up of a set of simple processing units called *artificial neurons* assembled in the form of a closely interconnected network which offer a surprisingly rich structure in exhibiting some features of the biological neural network [49]. The concept of artificial neurons was first proposed in 1943 by Warren McCulloch [50].

Figure 3.2 shows the McCulloch-Pitts model of an artificial neuron. In this model the activation value y is given by a weighted sum of its d input values x_i and a bias term θ. The output signal s is typically a nonlinear function (like binary function, ramp function or the sigmoid function) $f(y)$ of the activation value y.

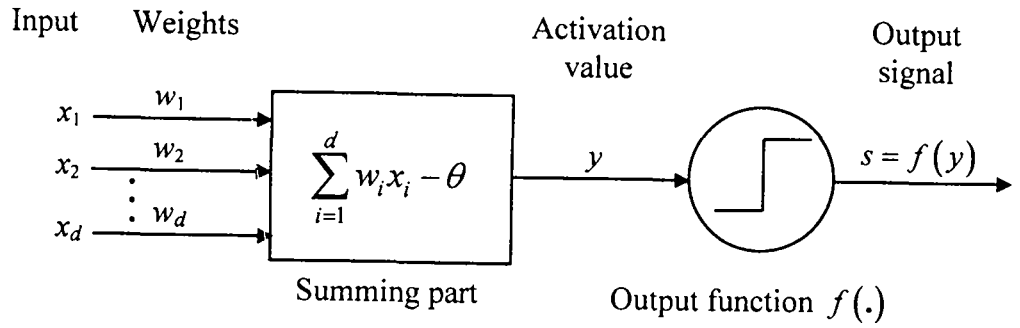


Figure 3.2: A McCulloch-Pitts model of an artificial neuron

The following equations describe the operation of a McCulloch-Pitts model of an artificial neuron:

$$\text{Activation:} \qquad y = \sum_{i=1}^{d} w_i x_i - \theta$$

$$\text{Output signal:} \qquad s = f(y)$$

Like human beings, ANNs learn by example. Thus an ANN can be configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true for ANNs as well. In case of an ANN the interconnections are assigned weights which are a measure to reflect the strength of the connection between the units and the amount of information flow between the connected units. ANNs can learn using both supervised and unsupervised learning paradigms.

A trained neural network can be thought of as an 'expert' in processing the category of information it is given to analyze. This expert can then be used to provide projections, given new situations of interest and to answer 'what if' questions. Other advantages of ANNs include:

(i) Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

(ii) Self organization: An ANN can create its own organization or representation of the information it receives during learning time.

(iii) Real time operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

(iv) Fault tolerance via redundant information coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

The most common type of ANNs consists of three groups or layers of processing units: *input layer*, *hidden layer* and *output layer*. The units in input layer are connected to the units in hidden layer, which are again connected to units in the output layer. An example is shown below:

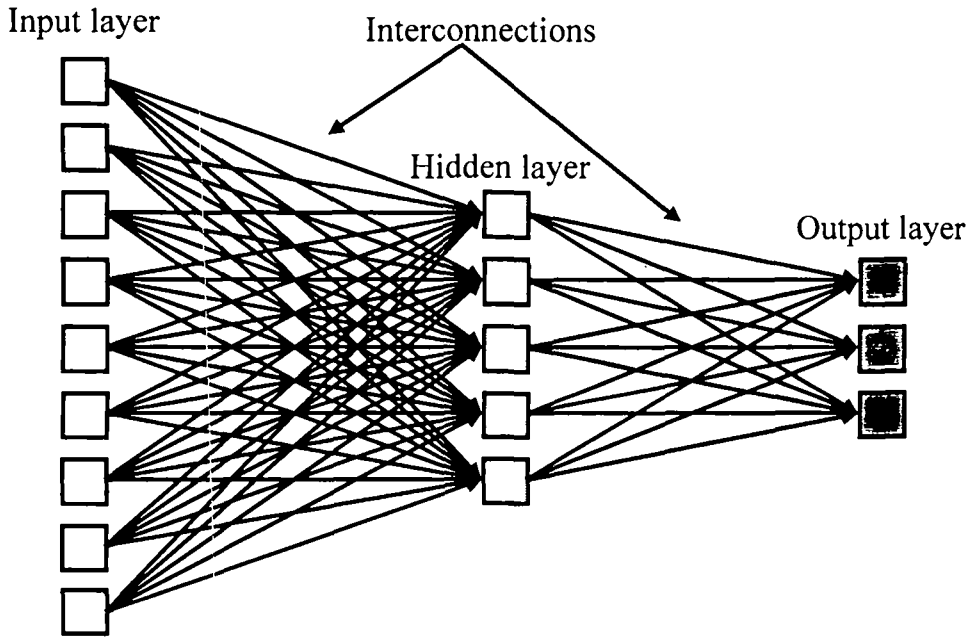


Figure 3.3: An artificial neural network with three layers of processing units

The raw information to be processed by an ANN is fed to the input units. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behaviour of the output units depends on the activity of the hidden units and the weights between the hidden and output units. This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights

between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

Artificial neural networks can have a variety of neural network architectures to accomplish different types of tasks. Some of these architectures include the feedforward (loop-free) ANNs, recurrent ANNs, Radial basis function ANNs, Kohonen self-organizing ANNs, stochastic ANNs, modular ANNs, Physical ANNs, Holographic associative memory, Neuro-fuzzy ANNs etc. Details on these architectures can be obtained in [49] and [51].

3.4 An Experiment on Digit Recognition using ANN

In this section an experiment on the application of a special type of neural network called a Hamming neural network for digit recognition from distorted images of printed and handwritten digits, is presented.

3.4.1 Hamming Network

A Hamming Network is a maximum likelihood classifier that can determine which of several exemplar pattern vectors is most similar to an input pattern vector [52]. The network has a very good ability to recognize noise corrupted patterns [53]. The network is used to solve pattern recognition problems which involve the use of binary vectors for pattern representation, with only two possible values, 0 and 1 for the individual elements of the pattern vectors. The hamming network implements a classifier that can determine which of the several stored exemplar vectors or templates is most similar to an input pattern vector. For doing this the network does a correlation or template matching between the input and the stored templates by calculating the distance between the binary input pattern vectors and the stored templates. Usually this distance measure is the *Hamming distance*. The hamming distance is defined as the number of differing bits between two corresponding, fixed-length input vectors.

The hamming network uses a matrix called the *prototype data matrix* which stores a set of noiseless pattern vectors as templates. The patterns are not learned by the system rather they are stored as exemplar vectors in the prototype data matrix. The objective of the hamming network is to decide which exemplar pattern vector in the prototype matrix is closest to the input vector. For doing this it calculates the similarities between all the vectors of the prototype matrix with the input vector. The input pattern vectors supplied to the network generally come with noise because of distortion by real-world events.

A diagrammatic representation of a hamming network is shown below:

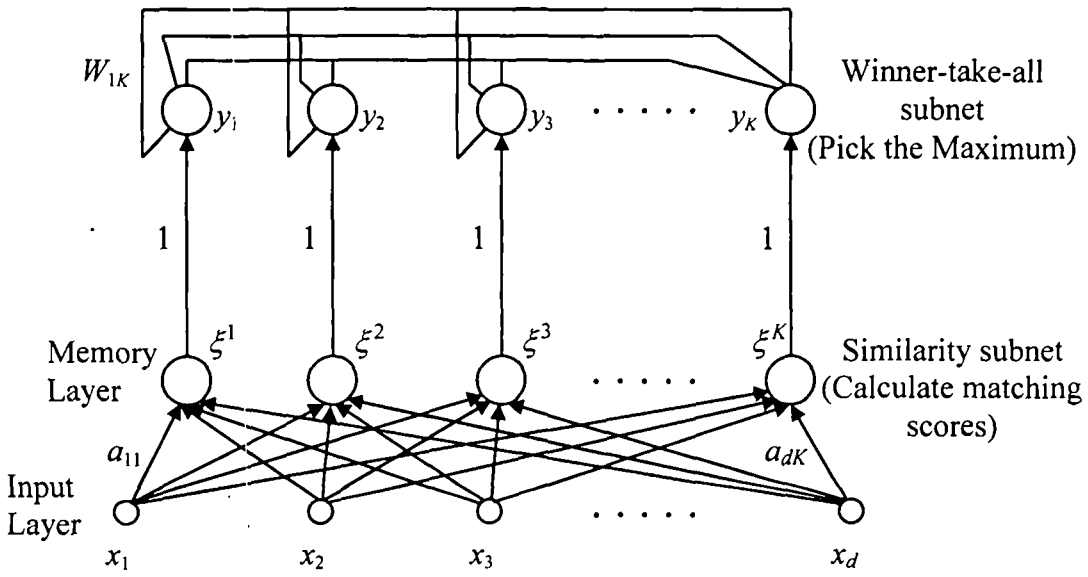


Figure 3.4: A Hamming network. The network input and output are represented by x and y vectors, respectively

As can be seen from Figure 3.3 the hamming network is divided into two parts, the *similarity* subnet and the *winner-take-all* subnet. The similarity subnet comprises of the d-neuron input layer and K-neuron memory layer. Each neuron k in the memory layer is connected to all d input layer neurons. The winner-take-all subnet consists of a fully connected K-neuron topology. An exemplar pattern vector ξ^k is stored in the network by letting the values of the connection weights between memory neuron k and the input layer neurons i (i=1, 2, ..., d) to be:

$$a_{ki} = \xi_i^k$$

The values of the weights W_{ij} in the winner-take-all subnet are chosen in such a way such that the output neurons inhibit each other. That is:

$$W_{ij} = \begin{cases} 1, & for\ i = j \\ -\varepsilon, & for\ i \neq j \end{cases}$$

where $0 < \varepsilon \leq 1/K$. ε is same for all W_{ij}'s for $i \neq j$ and is usually set to $1/K$.

When a binary input pattern vector **x** is presented to the network for classification then the computation is the network takes place in two steps:

Step 1): Each neuron k ($1 \leq k \leq K$) in the memory layer computes its similarity Z_k with the input pattern vector **x** using the following equation:

$$Z_k = \frac{1}{2}\left(\sum_{i=1}^{d} a_{ki} x_i + d\right)$$

Step 2): Each neuron k in the memory layer transfers its similarity value Z_k to the corresponding neuron k in the winner-take-all subnet (the connection weight between a neuron k in the memory layer and the corresponding neuron k in the winner-take-all subnet is set to 1). The winner-take-all subnet then finds the pattern vector i with the maximal similarity. Each neuron k in the winner-take-all subnet sets the initial value $y_k(0) = Z_k/d$ and then computes $y_k(t)$ iteratively (t = 1, 2, ...) by using the following:

$$y_k(t) = \Theta_0\left(\sum_{i=1}^{d} W_{ki} y_i(t-1)\right),$$

where Θ_T is the threshold logic function defined as:

$$\Theta_T(u) = \begin{cases} u, & if\ u \geq T, \\ 0, & otherwise. \end{cases}$$

The two steps are repeated until the activity levels of the neurons in the winner-take-all subnet no longer change and only one neuron k in the winner-take-all subnet remains active, i.e., with a non-zero positive value for y_k. This neuron is declared as the winner and it represents the exemplar pattern vector that is closest to the input patter vector **x**. The input pattern vector **x** is thus classified as the closest

match to the exemplar pattern vector ξ^k corresponding to the winner neuron k [54].

3.4.2 Digit Recognition using a Hamming Network

In the subsection, the experimental results and discussion of an experiment on digit recognition using a hamming network are presented. The experiment was carried out for illustration of a pattern recognition task by neural networks as a part of research on soft computing approaches.

For the experiment, two sets, each consisting of 10, noiseless, monochromatic, 32×48 pixels sized images of the 10 decimal digits, were considered as templates. Let such a set be called here as a *template image set*. A digit in a monochromatic image is represented as a grid of black and white pixels. The two template image sets are shown below:

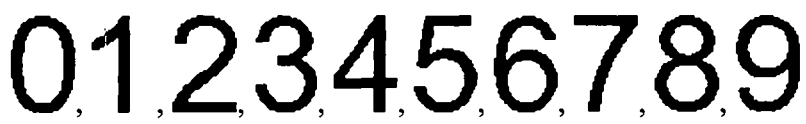


Figure 3.5(a): The first template image set of noiseless monochromatic images of the 10 decimal digits

Figure 3.5(b): The second template image set of noiseless monochromatic images of the 10 decimal digits

The similarity subnet of the hamming network used for the task of classification consists of an input layer with 1536 (32×48) neurons, a memory layer with 10 neurons and a winner-take-all subnet also with 10 neurons (as the memory layer has 10 neurons).

To recognize an image of a digit, after reading it into a 32×48 matrix (comprising of all 0's and 1's), as one of the 10 digits, the matrix is reshaped to a vector of size 1×1536 by concatenating each row of the 32×48 matrix, one after another, in a single row. This input pattern vector of size 1×1536 is then applied to the input layer

of the hamming network and after processing of this input vector (as described in the previous subsection) one of the 10 neurons in the winner-take-all subnet gives an output value of '1', that corresponds to one of the 10 decimal digits.

For testing the hamming network for recognition of digits from noisy images, the following 5 sets of input images of the 10 decimal digits were considered:

Figure 3.6(a): First input image set of distorted monochromatic images for the 10 decimal digits

Figure 3.6(b): Second input image set of distorted monochromatic images for the 10 decimal digits

Figure 3.6(c): Third input image set of distorted monochromatic images for the 10 decimal digits

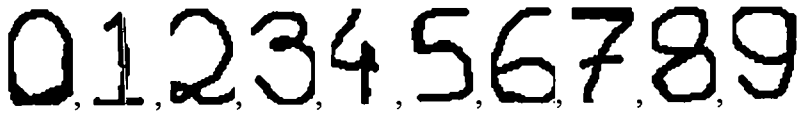


Figure 3.6(d): Fourth input image set of distorted monochromatic images for the 10 decimal digits

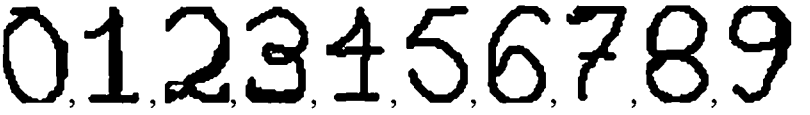


Figure 3.6(e): Fifth input image set of distorted monochromatic images for the 10 decimal digits

The experimental results carried out on the above 5 sets of images are now presented.

The following results were obtained from the hamming network for the 5 input image sets using the first template image set:

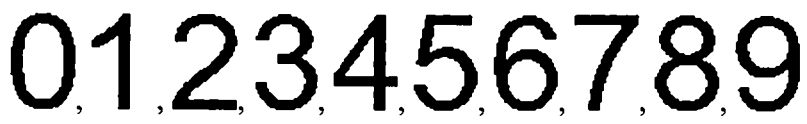


Table 3.1(a): Digit recognition obtained for first input image set using the first template image set

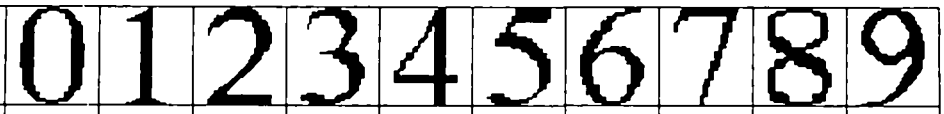
Input Images:										
Identified As:	0	1	2	3	4	5	6	7	8	9

Table 3.1(b): Digit recognition obtained for second input image set using the first template image set. Misclassifications are highlighted in gray colour

Input Images:										
Identified As:	0	1	2	5	9	5	5	7	0	9

Table 3.1(c): Digit recognition obtained for third input image set using the first template image set. Misclassifications are highlighted in gray colour

Input Images:										
Identified As:	0	1	2	3	9	5	5	7	5	9

Table 3.1(d): Digit recognition obtained for fourth input image set using the first template image set. Misclassifications are highlighted in gray colour

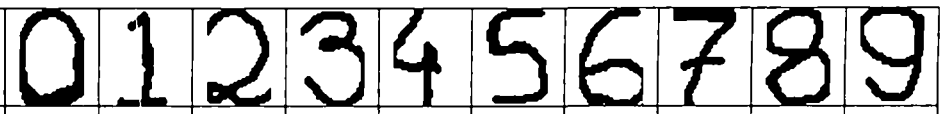
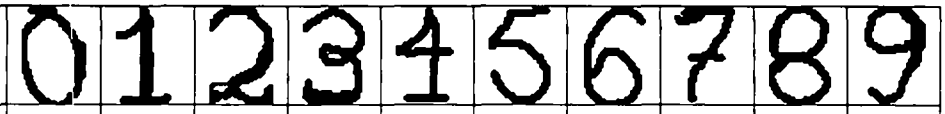
Input Images:										
Identified As:	0	1	2	3	1	5	5	7	8	9

Table 3.1(e): Digit recognition obtained for fifth input image set using the first template image set. Misclassifications are highlighted in gray colour

Input Images:										
Identified As:	0	1	2	3	5	5	8	7	8	8

The following results were obtained from the hamming network for the 5 input image sets using the second template image set:

Table 3.2(a): Digit recognition obtained for first input image set using the second template image set. Misclassifications are highlighted in gray colour

Input Images:										
Identified As:	0	1	2	3	6	3	6	7	8	9

Table 3.2(b): Digit recognition obtained for second input image set using the second template image set

Input Images:										
Identified As:	0	1	2	3	4	5	6	7	8	9

Table 3.2(c): Digit recognition obtained for third input image set using the second template image set

Input Images:										
Identified As:	0	1	2	3	4	5	6	7	8	9

Table 3.2(d): Digit recognition obtained for fourth input image set using the second template image set. Misclassifications are highlighted in gray colour

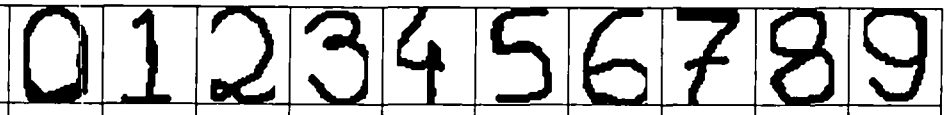
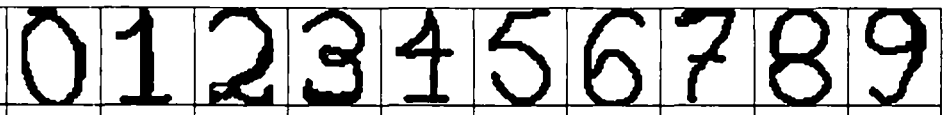
Input Images:										
Identified As:	0	1	0	0	1	5	6	7	6	9

Table 3.2(e): Digit recognition obtained for fifth input image set using the second template image set. Misclassifications are highlighted in gray colour

Input Images:										
Identified As:	0	1	0	8	5	6	6	7	8	9

For the above results it can be seen that out of 50 images from the input image sets 38 images were correctly recognized by the hamming network using the first template image set and for the second template image set, 40 images were correctly recognized. The average percentage of performance obtained is thus $((38/50+40/50)/2)\times 100 = 78$, which is a good performance. However it should also be noted that the classification accuracy of a hamming network is fully dependant on the fact that how well the templates stored by it match the input pattern vectors. For more sophisticated recognition, neural networks with dynamic learning ability should be used.

3.5 Chapter Summary

In this chapter, a discussion on some of the prominent soft computing approaches like fuzzy logic and artificial neural networks is presented. On the part of fuzzy logic, the chapter presented a brief overview of a fuzzy set and its representation. It also presented a discussion on how fuzzy logic can be used to deal with vagueness encountered in natural language statements. The two broad senses of fuzzy logic are

also discussed. On the part of artificial neural networks, the chapter presented discussions on overview of ANNs, concept of an artificial neuron and its computational model, advantages of ANNs and structure of an ANN. Further the chapter also presented a short discussion on hamming neural network based pattern recognition and presented an experiment on the use of hamming network in recognition of digits from distorted monochromatic images of printed and handwritten digits.

CHAPTER 4

Clustering Algorithms and Validity Indices

This chapter presents a brief overview on clustering, similarity measures and different taxonomical representations of clustering. The chapter also presents discussions on some prominent clustering algorithms and some popular clustering validity assessment indices. Further some experiments on the applications of clustering algorithms for pattern recognition on numeric, image and text data are also presented.

4.1 Overview

Clustering is a fundamental technique for pattern recognition and machine learning. It is one of the vital means of dealing with the data that is encountered in our day-to-day life. It is a key technique for discovering the inherent structure of any given data set. Formally, clustering is defined as the unsupervised classification or partitioning of input patterns (observations, data items, data points, or feature vectors) into clusters (classes, groups) such that data items within a cluster are similar in some respect (often based on proximity according to certain defined distance measure) and dissimilar from those in other clusters. Since clustering is unsupervised, the patterns do not come with prior class label information and so the classification is based on inherent statistical structure of the overall collection of input patterns.

Besides the term clustering, there are a number of other terms with similar meanings which include cluster analysis, automatic classification, numerical taxonomy, botryology and typological analysis, that are also in use [55].

Technically stated, the aim of any clustering technique is to obtain a $K \times n$ partition matrix $U(X)$ of a given data set $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ in $\Re^d$ (each point in X is d-dimensional) representing its partitioning into K clusters (C_1, C_2,...,C_K) by optimizing a criterion function defined globally over all of the patterns in the data set. The partition matrix $U(X)$ may be represented as $U = [\mu_{kj}]$, $k = 1,..., K$, and $j = 1,..., n$ where μ_{kj} is the membership degree of a pattern $\mathbf{x}_j$ in cluster C_k. For crisp or hard partitioning of the data, $\mu_{kj} = 1$ if $\mathbf{x}_j \in C_k$, otherwise $\mu_{kj} = 0$ [56].

A graphical representation of clusters and some possible cluster shapes is shown below:

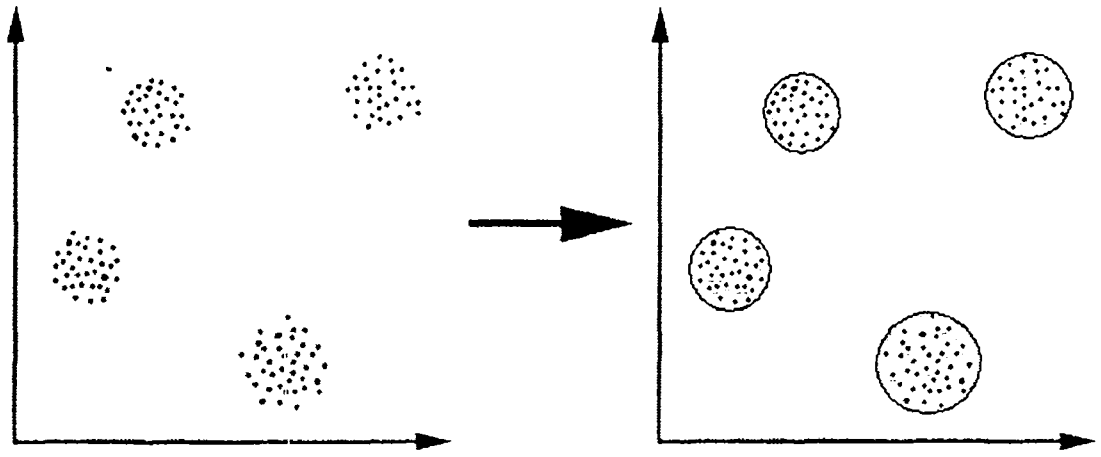
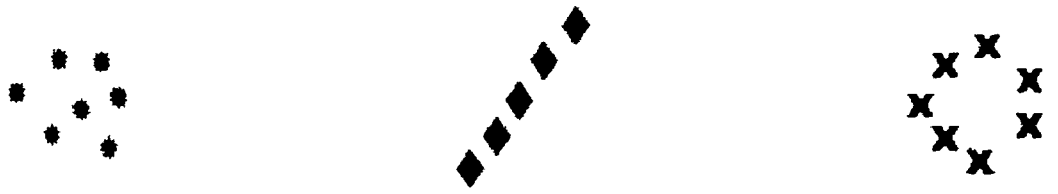


Figure 4.1: Graphical representation of clusters

When depicted graphically, clusters may come in several different shapes, not necessarily circular, as shown below:



Compact Linear Circular

Figure 4.2: Clusters of different shapes

4.2 Proximity Measures

One of the oldest and most influential theoretical assumptions is that the similarity between two patterns is inversely related to psychological distance between the perceptions of the two patterns. Thus an important step in any clustering process is to select a proximity measure that determines how the similarity or dissimilarity of two data points is to be calculated. This influences the shape of the clusters, as some elements may be close to one another according to one proximity measure and farther away according to another. It is thus a most common practice to calculate the dissimilarity using a distance measure defined on the feature space of patterns. Some common *proximity measures* [57] [58] (also called *similarity measures* or *distance measures*) used in literature are described below:

(i) The Euclidean distance: It is the most common distance measure used and preferred by researchers in the field of clustering as it takes the magnitude of the data into account. It is also called 2-norm distance. The Euclidean distance is a true metric, and is defined as:

$$D(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

(ii) The Manhattan distance: The Manhattan distance is also known by the names city-block distance or 1-norm distance. It is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. It is defined as:

$$D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d}|x_i - y_i|$$

(iii) The Minkowski distance: The Minkowski distance of order p between the two points $\mathbf{x}$ and $\mathbf{y}$ is defined as:

$$D(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d}|x_i - y_i|^p\right)^{1/p}$$

It is the generalization of both the Euclidean distance and the Manhattan distance. When $p = 2$, the Euclidean distance is obtained, and when $p = 1$ the Manhattan distance is obtained. Minkowski distance is a popular

distance measure for higher dimensional data.

In the above three cases **x** and **y** denote two d-dimensional points such that $\mathbf{x} = (x_1, x_2, ..., x_d)$ and $\mathbf{y} = (y_1, y_2, ..., y_d)$ and $D(\mathbf{x}, \mathbf{y})$ denotes the distance between them.

(iv) The cosine distance: The cosine distance gives the angular cosine distance between two vectors of d dimensions. Given two vectors P and Q the cosine distance, $\cos(\theta)$, is defined as:

$$\cos(\theta) = \frac{P.Q}{\|P\| \|Q\|}$$

The cosine distance is a popular distance measure for clustering a collection of documents represented using the vector space model [35].

Other than the above mentioned proximity measures, many other proximity measures are also found in literature like the Mahalanobis distance and the Hamming distance. A detailed discussion on proximity measures can be found in [58].

4.3 Taxonomy of Clustering

Data clustering can be done using a number of different approaches. Several taxonometric representations of clustering methodology are possible [3] [59] [60] which are described below:

(i) Agglomerative vs. Divisive: An agglomerative (also sometimes called hierarchical) approach begins with each pattern in a distinct (singleton) cluster, and successively merges clusters together until a stopping criterion is satisfied. In the end, a hierarchy of clusters is found. A divisive (also called partitional) method begins with all patterns in a single cluster and performs splitting of patterns into mutually exclusive (hard) or overlapping (fuzzy) clusters until a stopping criterion is met. The partitional clustering algorithms usually adopt an iterative optimization paradigm. It starts with an initial partitioning and requires several iterations to obtain the final

partitioning by optimizing the criterion function. Along with this, in all iterations, any partitional clustering algorithm also computes a representative point, called a centroid or center, using different measures of central tendency (mean, median, mode etc) of all points in the cluster, for all the clusters obtained thus far. The centroids are used a representatives of the clusters obtained.

(ii) Monothetic vs. Polythetic: Monothetic methods use single-feature-based assignment to divide objects into clusters. Polythetic algorithms consider multiple-feature-based assignment. Most algorithms are polythetic; that is, all features enter into the computation of distances between patterns, and decisions are based on those distances. A monothetic algorithm considers features sequentially to divide the given collection of patterns. An example of a monothetic clustering algorithm is presented by Anderberg in [61].

(iii) Hard vs. Fuzzy: A non-fuzzy, crisp or hard clustering algorithm assigns each pattern to exactly one cluster during its operation and in its output. A fuzzy clustering method assigns degrees of membership in several clusters to each input pattern; thus a pattern can belong to more than one cluster. A fuzzy clustering algorithm can be converted to a hard clustering algorithm by assigning each pattern to the cluster for which the pattern has the largest degree of membership.

(iv) Deterministic vs. Stochastic: Deterministic clustering methods always arrive at the same clustering for a given a data set. Stochastic clustering methods employ stochastic elements (e.g., random numbers) to find a good clustering and may not necessarily obtain the same clustering for a given a data set, always.

(v) Incremental vs. Non-incremental: Non-incremental clustering methods mainly rely on having the whole data set ready before applying the algorithm. A hierarchical agglomerative clustering algorithm belongs to this class. Incremental clustering algorithms work by assigning objects to their respective clusters as they are encountered.

4.4 K-means Algorithm

The K-means clustering algorithm is the simplest and most commonly used clustering algorithm that usually employs the sum of squared error (*SSE*) criterion or objective function [62]. It is hard, partitional, non-deterministic, polythetic, non-incremental clustering algorithm that partitions a data set X of n points, into a pre-specified number (K) of mutually exclusive clusters (C_1, C_2,.....,C_K) by optimizing the *SSE* criterion function defined globally over all of the patterns in the data set. The data set X may be represented as a matrix $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots\ldots, \mathbf{x}_n)$ in $\Re^d$ where each row of X, $\mathbf{x}_i$, represents a d-dimensional pattern.

Some important points in K-means clustering algorithm are:

1. None of the clusters is empty; $C_k \neq \emptyset$ for any k.
2. Every sample belongs to a single cluster; $C_i \cap C_j = \emptyset$, $\forall\, i \neq j$.
3. The centroids of clusters are to be found in such a way that they are, as much as possible, far away from each other.

The sum of squared error (*SSE*) criterion function is defined as:

$$J = \sum_{j=1}^{n} \sum_{k=1}^{K} \mu_{kj} \left\| \mathbf{x}_j - \mathbf{z}_k \right\|^2$$

where, $\left\| \mathbf{x}_j - \mathbf{z}_k \right\|^2$ is a chosen distance measure between a data point $\mathbf{x}_j$ in cluster C_k and its cluster centroid $\mathbf{z}_k$. $\mu_{kj} = 1$ if $\mathbf{x}_j \in C_k$, otherwise $\mu_{kj} = 0$. The index J is an indicator of the distance of the n data points from their respective cluster centroids.

The K-means algorithm works as follows:

Step 1): Randomly choose a user specified number of K initial centroids (also called seeds) inside the multidimensional feature space containing the pattern set.

Step 2): Assign each data point to the nearest centroid.

Step 3): Re-calculate the cluster centroids by taking the mean (thus the name) of all points in each cluster.

Step 4): Repeat the process until the K centroids no longer change or negligible changes occur in two successive steps.

The K-means clustering algorithm aims to minimize the value of the *SSE* objective function in each iteration. The algorithm stops when the values of this objective function, and as a result the centroids do not change or negligible changes occur in two successive iterations.

Advantages of K-means algorithm:

1) It is very easy to implement, and has been adapted to many problem domains.
2) Its time complexity is $O(nKdt)$, where n is the number of patterns, K is the number of clusters, d is the number of dimensions in a pattern and t is the number of iterations. Since K, d, and t are usually much less than n, the time complexity of K-means is approximately linear. Therefore, K-means is a good selection for clustering medium and large-sized data sets [63].
3) It does not require the user to specify too many parameters.
4) The algorithm is deterministic after the initial centroids are determined.

Based on the use of different measures of central tendency, the K-means clustering algorithm has three other popular hard clustering derivatives also, namely K-medoids, K-medians and K-modes. This is because the K-means algorithm requires the concept of a mean to be definable for all kinds of data set but which is not always the case [5]. Also for some data sets a mean is more influenced by outliers (*data points lying far apart from all other points in the feature space*) or other extreme values. The derivative K-medoids method can be used in such cases. Compared to means, medoids are less sensitive to such points. The medoids are the nearest objects to the mean of data in one cluster. Calculation of medoids is however costlier than calculation of the straightforward means, as after the calculation of the means, the points in a cluster again need to be checked for their vicinity from their mean. Use of median results in the K-medians method, where the median or 'middle value' is taken for each ordered attribute. Alternatively, in the K-modes method, the most frequent value for each attribute is used. All methods require the user to specify K, the number of clusters.

4.5 Fuzzy c-means Algorithm

The Fuzzy c-means clustering algorithm was proposed by Dunn [64] and was later extended by Bezdek [65] as a fuzzy version of the K-means algorithm. Thus the Fuzzy c-means (FCM) algorithm is sometimes also called as fuzzy K-means algorithm. The algorithm uses the soft computing technique, fuzzy logic (discussed in chapter 3) to obtain a fuzzy partitioning of the input data set X. The algorithm has been the dominant approach in both theoretical and practical applications of fuzzy techniques to unsupervised pattern classification or recognition [65] [66].

In traditional hard clustering approaches the partitioning of input data set X is done in a mutually exclusive manner, thus each pattern belongs to one and only one cluster. The clustering obtained in hard clustering is hence disjoint. Fuzzy clustering extends this notion and associates each pattern with every cluster using a membership function. The design of the membership function is the most important problem in fuzzy clustering. The aim of a fuzzy clustering algorithm is to obtain a fuzzy *pseudopartition*. Hard clustering can be obtained from a fuzzy pseudopartition by thresholding the membership degrees.

A fuzzy pseudopartition or fuzzy c-partition of the data set X of n points is a family of fuzzy subsets of X, denoted by $\mathcal{P} = \{A_1, A_2, ..., A_c\}$, which satisfies

$$\sum_{k=1}^{c} A_k(\mathbf{x}_j) = 1,$$

for all $j \in \{1, 2, ..., n\}$ and,

$$0 < \sum_{j=1}^{n} A_k(\mathbf{x}_j) < n,$$

for all $k \in \{1, 2, ..., c\}$, where c is the number of clusters (a positive number) and $\mathbf{x}_j$ is the j^{th} data point or pattern in X. Comparing $\mathcal{P}$ with $U(X)$, it can be easily deduced that:

$$A_k = \{\mu_{k1}, \mu_{k2}, ..., \mu_{kn}\}.$$

The FCM algorithm uses the fuzzy extension of the *SSE* criterion function as the objective function. It is defined as: $J_m(\mathcal{P}) = \sum_{j=1}^{n}\sum_{k=1}^{c}(\mu_{kj})^m \left\|\mathbf{x}_j - \mathbf{z}_k\right\|^2$, where $\left\|\mathbf{x}_j - \mathbf{z}_k\right\|^2$

represents the distance between $\mathbf{x}_j$ and $\mathbf{z}_k$ (the k^{th} cluster centroid). Like, K-means, in the case of FCM also the distance calculation is usually done using the Euclidean distance. The performance index $J_m(\mathcal{P})$ measures the weighted sum of distances between clusters and points in the corresponding fuzzy clusters. Thus, the smaller the value of $J_m(\mathcal{P})$ the better the clustering is. Therefore, the goal of the FCM clustering algorithm is to find a fuzzy pseudopartition $\mathcal{P}$ that minimizes the value of $J_m(\mathcal{P})$.

The FCM algorithm is now presented below. The algorithm is based on the assumption that the desired number of clusters c is given and, in addition, a particular distance measure, a real number $m \in (1,\infty)$, and a small positive number ε, serving as a stopping criterion, are also chosen.

Step 1): Let $t = 0$. Select an initial partition matrix $U^{(0)}$ of the original data set using a random generator.

Step 2): Calculate the c cluster centroids $\mathbf{z}_1^{(t)}, \mathbf{z}_2^{(t)}, ..., \mathbf{z}_k^{(t)}$ for $U^{(t)}$, for a chosen value of m.

$$\mathbf{z}_k^{(t)} = \frac{\sum_{j=1}^{n} (\mu_{kj})^m \mathbf{x}_j}{\sum_{j=1}^{n} (\mu_{kj})^m}$$

for all $k \in \{1,2,...,c\}$

Step 3): Update $U^{(t+1)}$ by the following procedure: For each $\mathbf{x}_j \in X$, if $\left\|\mathbf{x}_j - \mathbf{z}_k^{(t)}\right\|^2 > 0$ for all $k \in \{1,2,...,c\}$, then define

$$\mu_{kj}^{(t+1)} = \left[\sum_{i=1}^{c} \left(\frac{\left\|\mathbf{x}_j - \mathbf{z}_k^{(t)}\right\|^2}{\left\|\mathbf{x}_j - \mathbf{z}_i^{(t)}\right\|^2} \right)^{\frac{1}{m-1}} \right]^{-1};$$

if $\left\|\mathbf{x}_j - \mathbf{z}_k^{(t)}\right\|^2 = 0$ for some $k \in I \subseteq \{1,2,...,c\}$, then define $\mu_{kj}^{(t+1)}$ for $k \in I$ by any non-negative real numbers satisfying:

$$\sum_{k \in I} \mu_{kj}^{(t+1)} = 1,$$

and define $\mu_{kj}^{(t+1)} = 0$ for all $k \in \{1,2,...,c\} - I$.

Step 4): Compare $U^{(t)}$ and $U^{(t+1)}$. If $\left|U^{(t+1)} - U^{(t)}\right| \leq \varepsilon$, then stop; otherwise, increase t by one and return to Step 2.

$\left|U^{(t+1)} - U^{(t)}\right|$ denotes a distance between the fuzzy pseudopartitions $U^{(t+1)}$ and $U^{(t)}$ in the space $\Re^{n \times c}$. An example of this distance is

$$\left|U^{(t+1)} - U^{(t)}\right| = \max_{1 \leq k \leq c, 1 \leq j \leq n} \left|\mu_{kj}^{(t+1)} - \mu_{kj}^{(t)}\right|.$$

The parameter m is selected according to the problem under consideration. When $m \to 1$, the membership degrees converge to 0 or 1 and the algorithm converges to the crisp K-means algorithm. When $m \to \infty$, all clusters tend towards the centroid of the data set X. That is, the partition becomes fuzzier with increasing m. Presently, there is no theoretical basis for an optimal choice for the value of m. However, it is established that the algorithm converges for any $m \in (1,\infty)$ [14].

The advantages of the FCM algorithm are similar to those of its hard counterpart i.e., K-means and the time complexity of FCM is also reported to be similar to that of K-means [67]. Further, the FCM algorithm is more suitable than K-means for clustering fuzzy data sets, i.e., data sets whose clusters could not be distinctly separated in a mutually exclusive manner, easily.

4.6 Validity Indices

In most clustering algorithms, the number of clusters K is specified by the user. Once K is specified any typical clustering algorithm always tries to find the best fit partitioning (hard or fuzzy) for the specified number of clusters. However this does not mean that even the best fit is meaningful at all. Either the number of clusters might be wrong or the cluster shapes obtained might not correspond to the actual groups in the data if the data can be grouped in a meaningful way at all. Further in many cases a prior knowledge of the actual number of clusters is not available and the optimal number of clusters in a data set cannot be easily and intuitively estimated beforehand and requires evaluation of some metric for its determination. Thus the

determination of the optimal number of clusters in a given pattern set is also an important problem in cluster analysis [68] [69] [70]. Moreover, since clustering algorithms are unsupervised, irrespective of the clustering method (hard or fuzzy) the final partitions of data do require some kind of validation in most applications. This validation is done by evaluating some goodness measures called validity assessment indices or validity indices for short.

The validity indices should be such that they should be able to impose an ordering of the clusters in terms of their goodness. In other words, if $U_1,U_2,...,U_m$ be m partitions of X, and the corresponding values of a validity index be $V_1,V_2,...,V_m$, then $V_{k1} \geq V_{k2} \geq ... \geq V_{km}$ will indicate that $U_{k1} \uparrow U_{k2} \uparrow ... \uparrow U_{km}$, for some permutation $k1$, $k2$,..., km of $\{1, 2,...,m\}$. Here '$U_i \uparrow U_j$' indicates that partition U_i is a better clustering than U_j [56].

There are two criteria [71] [72] for evaluation and selection of the optimal clustering from a given set of clustering results of a clustering algorithm:

1) *Compactness*: Members of each cluster should be as close to each other as possible.
2) *Separation*: The clusters should be widely spaced from each other.

A good clustering obtained using a clustering algorithm should have both high compactness and high separation. Further the validity indices should be so defined that they should focus on the checking and measurement of these two criteria for every clustering they evaluate.

Various validity indices have been proposed to quantitatively evaluate the correctness and goodness of a clustering structure to solve the cluster validity problem. Once an appropriate cluster validity index is selected or defined, the general solution to the cluster validity problem is to execute a clustering algorithm for all possible numbers of clusters in sequence and evaluate the cluster validity index value for each clustering structure obtained and select the clustering structure with the best or optimal validity value.

Some representative validity indices for hard and fuzzy clustering are now presented in the following sections. A thorough survey of validity indices can be found in [72].

4.6.1 Validity Indices for Hard Clustering

In this subsection, some of the popular validity indices, used for evaluating the goodness of partitioning obtained by any hard clustering algorithm, are discussed.

***Pakhira Bandyopadhyay Maulik (PBM) Index*:**

This index was proposed by Pakhira *et al.* [68]. The validity index is the square of product of three quantities, $1/K$, E_1/E_K and D_K. Mathematically it is defined as under:

$$PBM(K) = \left[\frac{1}{K} \times \frac{E_1}{E_K} \times D_K \right]^2$$

where, K is the number of clusters. The factor E_1 is the sum of the distances of each pattern $\mathbf{x}_j$ from the geometric center of all patterns, w_0. This factor is independent of the number of clusters and is computed as:

$$E_1 = \sum_{j=1}^{n} \left\| \mathbf{x}_j - w_0 \right\|$$

The factor E_K is the sum of within cluster distances of K clusters, weighted by the corresponding membership degree μ_{kj}:

$$E_K = \sum_{k=1}^{K} \sum_{j=1}^{n} \mu_{kj} \left\| \mathbf{x}_j - \mathbf{z}_k \right\|$$

where, $\mathbf{z}_k$'s are the centroids of the corresponding clusters C_k.

The factor D_K measures the maximum separation between two clusters over all possible pairs of clusters:

$$D_K = \max_{1 \le i \le K, 1 \le j \le K} \left\| \mathbf{z}_i - \mathbf{z}_j \right\|$$

The maximum value of $PBM(K)$ across a hierarchy of m partitions indicates the best partitioning and ensures the formation of a small number of compact clusters with large separation between at least two clusters.

***Xie-Beni (XB) Index*:**

The Xie and Beni index was proposed by Xuan-Li Xie and Gerardo Beni in the year 1991 [73]. The index focuses on the compactness and separation of clusters of

patterns. The index was originally proposed for validation of fuzzy clustering but it can also used to validate hard clustering results. The index is defined by:

$$XB(K) = \frac{\sum_{k=1}^{K}\sum_{j=1}^{n}(\mu_{kj})^m \left\|\mathbf{x}_j - \mathbf{z}_k\right\|^2}{n \times \min_{1\le i\le K, 1\le j\le K}\left\|\mathbf{z}_i - \mathbf{z}_j\right\|^2}$$

Comparing the index with the objective function of the FCM clustering algorithm

$$J_m(\mathcal{P}) = \sum_{j=1}^{n}\sum_{k=1}^{K}(\mu_{kj})^m \left\|\mathbf{x}_j - \mathbf{z}_k\right\|^2$$

it can be inferred that

$$XB(K) = \frac{J_m(\mathcal{P})}{n \times \min_{1\le i\le K, 1\le j\le K}\left\|\mathbf{z}_i - \mathbf{z}_j\right\|^2}$$

where, $J_m(\mathcal{P})$ gives the measure of compactness of clusters and $\min_{1\le i\le K, 1\le j\le K}\left\|\mathbf{z}_i - \mathbf{z}_j\right\|^2$ gives the measure of separation between different cluster centers. The best partitioning across a hierarchy of partitions is indicated by the minimum value of $XB(K)$. The parameter m is called as *fuzzifier* and it is usually chosen between 1 and 2.

Dunn's Index (DI):

The Dunn's Index was proposed by J. C. Dunn [74] in the year 1974 as a validity index for the identification of compact and well separated clusters. The index is thus not suitable for use with fuzzy clusters and is usually used to validate the partitioning obtained by a hard clustering algorithm.

The *DI* index is defined as:

$$DI(K) = \min_{1\le i\le K}\left\{\min_{1\le j\le K, i\ne j}\left\{\frac{\min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} D(\mathbf{x},\mathbf{y})}{\max_{1\le k\le K}\left\{\max_{\mathbf{x},\mathbf{y}\in C_k} D(\mathbf{x},\mathbf{y})\right\}}\right\}\right\}$$

The compactness measure of a cluster C_k is given by $\max_{\mathbf{x},\mathbf{y}\in C_k} D(\mathbf{x},\mathbf{y})$ and the separation measure between two clusters C_i and C_j is given by $\min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} D(\mathbf{x},\mathbf{y})$. $D(\mathbf{x},\mathbf{y})$ denotes

the distance between **x** and **y** as per some chosen distance norm.

The best partitioning across a hierarchy of partitions is indicated by the maximum value of $DI(K)$.

The Dunn's Index suffers from the following problems:

- It is computationally very expensive.
- It is sensitive to the presence of noise.

Separation and Compactness (SC) index:

It is the ratio of the sum of compactness and separation of the clusters in a given partitioning. It is defined as:

$$SC(K) = \sum_{k=1}^{K} \frac{\sum_{j=1}^{n} (\mu_{kj})^m \left\| \mathbf{x}_j - \mathbf{z}_k \right\|^2}{n_k \times \sum_{i=1}^{K} \left\| \mathbf{z}_i - \mathbf{z}_k \right\|^2}$$

where, m is usually chosen between 1 and 2.

From the above equation it can be observed that the SC index can also be viewed as a sum of individual cluster validity indices normalized through division by the cardinality n_k (the number of points in a cluster C_k) of each cluster [75].

The SC index is useful when comparing different partitions having equal number of clusters. A lower value of $SC(K)$ indicates a better partitioning across a hierarchy of partitions. The SC index is also called Partition Index.

Davies-Bouldin (DB) Index:

This index [76] is a function of the ratio of the sum of within-cluster scatter to between-cluster separation. The scatter within the i^{th} cluster is computed as $S_{i,q} = \left(\frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \left\{ \left\| \mathbf{x} - \mathbf{z}_i \right\|^q \right\} \right)^{1/q}$ and the distance between cluster C_i and C_j, denoted by d_{ij}, is defined as $d_{ij,t} = \left\{ \sum_{s=1}^{d} \left| \mathbf{z}_{is} - \mathbf{z}_{js} \right|^t \right\}^{1/t}$. The Davies-Bouldin ($DB$) index is then defined as:

$$DB(K) = \frac{1}{K}\sum_{i=1}^{K} \max_{j, j \neq i} \left\{ \frac{S_{i,q} + S_{j,q}}{d_{ij,t}} \right\}$$

The values of both t and q are usually chosen as 2 for Euclidean distance. A lower value of $DB(K)$ indicates a better partitioning across a hierarchy of partitions.

4.6.2 Validity Indices for Fuzzy Clustering

This subsection presents a discussion on some popularly used validity indices for evaluating the goodness of partitioning obtained by the fuzzy clustering algorithms.

The Fuzzy PBM (PBMF) Index:

The fuzzy version of the *PBM* index called *PBMF* index is obtained by incorporating fuzzy distances. It was also introduced by Pakhira *et al.* [68]. It is defined as follows:

$$PBMF(c) = \left[\frac{1}{c} \times \frac{E_1}{J_m(U,Z)} \times D_c \right]^2$$

where,

$$J_m(U,Z) = \sum_{j=1}^{n}\sum_{k=1}^{K} (\mu_{kj})^m \left\| \mathbf{x}_j - \mathbf{z}_k \right\|$$

U is the partition matrix and Z is the set of cluster centroids. $m \in (1, \infty)$.

Like the hard version, i.e., the *PBM* index, in this case also the best pseudopartitioning across a hierarchy of fuzzy pseudopartitions is determined by the maximum value of the *PBMF* index. It should be noted that D_c is same as D_K, only the notation is different; as in case of fuzzy clustering it is customary to use 'c' to denote the number of clusters, instead of K, which is used in case of hard clustering.

The Xie-Beni (XB) Index:

As already mentioned in subsection 4.6.1, the Xie-Beni Index *XB* was originally proposed for validating the results of fuzzy clustering. The performance of the index heavily depends on the value of the fuzzifier m. For m = 1.5, the index is given by:

$$XB(c) = \frac{\sum_{k=1}^{c}\sum_{j=1}^{n}(\mu_{kj})^{1.5}\left\|\mathbf{x}_j - \mathbf{z}_k\right\|^2}{n \times \min_{1\le i\le c, 1\le j\le c}\left\|\mathbf{z}_i - \mathbf{z}_j\right\|^2}$$

The number of clusters is denoted by c instead of K. Like the hard version the best fuzzy pseudopartitioning across a hierarchy of fuzzy pseudopartitions is indicated by the minimum value of $XB(c)$. The Xie-Beni index XB, has some drawbacks [77]:

- It monotonically decreases when the number of clusters gets close to n.
- The behaviour of XB index becomes unpredictable for very low or high value of the fuzzifier m.

The Partition Coefficient (PC) Index:

This validity index was the first validity index associated with the FCM clustering algorithm. The PC index was proposed by Bezdek [78] as follows:

$$PC(c) = \frac{1}{n}\sum_{k=1}^{c}\sum_{j=1}^{n}(\mu_{kj})^m$$

where, $1/c \le PC(c) \le 1$. The closer to unity the value of PC index is the 'crisper' is the clustering. If all the membership degree values to a fuzzy pseudopartition are equal, i.e., $\mu_{kj} = 1/c$, the PC index obtains its lowest value. Thus a value close to $1/c$ indicates that there is no clustering tendency in the data set or the clustering algorithm failed to reveal it. The best fuzzy pseudopartitioning (or the optimal value of c) is indicated by the maximum value of the PC index. The index provides compact clusters with higher values of μ_{kj}. There are some drawbacks of the PC index:

- The index is monotonically dependant on the number of clusters c.
- The index is sensitive to the value of fuzzifier m.
- The index has no direct connection to any property of the data set.

The Monotonic Partition Coefficient (MPC) Index:

The MPC validity index was proposed by Dave [79] as a modification over the PC index in order to reduce the monotonic dependency of the PC index on c.

The index is defined as:

$$MPC(c) = 1 - \frac{c}{c-1}\left(1 - PC(c)\right),$$

where, $0 \le MPC(c) \le 1$. Like the *PC* index, the optimal pseudopartitioning (or optimal value of c) is obtained by maximizing the value of the *MPC* index.

The Classification Entropy (CE) Index:

The Classification Entropy index *CE* measures the fuzziness in the partition matrix *U*, which is similar to the Partition Coefficient. It was proposed by Bezdek [80]. It is defined as:

$$CE(c) = -\frac{1}{n}\sum_{k=1}^{c}\sum_{j=1}^{n}(\mu_{kj}).\log_a(\mu_{kj}), \text{ where, } a \text{ is the base of the logarithm.}$$

The *CE* index is computed for values of c greater than 1 and the values of the index are obtained in the range $\left[0, \log_a c\right]$. The closer the value of the index is to 0, the harder is the clustering. Thus the optimal number of clusters is indicated by the minimum value. Like the *PC* index, the *CE* index also suffers from the drawbacks of monotonous dependency on the number of clusters c and of lack of direct connection to any property of the data set, because both the indices do not use the data set itself.

4.7 Experiments

In this section some experiments of pattern recognition using clustering techniques on numeric, image and text data, are presented. As clustering techniques, the K-means and FCM clustering algorithms were used.

4.7.1 Clustering of Numeric Data

In this subsection, the experimental results of clustering of numeric data, obtained using the K-means and FCM clustering algorithms, are presented. For the validation of clustering results obtained by K-means, the validity indices *PBM*, *XB*, *DI*, *SC*, *DB*

(discussed in subsection 4.6.1) were used, and for the validation of clustering results obtained by FCM, the validity indices *PBMF*, *XB*, *PC*, *MPC*, *CE* (discussed in subsection 4.6.2) were used. To illustrate the wide applicability of the clustering algorithms they were tested on a number of benchmark numeric data sets obtained from the famous UCI Machine Learning Repository [81] and other sources. The data sets are now described next.

Benchmark real-life data sets from UCI Machine Learning Repository

Iris data set: This is a very well known data set about three species of the Iris flower, namely, Iris Setosa, Iris Versicolor, and Iris Virginica [82]. This data set is in 4-dimensional space, and has 3 clusters. The total number of observations is 150 with the 3 clusters having 50 instances each. The four features are: *sepal length* in cm, *sepal width* in cm, *petal length* in cm, *petal width* in cm. Out of the 3 clusters, 2 clusters are overlapping (Iris Versicolor, and Iris Virginica), and hence the classification of the data set into the optimal number of clusters (i.e., 3) is a difficult problem with this data set. The data set has often been used as a standard for testing clustering algorithms [77].

Cancer data set: This is the original Breast Cancer data obtained from the University of Wisconsin Hospitals. This data set is in 9-dimensional space, and has 2 classes: benign, malignant. The benign class has 458 observations whereas the malignant class has 241 observations. The two clusters are known to be linearly inseparable i.e. overlapping. The total number of observations is 699 (as of 15 July 1992). The nine features are: *clump thickness*, *uniformity of cell size*, *uniformity of cell shape*, *marginal adhesion*, *single epithelial cell size*, *bare nuclei*, *bland chromatin*, *normal nucleoli*, and *mitoses*.

Wine data set: This is the wine recognition data obtained from a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents, namely, *Alcohol*, *Malic acid*, *Ash*, *Alkalinity of ash*, *Magnesium*, *Total phenols*, *Flavanoids*, *Nonflavanoid phenols*,

Proanthocyanins, *Color intensity*, *Hue*, *OD280/OD315 of diluted wines*, and *Proline*; found in each of the three types of wines. The data set is thus in 13-dimensional space. It has 3 clusters. The total number of observations is 178. The first class has 59 observations, the second class has 71 observations and the third class has 48 observations. The three clusters are not linearly separable i.e. the clusters are overlapping.

Other real-life data sets

SODAR1 Data Set: This data set is in 2-dimensional space. The total number of observations is 90 and has 3 clusters with 30 observations each. Each point describes the convective boundary layer height return of backscattered SODAR signal (in meters), measured at the interval of every 2 minutes in the time period from 1200 hours to 1500 hours [83].

SODAR2 Data Set: This data is in 2-dimensional space. The total number of observations is 90 and has 3 clusters with 30 observations each. Each point describes the temperature inversion layer with flat top height return of backscattered SODAR signal (in meters), measured at the interval of every 2 minutes in the time period from 2300 hours to 0200 hours [83].

Other numeric data sets

Data_3_2: This data is in 2-dimensional space and has 3 clusters. The total number of points is 76 [84] [85] [86]. The data set is not provided with any class labels.

Data_5_2: This data is in 2-dimensional space, and has 5 clusters. The total number of points is 250 with 50 points in each cluster. It is sometimes also referred to as AD_5_2 [84] [86] [87]. The clusters in this data set are not very well separated.

Data_6_2: This data is in 2-dimensional space, and has 6 clusters. The total number of points is 300 with 50 points in each cluster [84].

Data_9_2: This data is in 2-dimensional space. The total number of points is 900 with 87, 116, 95, 102, 90, 99, 105, 111 and 95 points in 9 different clusters. It is sometimes also referred to as st900_2_9 [86] [88]. The clusters in this data set are quite fuzzy and not very well separated.

Data_10_2: This data is in 2-dimensional space, and has 10 clusters. The total number of points is 500 with 50 points in each cluster. It is sometimes also referred to as AD_10_2 [56] [84] [86] [87]. The data set is not provided with any class labels. Some of the clusters in this data set are not very well separated.

Data_4_3: This data is in 3-dimensional space, and has 4 clusters. The total number of points is 400 with 100 points in each cluster. It is sometimes also referred to as AD_4_3 [84] [86] [87].

Ruspini Data: The famous Ruspini data set is in 2-dimensional space and it has 4 different clusters. The total number of points is 75 [89]. The data set is not provided with any class labels.

The general format of each line of the data files, for all the above data sets, is:

feature1, feature2, feature3, ..., class label

Each line in a data file represents one pattern or feature vector of the data set where feature1, feature2, feature3 denote the values of different features of the pattern and the optional class label provides the class information of the pattern. Data files of data sets Data_3_2, Data_5_2, Data_6_2, Data_9_2, Data_10_2 and Data_4_3 also contain additional information about the number of observations, number of dimensions and the number of clusters in the very first line. So before applying any clustering algorithm on them, they were preprocessed by the removal of this additional first line. These three pieces of deleted information can be easily obtained from the actual feature vectors and class labels. The data files of Data_3_2, Data_10_2 and Ruspini data sets do not contain any class labels.

The graphical representations of all the numeric data sets considered here for experimentation are now shown next in Figures 4.3 through 4.14.

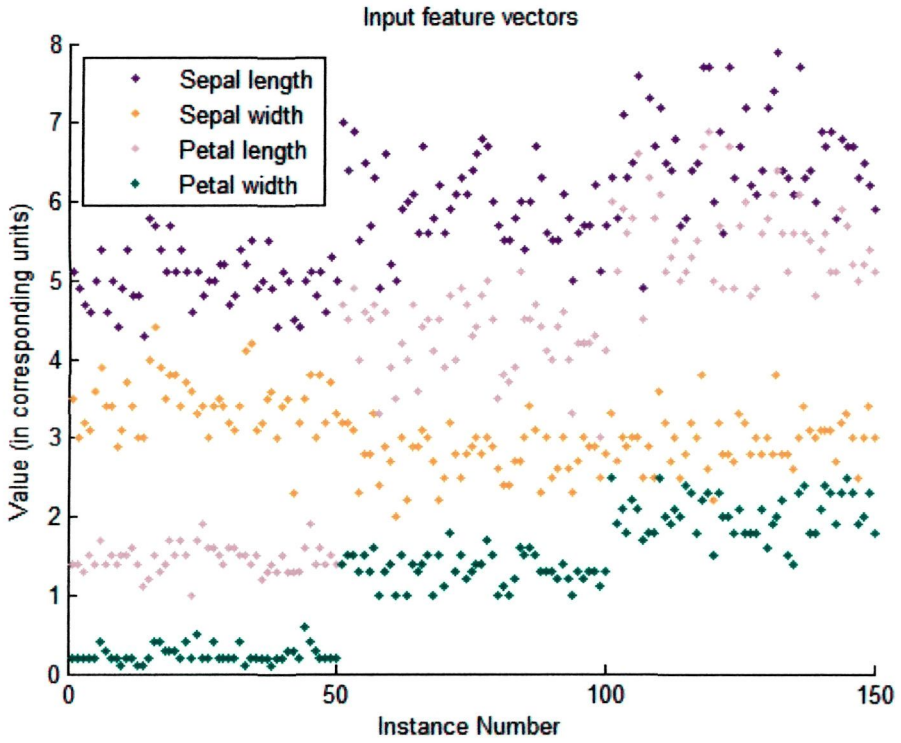


Figure 4.3: Instance number vs. input feature vector values' plot for the Iris data set. $K = 3$, $n = 150$, $d = 4$

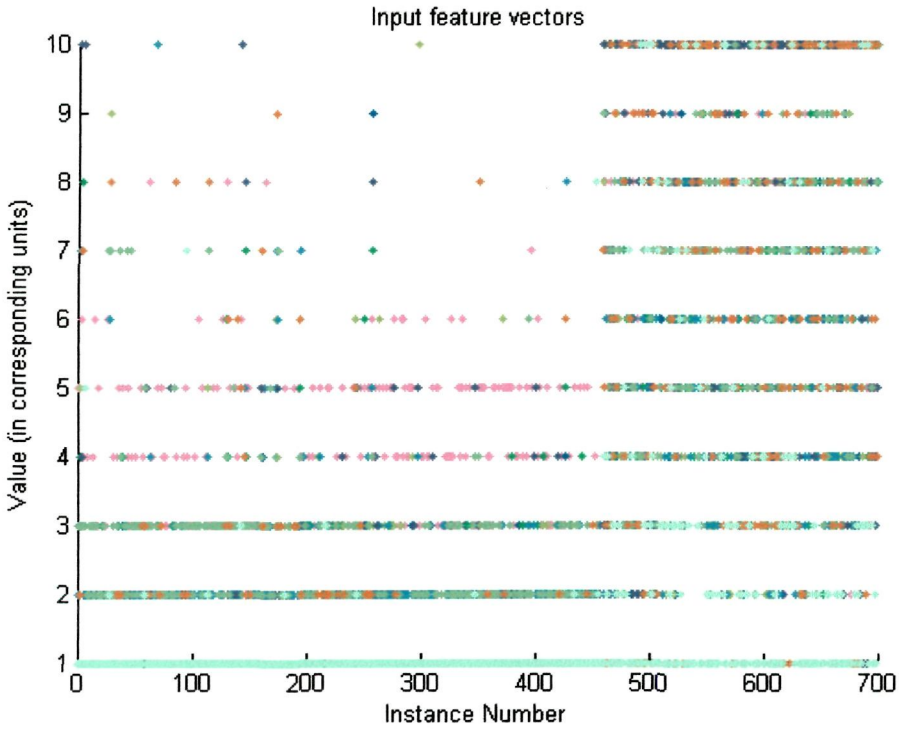


Figure 4.4: Instance number vs. input feature vector values' plot for the Cancer data set. $K = 2$, $n = 699$, $d = 9$

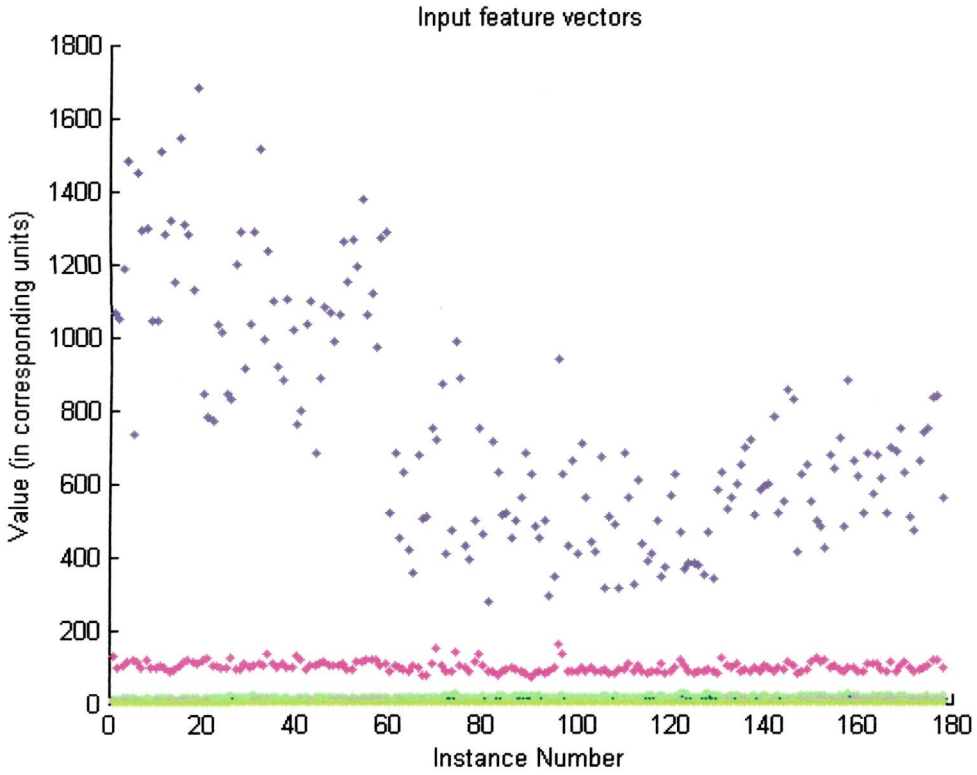


Figure 4.5: Instance number vs. input feature vector values' plot for the Wine data set. $K = 3$, $n = 178$, $d = 13$

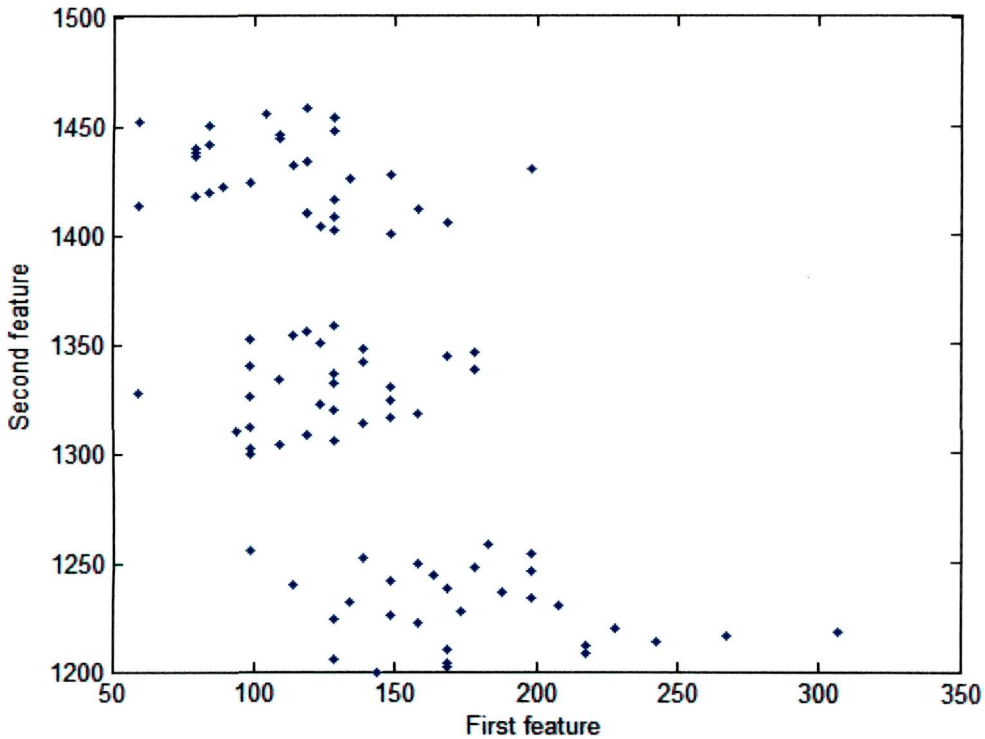


Figure 4.6: Scatter plot for SODAR1 data set. $K = 3$, $n = 90$, $d = 2$

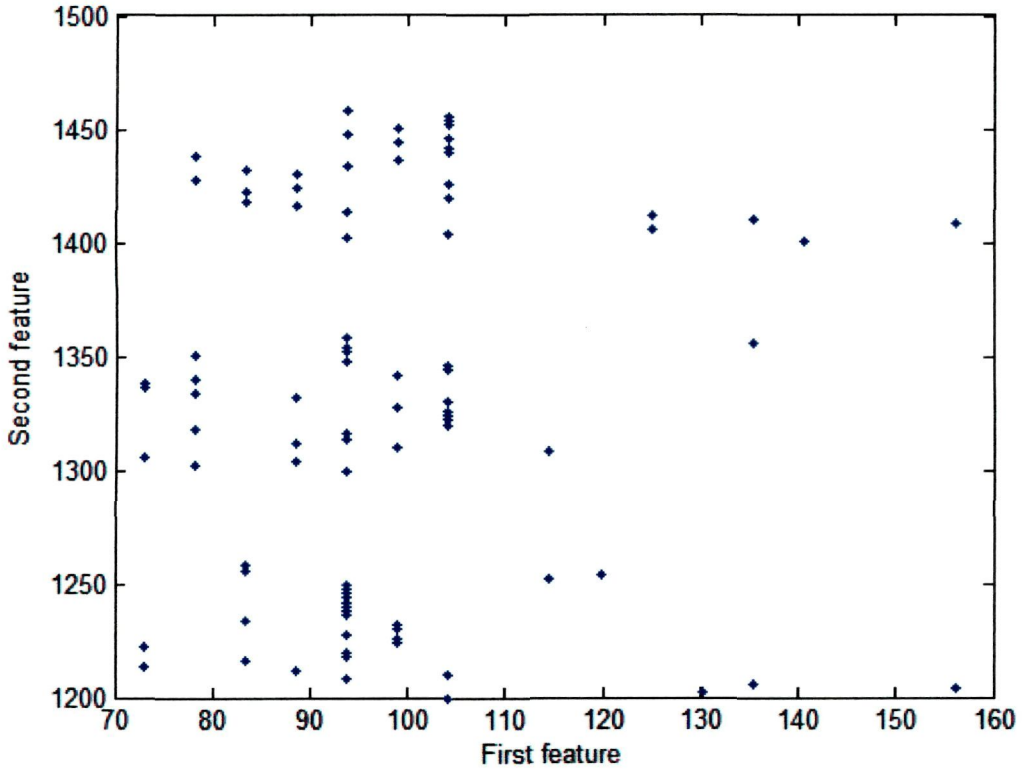


Figure 4.7: Scatter plot for SODAR2 data set. $K = 3$, $n = 90$, $d = 2$

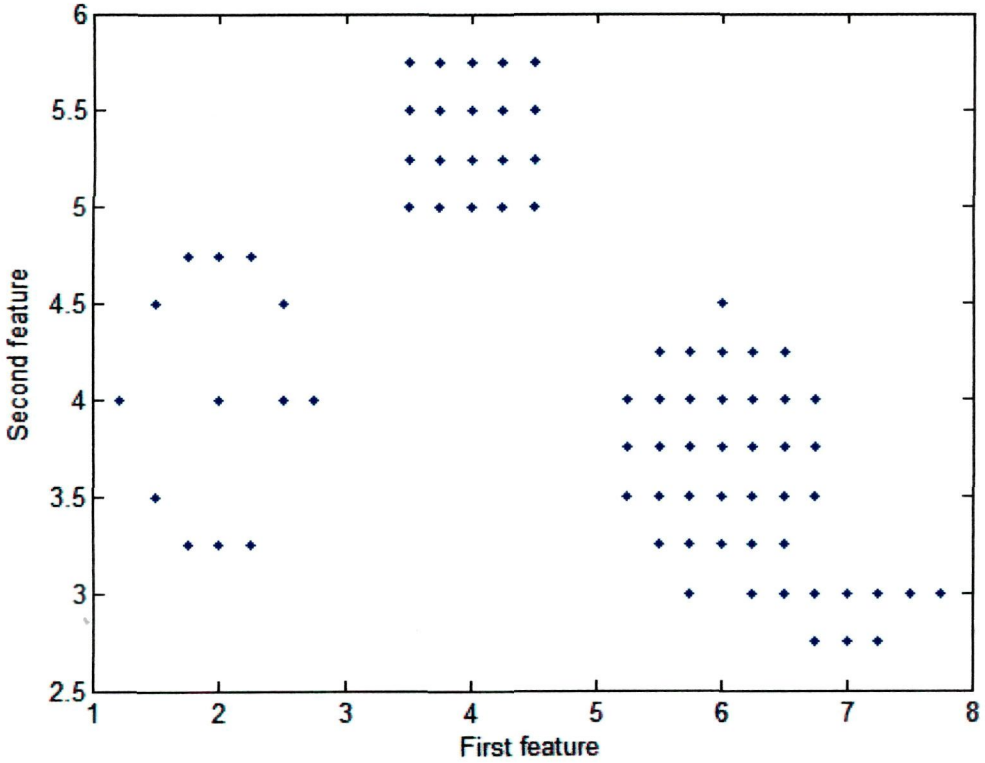


Figure 4.8: Scatter plot for Data_3_2 data set. $K = 3$, $n = 76$, $d = 2$

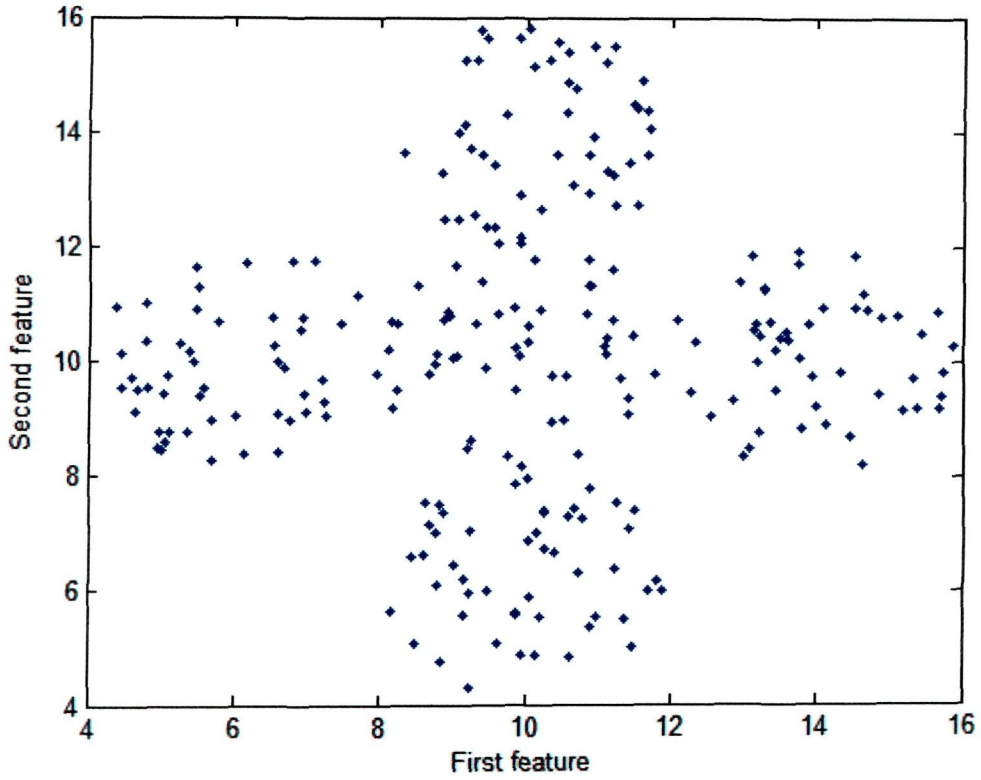


Figure 4.9: Scatter plot for Data_5_2 data set. $K = 5$, $n = 250$, $d = 2$

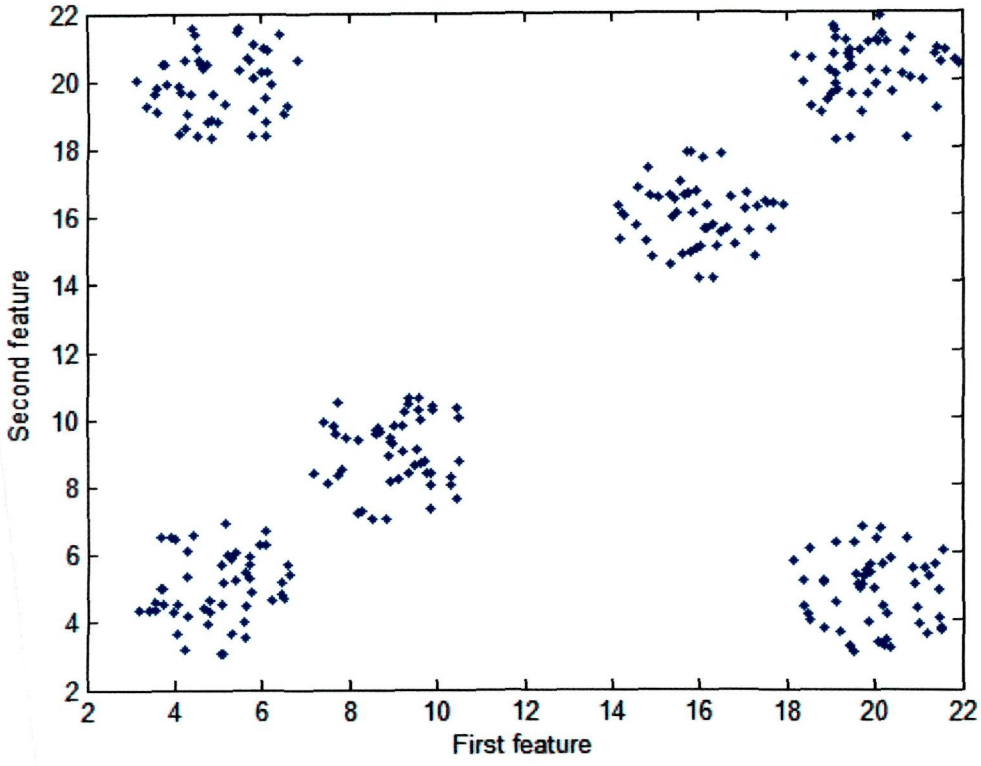


Figure 4.10: Scatter plot for Data_6_2 data set. $K = 6$, $n = 300$, $d = 2$

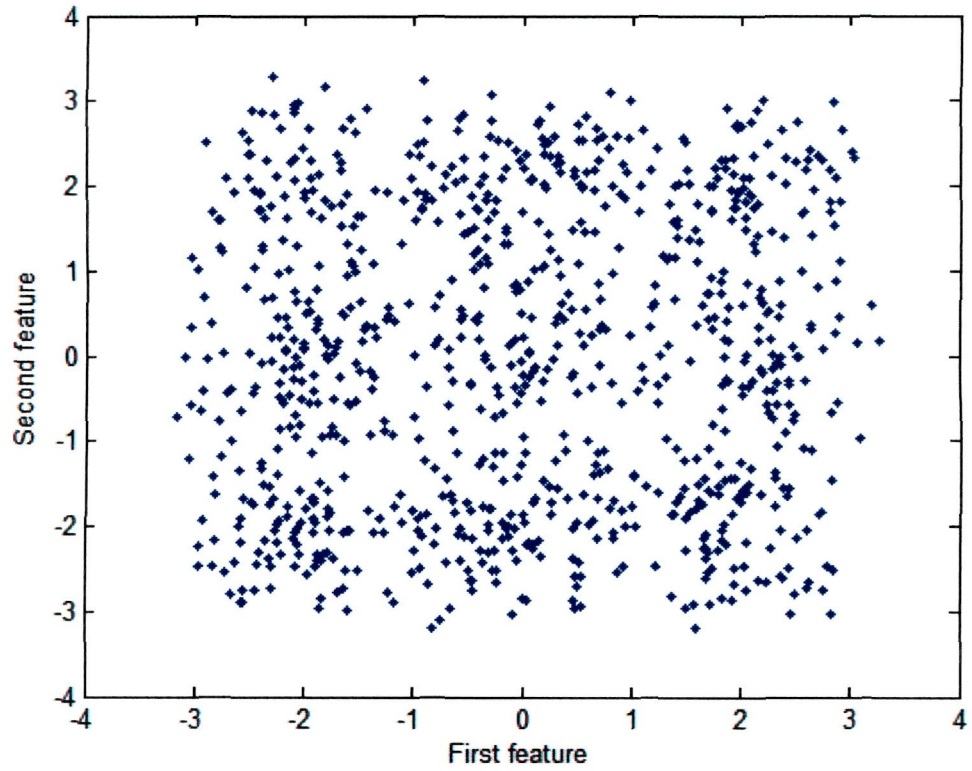


Figure 4.11: Scatter plot for Data_9_2 data set. $K = 9$, $n = 900$, $d = 2$

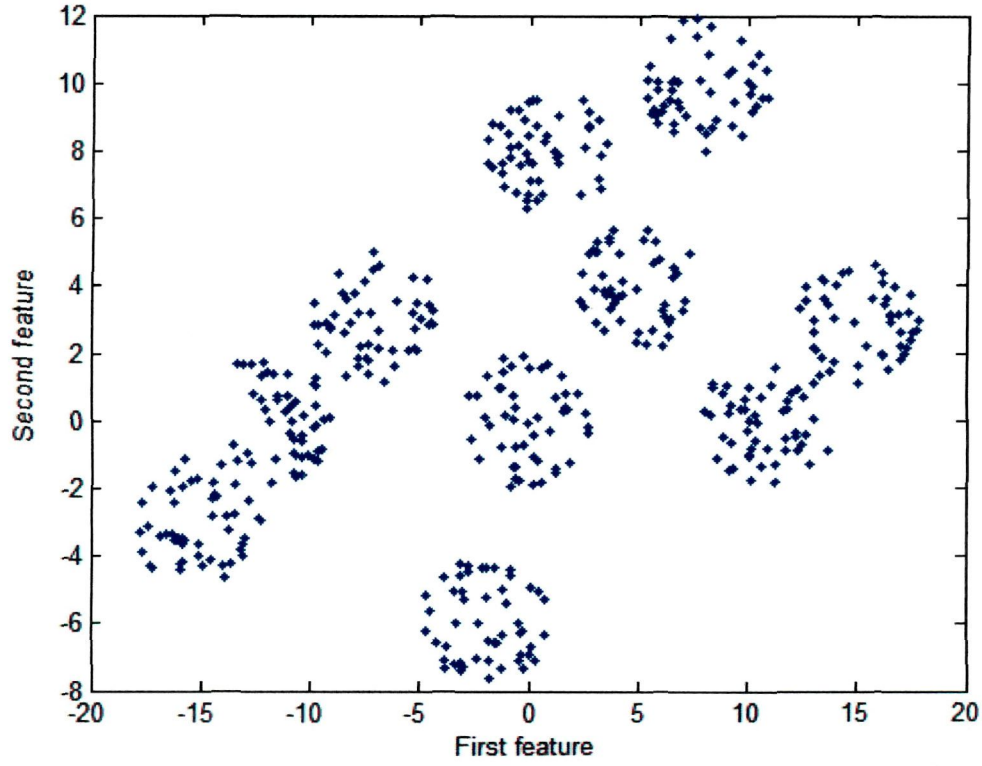


Figure 4.12: Scatter plot for Data_10_2 data set. $K = 10$, $n = 500$, $d = 2$

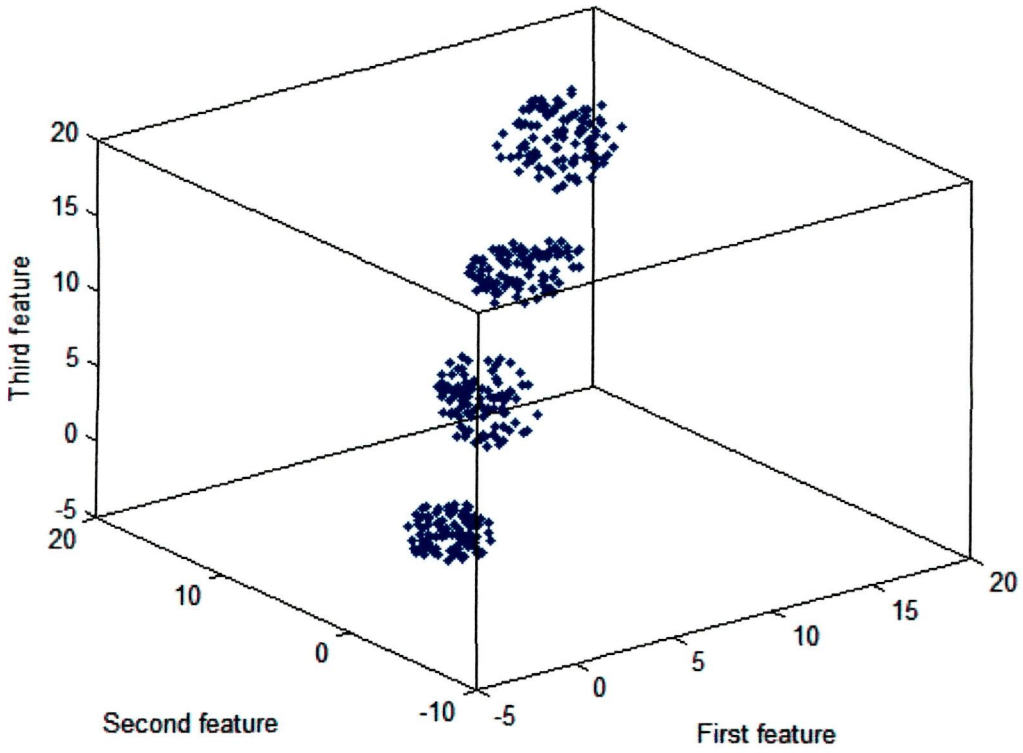


Figure 4.13: Scatter plot for Data_4_3 data set. $K = 4$, $n = 400$, $d = 3$

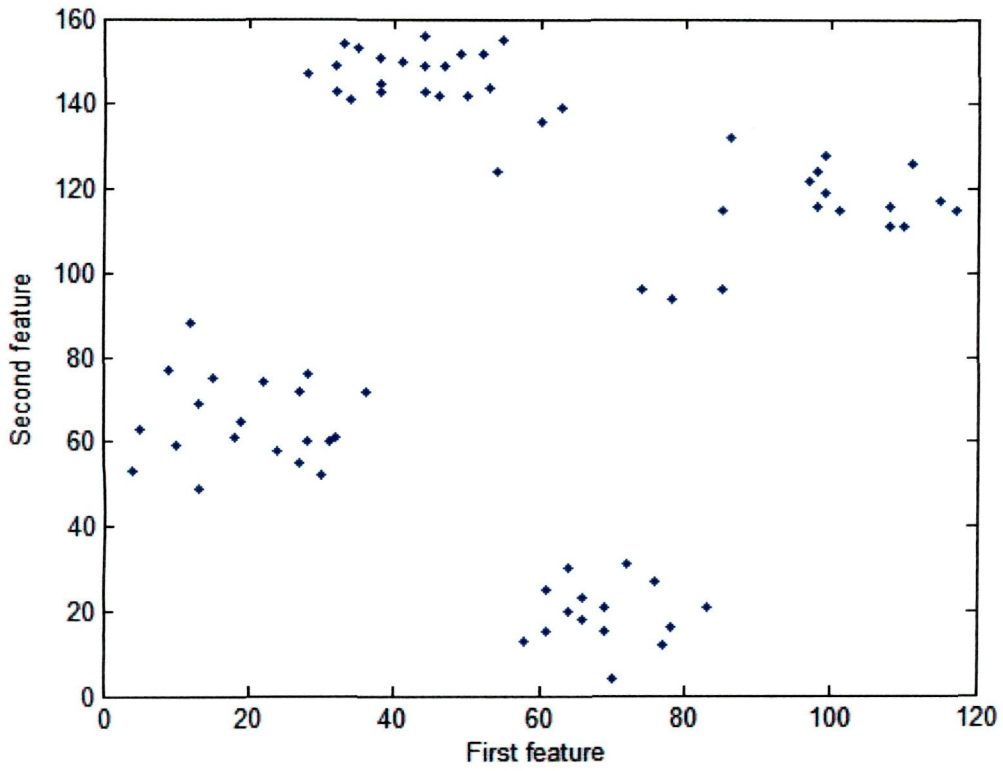


Figure 4.14: Scatter plot for Ruspini data set. $K = 4$, $n = 75$, $d = 2$

Some of the experimental results obtained by the application of the K-means and FCM clustering algorithms on the considered data sets are now presented next in Tables 4.1 - 4.13. For obtaining the results presented in Table 4.1 and Table 4.2, the two algorithms were executed on all the data sets for 50 trials (because of their non-deterministic nature), using the 'Single Clustering' mode (discussed in chapter 8) of the software package PatternWiz. Table 4.1 evaluates the two algorithms based on the similarity of cluster sizes obtained by them compared to the original cluster sizes, and based on the misclassification error percentages in their clustering results. For this evaluation the data sets Iris, Cancer, Wine, SODAR1, Data_5_2 and Data_9_2 were chosen because the clusters in these data sets are not so well separated, i.e. the clusters are overlapping and fuzzy, and so the chances of misclassification error in clustering are more. Moreover for all these data sets, the information about the number of

Table 4.1: Best cluster sizes and lowest misclassification error percentages attained by the K-means and FCM clustering algorithms for the Iris, Cancer, Wine, SODAR1, Data_5_2 and Data_9_2 data sets. For FCM algorithm $m = 1.5$

Data set	Actual no. of clusters	Actual cluster sizes	Best cluster sizes obtained by		Lowest misclassification error percentage	
			K-means	FCM	K-means	FCM
Iris	3	(50, 50, 50)	(38, 50, 62)	(39, 50, 61)	10.6667	11.3333
Cancer	2	(241, 458)	(234, 465)	(231, 468)	4.1488	4.5780
Wine	3	(48, 59, 71)	(47, 62, 69)	(46, 63, 69)	52.2472	52.8090
SODAR1	3	(30, 30, 30)	(29, 30, 31)	(29, 30, 31)	1.1111	1.1111
Data_5_2	5	(50, 50, 50, 50, 50)	(48, 49, 50, 51, 52)	(43, 48, 50, 53, 56)	1.6000	4.8000
Data_9_2	9	(87, 90, 95, 95, 99, 102, 105, 111, 116)	(83, 93, 93, 95, 95, 103, 105, 114, 119)	(84, 92, 92, 95, 96, 104, 106, 113, 118)	7.7778	7.5556

clusters, the sizes of individual clusters in each data set and the class labels of each observation in each data set, are also available. The values in Table 4.1 are the best values obtained from the 50 trials. From Table 4.1 it may seem that the performance of K-means algorithm is better than the FCM algorithm. However, it should be noted that the FCM algorithm is very much dependant on the value of the fuzzifier m which affects the fuzzy belongingness of the observations in the different clusters. As already mentioned in section 4.5, there is no theoretical basis for an optimal choice for the value of m, and the choice of a good value of m is dependant upon the problem under consideration. Thus in the case of Iris, Cancer and SODAR1 data sets, when the value of m was chosen as 12.5, the corresponding cluster sizes and misclassification error percentages obtained were (49, 50, 51) and 7.3333 %; (238, 461) and 3.5765 %; and (30, 30, 30) and 0 % respectively, which are much better than those for K-means. Thus the selection of a good value of m is a tricky task for the FCM algorithm. In general, the value of m is chosen in the interval (1, 2.5] [90] [77]. As such, in this thesis, for all experiments on FCM, m was taken as 1.5.

In Table 4.2 the numbers of iterations and average execution times of the first best trials with optimal clustering (indicated by the lowest value of misclassification error percentage or highest value of the *PBM* index), encountered in a series of 50 trials of each of the algorithms on each of the data sets, are presented. The total execution times of all trials are also presented. The average execution time for a '*first best trial*' is calculated using the following:

$$Average\ execution\ time = \frac{Total\ execution\ time\ of\ all\ trials \times no.of\ iterations\ in\ first\ best\ trial}{Total\ no.of\ iterations\ in\ all\ trials}$$

From Table 4.2 it can be inferred that FCM is computationally costlier than K-means.

Table 4.3 shows the objective function values obtained by K-means and FCM for the Iris and Cancer data sets for varying values for number of clusters (2 to 10). In general, the value of the objective function decreases with increase in the number of clusters. The function has the maximum value when all the features are lumped together into a single cluster and has the minimum value of 0 when the number of clusters equals the total number of observations n [91] [92]. This decreasing trend in the values of the objective function can be easily verified from Table 4.3. It can also be noted that FCM yields comparatively smaller values for the fuzzy *SSE* objective

function compared to the hard *SSE* objective function for K-means.

Table 4.2: Iteration counts and execution times (in seconds) for the K-means and FCM clustering algorithms. For FCM algorithm $m = 1.5$

Data set	No. of clusters	No. of iterations (First best trial)	Average execution time (First best trial)	Total execution time of 50 trials
Algorithm: K-means				
Iris	3	7	0.0014647	0.081602
Cancer	2	6	0.0094869	0.39529
Wine	3	8	0.0031219	0.1518
SODAR1	3	8	0.0018439	0.061308
SODAR2	3	4	0.00094103	0.052462
Data_3_2	3	4	0.00092164	0.05023
Data_5_2	5	8	0.0021571	0.13428
Data_6_2	6	6	0.0027303	0.1397
Data_9_2	9	16	0.016875	1.042
Data_10_2	10	9	0.0064658	0.44327
Data_4_3	4	6	0.002571	0.11527
Ruspini	4	4	0.0011199	0.054594
Algorithm: FCM				
Iris	3	27	0.011445	0.57225
Cancer	2	11	0.014367	0.77189
Wine	3	26	0.013246	0.78153
SODAR1	3	14	0.0047126	0.21005
SODAR2	3	10	0.0036931	0.17542
Data_3_2	3	11	0.0028061	0.28596
Data_5_2	5	25	0.023696	1.3952
Data_6_2	6	11	0.014881	1.6639
Data_9_2	9	38	0.2169	11.8722
Data_10_2	10	27	0.095032	6.1877
Data_4_3	4	9	0.011454	0.81325
Ruspini	4	7	0.0029079	0.25091

Table 4.4 shows a comparison of the actual number of clusters present in the data sets considered, to the number of clusters obtained by the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, using the K-means algorithm. Tables 4.5 through 4.8 provide the actual values of the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, obtained using the K-means clustering algorithm for $K = 2, 3, \ldots, 10$ on all the data sets considered.

Table 4.3: Comparison of objective function values obtained by the K-means and FCM algorithms for the Iris and Cancer data sets for varying number of clusters. For FCM algorithm $m = 1.5$

No. of clusters	Objective function values obtained by K-means		Objective function values obtained by FCM	
	Iris	Cancer	Iris	Cancer
2	152.348	19685.2463	146.2985	18434.8385
3	78.8514	16604.7832	74.3822	14013.8382
4	71.4452	15487.2268	53.736	12021.4487
5	70.8908	13987.1797	45.8452	10597.0034
6	42.4215	13144.7367	38.9158	9487.6806
7	40.5461	12397.126	31.1802	8707.2165
8	36.4972	12901.7124	26.8487	8106.3123
9	31.4269	11591.4448	24.5957	7598.1752
10	33.6003	11678.6978	22.6384	7189.015

Table 4.4: Number of clusters obtained by the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, using the K-means algorithm (entries in bold face indicate the correct determination of number of clusters in a data set by the respective indices)

Data set	Actual no. of clusters	No. of clusters obtained				
		PBM	*XB*	*DI*	*SC*	*DB*
Iris	3	**3**	2	4	9	2
Cancer	2	**2**	**2**	**2**	8	**2**
Wine	3	6	2	4	10	2
SODAR1	3	**3**	**3**	**3**	10	**3**
SODAR2	3	**3**	**3**	**3**	9	**3**
Data_3_2	3	4	**3**	**3**	10	**3**
Data_5_2	5	**5**	4	7	10	4
Data_6_2	6	**6**	4	4	9	4
Data_9_2	9	**9**	**9**	**9**	10	**9**
Data_10_2	10	**10**	**10**	5	**10**	2
Data_4_3	4	**4**	**4**	**4**	9	2
Ruspini	4	**4**	**4**	**4**	10	**4**

Table 4.5: Values of the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, for K = 2, 3, ..., 10 for the Iris, Cancer and Wine data sets using the K-means algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: Iris				
	PBM	***XB***	***DI***	***SC***	***DB***
2	19.9233	**0.0658002**	0.0765063	0.117585	**0.237183**
3	**25.1754**	0.162755	0.0988074	0.0747351	0.384907
4	20.7486	0.231735	**0.100844**	0.0617575	0.555922
5	11.8664	1.18882	0.0373457	0.0338432	0.659995
6	12.4381	0.818217	0.0482243	0.0301706	0.707144
7	10.7152	0.712178	0.0522708	0.0310004	0.736929
8	9.21186	0.63098	0.0739221	0.0359406	0.752915
9	10.3237	0.529041	0.058621	**0.0237124**	0.840559
10	8.7435	0.528155	0.0973585	0.0298985	0.832532
	Data set: Cancer				
2	**142.822**	**0.148954**	**0.155446**	0.394815	**0.40595**
3	103.126	0.436167	0.144338	0.470394	0.946671
4	75.9747	0.384287	0.144905	0.475841	1.01249
5	61.5145	2.11989	0.0512316	0.288958	1.22385
6	45.927	2.29214	0.0558146	0.327916	1.28851
7	35.0487	6.3263	0.0558146	0.254836	1.38493
8	27.6969	6.58208	0.0558146	**0.239468**	1.57122
9	24.3976	4.752	0.0585206	0.261513	1.40588
10	23.1193	4.33901	0.0581238	0.248454	1.48279
	Data set: Wine				
2	325954	**0.0743498**	0.0228207	0.165497	**0.285064**
3	473716	0.182226	0.0162604	0.0798641	0.406442
4	640729	0.124934	**0.0339871**	0.0414293	0.379872
5	677339	0.265377	0.0161491	0.0282448	0.441874
6	**759849**	0.207747	0.0280939	0.0196609	0.449514
7	672397	0.258884	0.0190524	0.015582	0.403681
8	640951	0.456965	0.0216495	0.0122152	0.427851
9	582632	0.667629	0.0185358	0.0111505	0.485289
10	530515	0.886174	0.01341	**0.00906197**	0.479391

Table 4.6: Values of the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, for K = 2, 3, ..., 10 for the SODAR1, SODAR2 and Data_3_2 data sets using the K-means algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: SODAR1				
	PBM	***XB***	***DI***	***SC***	***DB***
2	14038	0.151336	0.0344976	0.302672	0.387663
3	**31105.1**	**0.14322**	**0.112338**	0.109346	**0.36403**
4	27603.7	0.195303	0.0719655	0.0677899	0.468386
5	23446.3	0.490971	0.090693	0.0517513	0.538331
6	21379.6	0.516166	0.0840505	0.0383398	0.554853
7	12565.1	0.609603	0.0648082	0.0499664	0.625845
8	19883.9	0.339092	0.104914	0.0295897	0.645206
9	12909.9	0.603838	0.0880515	0.0353606	0.602753
10	12782	0.992903	0.0880515	**0.0231405**	0.572244
	Data set: SODAR2				
2	16053.6	0.102822	0.0289439	0.205644	0.320648
3	**56229.6**	**0.0569907**	**0.476905**	0.0487374	**0.311482**
4	38931.4	0.54479	0.123007	0.0448373	0.467405
5	26739.6	0.80103	0.0875747	0.0417983	0.579622
6	38563.1	0.401533	0.0905689	0.0210194	0.472651
7	34684.1	0.31038	0.0304776	0.0173961	0.517548
8	21785.8	1.30911	0.0440831	0.0181143	0.574191
9	32497.2	0.475229	0.0850429	**0.0117739**	0.511788
10	30933.5	0.289406	0.0467004	0.0128716	0.568759
	Data set: Data_3_2				
2	9.80666	0.100759	0.333333	0.214972	0.317832
3	16.0331	**0.0832373**	**0.415227**	0.0585122	**0.303267**
4	**18.1554**	0.218963	0.158114	0.0395386	0.412564
5	13.1092	0.707726	0.158114	0.0339857	0.517189
6	13.8253	0.369541	0.158114	0.0281657	0.494736
7	12.4554	0.36538	0.185695	0.021115	0.562219
8	10.0363	0.696626	0.16129	0.0195892	0.657952
9	11.1287	0.327264	0.193746	0.0153449	0.625656
10	10.3158	0.299878	0.193746	**0.014048**	0.640545

Table 4.7: Values of the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, for K = 2, 3, ..., 10 for the Data_5_2, Data_6_2 and Data_9_2 data sets using the K-means algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: Data_5_2				
	PBM	***XB***	***DI***	***SC***	***DB***
2	10.1531	0.341318	0.0154519	0.682636	0.584221
3	8.35199	0.233192	0.0320038	0.290604	0.595959
4	15.4613	**0.13752**	0.0644697	0.126212	**0.499374**
5	**18.7947**	0.137765	0.0259231	0.0870959	0.567185
6	15.2806	0.240247	0.0118684	0.0817806	0.605405
7	13.1366	0.573653	**0.0887616**	0.0653362	0.746764
8	12.5845	0.5203	0.0387182	0.056068	0.69231
9	11.2389	0.432838	0.0670984	0.0515245	0.768607
10	12.8788	0.407901	0.057631	**0.0470591**	0.820815
	Data set: Data_6_2				
2	76.2417	0.285924	0.3318	0.571848	0.534715
3	103.703	0.151669	0.3318	0.142703	0.36152
4	360.856	**0.0434775**	**0.657568**	0.039085	**0.221265**
5	399.975	0.140488	0.219233	0.0219008	0.258787
6	**626.418**	0.0549314	0.515015	0.0117286	0.27691
7	503.456	0.55416	0.09683	0.0111395	0.393821
8	451.743	0.568062	0.0809665	0.00878719	0.470676
9	405.613	0.521981	0.0809665	**0.00779742**	0.539387
10	380.223	0.580432	0.0691405	0.00779945	0.625711
	Data set: Data_9_2				
2	3.95366	0.39126	0.00579991	0.782513	0.625505
3	3.58962	0.240239	0.016931	0.315337	0.619418
4	4.75711	0.180899	0.0115062	0.169923	0.578885
5	4.32354	0.269542	0.0172067	0.144976	0.700189
6	4.50529	0.260109	0.00593429	0.104958	0.730505
7	4.563	0.215312	0.0174149	0.0801909	0.726383
8	4.45299	0.166231	0.0205335	0.0653071	0.606485
9	**4.97797**	**0.144973**	**0.0383366**	0.0514353	**0.560781**
10	4.37909	0.28042	0.0279794	**0.0506244**	0.64241

Table 4.8: Values of the *PBM*, *XB*, *DI*, *SC* and *DB* validity indices, for K = 2, 3, ..., 10 for the Data_10_2, Data_4_3 and Ruspini data sets using the K-means algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: Data_10_2				
	PBM	***XB***	***DI***	***SC***	***DB***
2	155.421	0.159742	0.0270727	0.318775	**0.399191**
3	192.331	0.26049	0.0190878	0.161286	0.417247
4	249.283	0.148284	0.0351633	0.0987887	0.512975
5	269.827	0.217147	**0.0662246**	0.0605545	0.516849
6	233.684	0.246841	0.051412	0.0458703	0.45471
7	214.875	0.352033	0.0397119	0.0396538	0.520334
8	216.269	0.273105	0.0400711	0.0302446	0.523658
9	285.089	0.137057	0.0410937	0.0217647	0.477216
10	**300.15**	**0.136089**	0.0656321	**0.0180894**	0.458114
	Data set: Data_4_3				
2	274.903	0.0755651	0.326609	0.15113	**0.274846**
3	388.445	0.152236	0.0415338	0.0689785	0.292048
4	**941.875**	**0.0520016**	**0.648233**	0.0244999	0.286967
5	180.142	1.715	0.0415338	0.0420537	0.657471
6	506.692	1.32732	0.0365537	0.0204898	0.689287
7	98.8002	2.29986	0.0415338	0.0329479	0.697946
8	358.767	0.811093	0.0338636	0.0203799	0.799155
9	344.214	0.928837	0.0318273	**0.0113638**	0.805329
10	284.893	1.0361	0.0584998	0.0142051	0.866336
	Data set: Ruspini				
2	5848.23	0.14342	0.440293	0.287543	0.379108
3	7018.97	0.173883	0.235521	0.100126	0.346559
4	**24005**	**0.0438631**	**0.504716**	0.029182	**0.265825**
5	19170.7	0.558863	0.0629802	0.0241887	0.401386
6	15592.3	0.489	0.0629802	0.0214562	0.527012
7	14248.4	0.351515	0.0848528	0.0187636	0.536729
8	13156.6	0.305388	0.0768978	0.0133081	0.533993
9	12087	0.571285	0.0768978	0.0115953	0.619698
10	11699	0.470023	0.0967239	**0.00995556**	0.600024

Table 4.9 shows a comparison of the actual number of clusters present in all the benchmark data sets considered to the number of clusters obtained by the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices, using the FCM clustering algorithm. Tables 4.10 - 4.13 provide the actual values of the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices obtained using the FCM clustering algorithm, for $c = 2, 3, \ldots, 10$ for all the data sets.

For obtaining the results presented in Tables 4.3, 4.5, 4.6, 4.7, 4.8, 4.10, 4.11, 4.12, 4.13, the 'Multiple Clustering' mode (described in chapter 8) of the software package PatternWiz was used and the best values obtained are presented in the tables.

Table 4.9: Number of clusters obtained by the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices, using the FCM algorithm (entries in bold face indicate the correct determination of number of clusters in a data set by the respective indices). For FCM algorithm $m = 1.5$

Data set	Actual no. of clusters	No. of clusters obtained				
		PBMF	*XB*	*PC*	*MPC*	*CE*
Iris	3	**3**	2	2	2	2
Cancer	2	**2**	**2**	**2**	**2**	**2**
Wine	3	7	2	2	2	2
SODAR1	3	**3**	2	**3**	**3**	**3**
SODAR2	3	**3**	**3**	**3**	**3**	**3**
Data_3_2	3	4	**3**	**3**	**3**	**3**
Data_5_2	5	**5**	4	**5**	**5**	**5**
Data_6_2	6	**6**	4	**6**	**6**	**6**
Data_9_2	9	**9**	**9**	2	**9**	2
Data_10_2	10	**10**	**10**	**10**	**10**	2
Data_4_3	4	**4**	**4**	**4**	**4**	**4**
Ruspini	4	**4**	**4**	**4**	**4**	**4**

Table 4.10: Values of the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices, for $c = 2, 3, \ldots, 10$ for the Iris, Cancer and Wine data sets using the FCM algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: Iris				
	PBMF	***XB***	***PC***	***MPC***	***CE***
2	21.8314	**0.0620203**	**0.979713**	**0.959425**	**0.0591903**
3	**28.2204**	0.156352	0.949132	0.923698	0.145878
4	24.7414	0.219783	0.928866	0.905155	0.207842
5	20.7735	0.507059	0.901455	0.876819	0.283352
6	18.5807	0.394045	0.899112	0.878935	0.303569
7	17.7894	0.63782	0.884802	0.865602	0.343091
8	15.941	0.548355	0.862657	0.843037	0.406182
9	14.8778	0.466376	0.861481	0.844167	0.417872
10	13.8462	0.694578	0.852829	0.836477	0.446799
	Data set: Cancer				
2	**167.963**	**0.135267**	**0.958273**	**0.916547**	**0.116559**
3	143.575	0.524358	0.903204	0.854806	0.265562
4	120.485	2.11283	0.821695	0.762261	0.495875
5	104.341	1.88657	0.795454	0.744317	0.580622
6	93.2341	1.72108	0.772855	0.727426	0.657373
7	84.0972	2.27286	0.756395	0.715794	0.718132
8	74.8518	1.52496	0.747244	0.711136	0.761648
9	66.6795	1.4533	0.742303	0.710091	0.792195
10	64.6411	3.8884	0.697791	0.664212	0.929737
	Data set: Wine				
2	372177	**0.0684558**	**0.969464**	**0.938928**	**0.0839274**
3	553537	0.16076	0.949492	0.924237	0.144544
4	788064	0.119303	0.945154	0.926872	0.153577
5	897532	0.0969246	0.944545	0.930681	0.158683
6	1.00171e+6	0.097282	0.942468	0.930962	0.165287
7	**1.34749e+6**	0.111672	0.943863	0.934507	0.162565
8	1.29341e+6	0.14526	0.941334	0.932953	0.171758
9	1.18169e+6	0.123471	0.941245	0.933901	0.173618
10	1.30725e+6	0.110358	0.941135	0.934595	0.173518

Table 4.11: Values of the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices, for c = 2, 3, ..., 10 for the SODAR1, SODAR2 and Data_3_2 data sets using the FCM algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: SODAR1				
	PBMF	***XB***	***PC***	***MPC***	***CE***
2	19402.5	**0.11969**	0.939358	0.878717	0.161218
3	**34745.1**	0.137554	**0.9639**	**0.94585**	**0.115619**
4	32973.3	0.176459	0.950936	0.934582	0.154716
5	24010.8	0.48083	0.91346	0.891826	0.261289
6	24976.4	0.373252	0.904747	0.885696	0.283045
7	25418	0.362437	0.903734	0.887689	0.293598
8	23302.9	0.327553	0.907729	0.894547	0.290425
9	23097.8	0.375463	0.893127	0.879767	0.325711
10	22393.4	0.253546	0.897694	0.886326	0.315037
	Data set: SODAR2				
2	21334.6	0.0870571	0.939116	0.878232	0.159847
3	**58914.4**	**0.0560778**	**0.986798**	**0.980196**	**0.0473207**
4	45523.6	0.187689	0.974161	0.965547	0.0854064
5	39065.3	0.392895	0.949663	0.937079	0.154049
6	31187.3	0.533354	0.940988	0.929186	0.185466
7	45525.2	0.283838	0.932241	0.920948	0.201596
8	41183.4	0.311369	0.92649	0.915988	0.22212
9	41530.4	0.494951	0.916756	0.906351	0.248813
10	38569.2	0.256546	0.921322	0.91258	0.237547
	Data set: Data_3_2				
2	10.2777	0.0991942	0.980581	0.961163	0.0655075
3	16.6513	**0.0820799**	**0.986119**	**0.979178**	**0.0521428**
4	**20.3442**	0.198198	0.952736	0.936981	0.140287
5	18.5852	0.408041	0.929793	0.912241	0.206435
6	16.5135	0.327192	0.91689	0.900268	0.240078
7	14.6465	0.447889	0.902874	0.886687	0.290466
8	12.914	0.408889	0.896624	0.881857	0.311045
9	13.6835	0.297675	0.887109	0.872998	0.332147
10	14.1117	0.254865	0.886583	0.873981	0.334677

Table 4.12: Values of the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices, for c = 2, 3, ..., 10 for the Data_5_2, Data_6_2 and Data_9_2 data sets using the FCM algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: Data_5_2				
	PBMF	***XB***	***PC***	***MPC***	***CE***
2	9.66792	0.427427	0.892912	0.785824	0.290434
3	11.1323	0.199091	0.881241	0.821862	0.339221
4	20.6979	**0.11544**	0.917881	0.890508	0.255557
5	**22.9332**	0.132255	**0.925252**	**0.906565**	**0.238663**
6	21.1297	0.337375	0.900893	0.881071	0.309039
7	19.0783	0.266367	0.888878	0.870358	0.346722
8	17.9805	0.350945	0.877278	0.859746	0.385514
9	18.1967	0.312778	0.867334	0.85075	0.410412
10	18.1206	0.308445	0.864894	0.849882	0.420045
	Data set: Data_6_2				
2	90.9368	0.307837	0.921979	0.843957	0.210224
3	113.558	0.136096	0.936966	0.905449	0.180614
4	372.346	**0.0429946**	0.986684	0.982245	0.0545747
5	429.522	0.130485	0.986435	0.983044	0.053018
6	**638.347**	0.0543478	**0.992801**	**0.991361**	**0.0288779**
7	527.582	0.572068	0.978099	0.974448	0.0701231
8	492.956	0.550921	0.960657	0.955037	0.115773
9	491.928	0.489647	0.947522	0.940963	0.152752
10	390.247	0.437004	0.948617	0.942908	0.153619
	Data set: Data_9_2				
2	4.97341	0.35803	**0.894294**	0.788587	**0.280409**
3	4.81928	0.191838	0.880133	0.8202	0.339713
4	6.70596	0.152081	0.874517	0.83269	0.366763
5	6.72629	0.237343	0.856118	0.820148	0.429569
6	6.15833	0.176372	0.861123	0.833348	0.42562
7	6.00433	0.18247	0.861442	0.838349	0.430944
8	6.07322	0.147432	0.871677	0.853345	0.406284
9	**6.78326**	**0.120628**	0.885222	**0.870875**	0.367012
10	6.24479	0.309694	0.877447	0.86383	0.393388

Table 4.13: Values of the *PBMF*, *XB*, *PC*, *MPC* and *CE* validity indices, for $c = 2, 3, \ldots, 10$ for the Data_10_2, Data_4_3 and Ruspini data sets using the FCM algorithm (entries in bold face indicate the optimal values for the respective indices)

No. of clusters	Data set: Data_10_2				
	PBMF	***XB***	***PC***	***MPC***	***CE***
2	187.067	0.144337	0.941036	0.882072	**0.163118**
3	282.279	0.219259	0.899258	0.848887	0.280982
4	298.683	0.135128	0.928515	0.904687	0.221545
5	256.374	0.22928	0.908961	0.886202	0.27871
6	297.758	0.187855	0.914709	0.897651	0.26662
7	333.641	0.194382	0.908737	0.893527	0.284127
8	343.843	0.149256	0.917017	0.905162	0.261649
9	260.966	0.643732	0.907537	0.89598	0.295511
10	**345.917**	**0.12599**	**0.950802**	**0.945336**	0.167801
	Data set: Data_4_3				
2	290.283	0.0735712	0.980701	0.961401	0.0648827
3	598.351	0.0838747	0.934147	0.90122	0.181087
4	**960.79**	**0.051736**	**0.991145**	**0.988193**	**0.037491**
5	720.063	0.646528	0.95929	0.949113	0.125813
6	581.335	1.4417	0.919534	0.90344	0.230634
7	575.107	1.2004	0.884122	0.864809	0.317336
8	333.336	0.768899	0.940963	0.932529	0.198867
9	466.378	0.948232	0.853353	0.835022	0.406954
10	413.704	0.893127	0.839023	0.821137	0.453226
	Data set: Ruspini				
2	6280.37	0.140302	0.967713	0.935427	0.103642
3	7675.75	0.167926	0.966268	0.949402	0.114036
4	**25196.7**	**0.0426941**	**0.988394**	**0.984525**	**0.0429053**
5	20022.5	0.135182	0.982153	0.977692	0.0603392
6	17978.9	0.396504	0.960694	0.952833	0.118587
7	15420.2	0.365011	0.95686	0.949671	0.132029
8	13948.6	0.569625	0.937847	0.928968	0.182045
9	15160.2	0.435647	0.917591	0.90729	0.235139
10	10462.6	0.587345	0.938273	0.931415	0.191991

Results presented in Tables 4.4 – 4.8 show that the best validation result for crisp clustering is obtained by the *PBM* validity index among all the tested validity indices. Other than the *PBM* validity index, the *XB* and *DI* validity indices also show good performances. The *DB* validity index also shows somewhat satisfactory performance for some of the data sets but the *SC* validity index shows poor performance for almost all the data sets. Results presented in Tables 4.9 – 4.13 show that the best validation result for fuzzy clustering is obtained by the *PBMF* validity index among all the tested validity indices. Other than *PBMF* index the *MPC*, *PC* and *CE* validity indices also depict good performances in that order, but the *XB* index does not perform much well. For *CE* index the natural logarithm (logarithm to the base $e \approx 2.71828183$) was used.

By studying all the Tables 4.4 – 4.13 it can be inferred that both the hard and fuzzy versions of the *PBM* validity index in general outperform all the other validity indices for almost all the data sets considered. This reinforces the results presented by Maulik and Bandyopadhyay in [56] and Pakhira et al. in [68].

4.7.2 Image Clustering

The classification of a digital image into distinct regions of interest is a fundamental step in computer vision. Technically, it is called as image segmentation. The goal of image segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. It is typically used to locate meaningful objects in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share some common visual characteristics. The result of image segmentation is a set of segments that collectively cover the entire image. Each of the pixels in a region is similar with respect to some characteristic or computed property, such as color, intensity, or texture [93].

Clustering is a key tool in image segmentation [94]. To demonstrate the effectiveness of clustering in pattern recognition on images, experiments on image clustering were conducted using the K-means and FCM clustering algorithms on several well known natural images. The results of some of the experiments are presented in this subsection. To begin with the presentation of results, the natural

images used in the experiments are first presented:

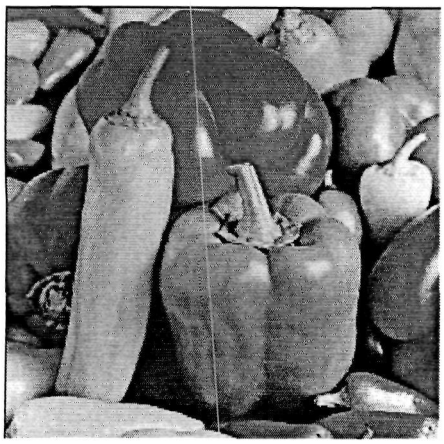


Figure 4.15: ***Peppers image.*** **Courtesy of the Signal and Image Processing Institute at the University of Southern California. Dimension of image: 256×256 pixels**

Figure 4.16: ***Brandy rose image.*** **Copyright photo courtesy of Toni Lankerd, 18347 Woodland Ridge Dr. Apt #7, Spring Lake, MI 49456, U.S.A. (tlankerd@charter.net). Dimension of image: 200×287 pixels**

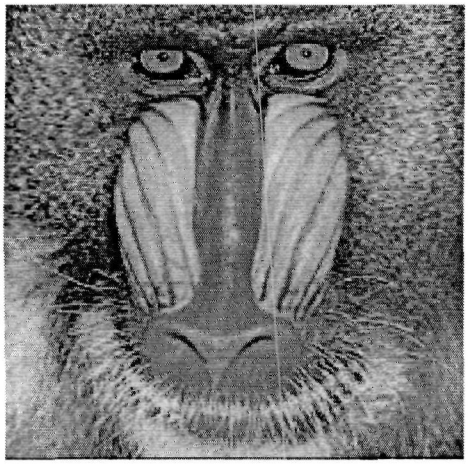


Figure 4.17: ***Baboon image.*** **Courtesy of the Signal and Image Processing Institute at the University of Southern California. Dimension of image: 200×200 pixels**

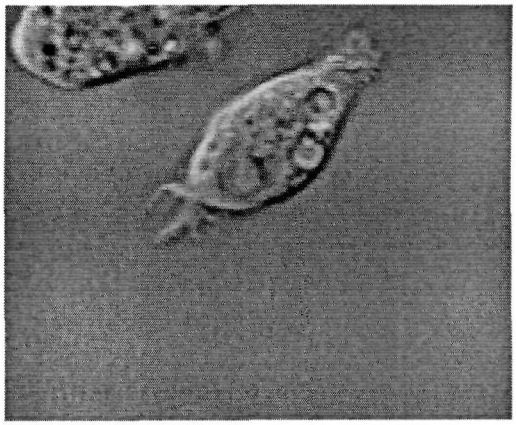


Figure 4.18: ***Cell image.*** **Courtesy of Alan W. Partin, Johns Hopkins University. Dimension of image: 191×159 pixels**

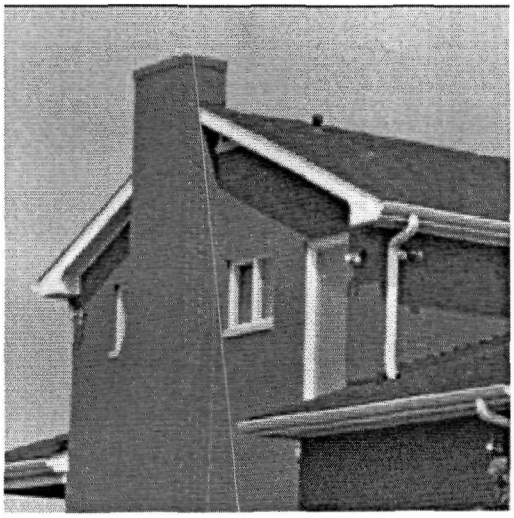


Figure 4.19: *House image.*
Courtesy of the Signal and Image Processing Institute at the University of Southern California.
Dimension of image: 256×256 pixels

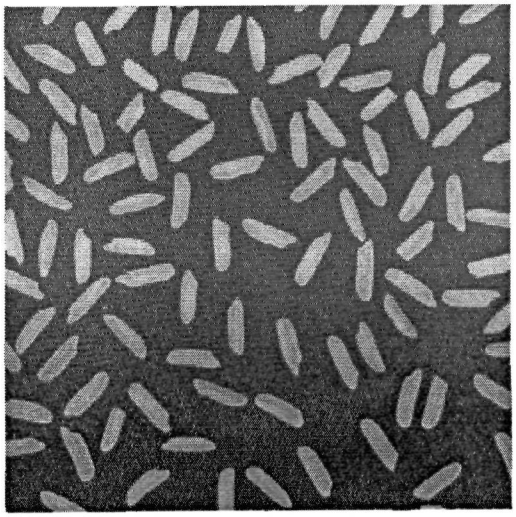


Figure 4.20: *Rice image.*
Courtesy of MATLAB 7 Image Processing Toolbox, © The MathWorks, Inc. 1984-2006.
Dimension of image: 256×256 pixels

Figure 4.21: *Pout image.*
Courtesy of MATLAB 7 Image Processing Toolbox, © The MathWorks, Inc. 1984-2006.
Dimension of image: 240×291 pixels

Figure 4.22: *Coins image.*
Courtesy of MATLAB 7 Image Processing Toolbox, © The MathWorks, Inc. 1984-2006.
Dimension of image: 300×246 pixels

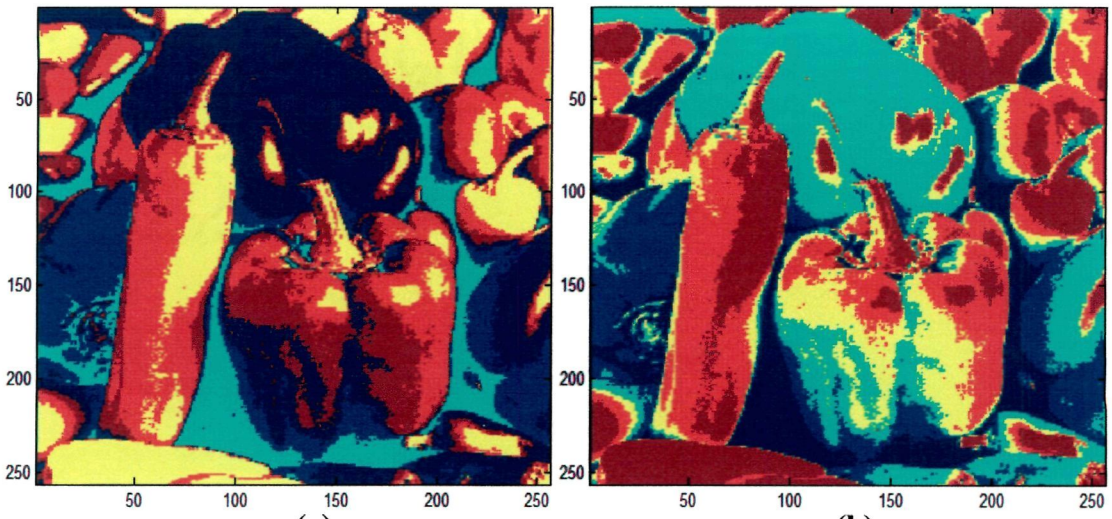
As can be seen from the images, the determination of the optimal number of clusters for the images may be quite complicated, because, the selection of the number of clusters for clustering an image is usually dependant on the number of 'objects of interest' to be considered. Thus, if for the *Coins* image, only the circular coins are considered as 'objects of interest' then the *Coins* image can be clustered by considering the number of clusters as 2 only, however, if somebody is also interested in a detailed segmentation of the coins also, so that the designs carved on the circular coins also get demarcated, then the number of clusters will vary.

For the experiments presented in this subsection, the optimal number of clusters for the images *Baboon* and *Peppers* were obtained from [95]. For all other images, the optimal number of clusters for each of them was estimated through a visual analysis survey by a group of five people. In a visual analysis survey, several clusterings of each image for a varying number of clusters are presented to a group of people and a common consensus is achieved on the optimal segmentation and consequently on the optimal number of clusters for that image. This approach of determination of number of clusters has been used by other researchers also [95] [96].

Table 4.14: Optimal number of clusters considered for the natural images

Image name	No. of clusters considered
Peppers	6
Brandy rose	3
Baboon	6
Cell	3
House	9
Rice	3
Pout	9
Coins	6

Some experimental results of clustering obtained by K-means and FCM on the images considered are now presented next. The values for the number of iterations and average execution time are provided for the first optimal clustering encountered in a series of 50 trials. For FCM, $m = 1.5$. From the results it can be seen that FCM is quite costlier than K-means, however the results obtained by both the algorithms are comparable. For validation, the *PBM* and *PBMF* validity indices were used.



(a) **(b)**

Figure 4.23(a): Peppers image clustered using K-means for $K = 6$.

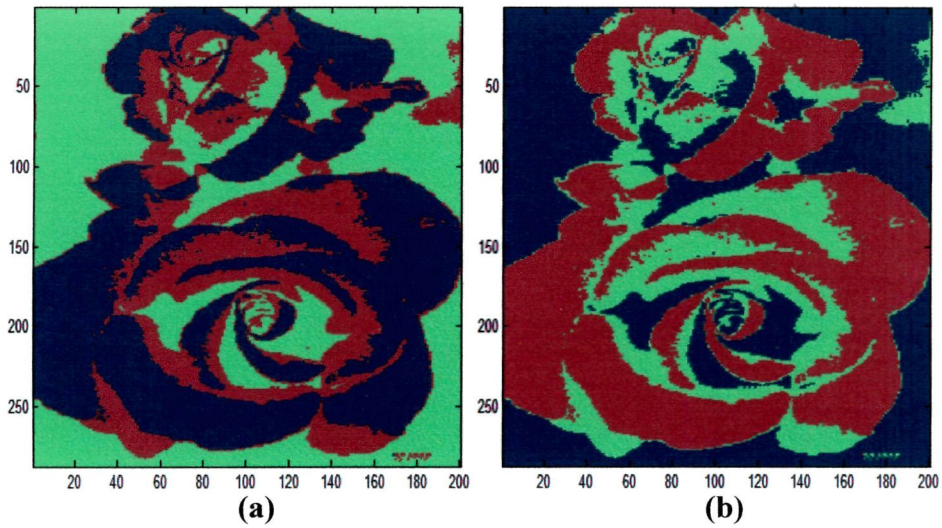
No. of iterations = 12. Average execution time = 0.90688 seconds.

Cumulative execution time = 81.6258. Best *PBM* index value = 83062.4

Figure 4.23(b): Peppers image clustered using FCM for $c = 6$.

No. of iterations = 87. Average execution time = 26.746 seconds.

Cumulative execution time = 1191.7891. Best *PBMF* index value = 99914.9



(a) **(b)**

Figure 4.24(a): Brandy rose image clustered using K-means for $K = 3$.

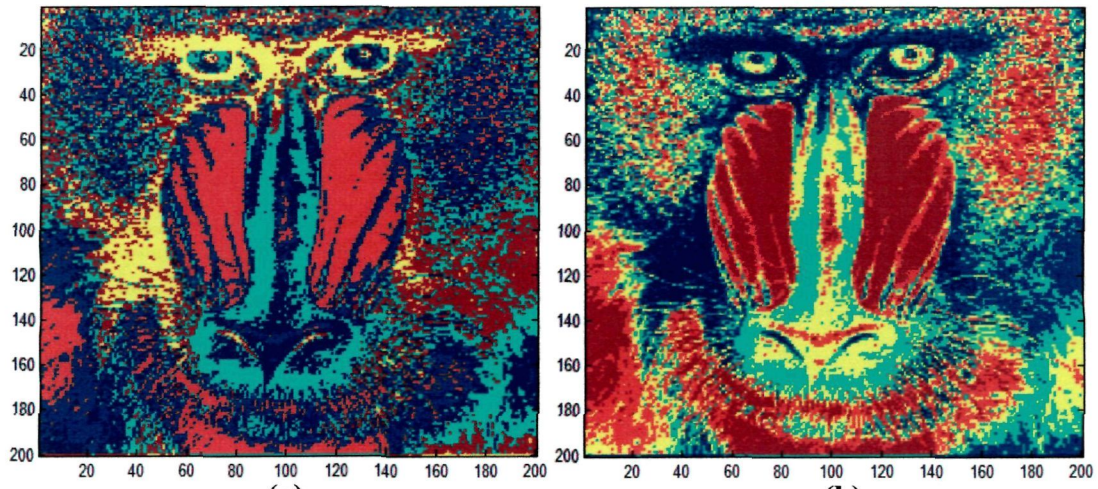
No. of iterations = 11. Average execution time = 0.34096 seconds.

Cumulative execution time = 19.3949. Best *PBM* index value = 51390.5

Figure 4.24(b): Brandy rose image clustered using FCM for $c = 3$.

No. of iterations = 35. Average execution time = 4.6812 seconds.

Cumulative execution time = 247.2499. Best *PBMF* index value = 59738.7



(a) **(b)**

Figure 4.25(a): Baboon image clustered using K-means for $K = 6$.

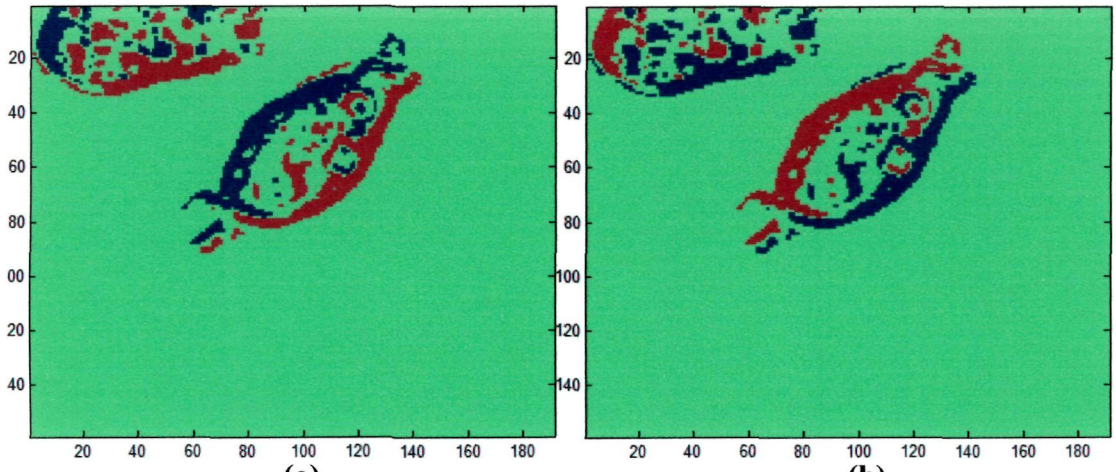
No. of iterations = 8. Average execution time = 0.32334 seconds.

Cumulative execution time = 37.777. Best *PBM* index value = 30190.5

Figure 4.25(b): Baboon image clustered using FCM for $c = 6$.

No. of iterations = 130. Average execution time = 23.2572 seconds.

Cumulative execution time = 1151.1049. Best *PBMF* index value = 37068.5



(a) **(b)**

Figure 4.26(a): Cell image clustered using K-means for $K = 3$.

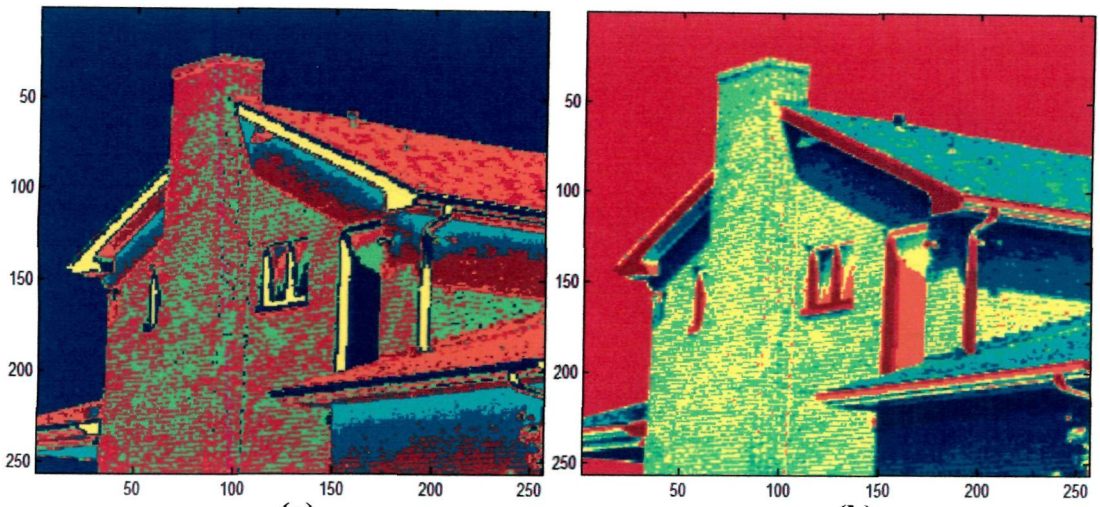
No. of iterations = 9. Average execution time = 0.14264 seconds.

Cumulative execution time = 6.6636. Best *PBM* index value = 5811.83

Figure 4.26(b): Cell image clustered using FCM for $c = 3$.

No. of iterations = 27. Average execution time = 1.8842 seconds.

Cumulative execution time = 87.0683. Best *PBMF* index value = 6595.66



(a) (b)

Figure 4.27(a): House image clustered using K-means for $K = 9$.

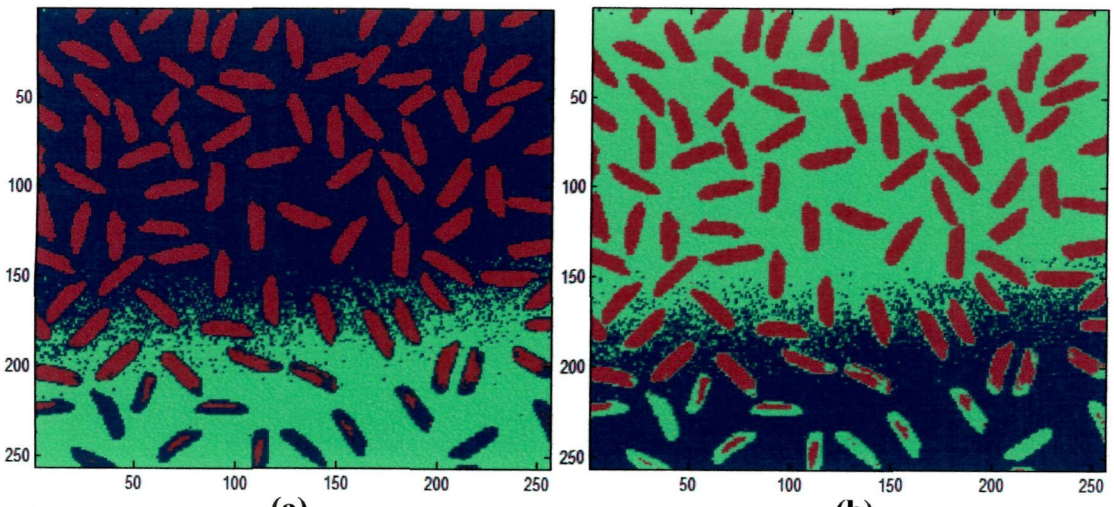
No. of iterations = 8. Average execution time = 0.91052 seconds.

Cumulative execution time = 104.6877. Best *PBM* index value = 167654

Figure 4.27(b): House image clustered using FCM for $c = 9$.

No. of iterations = 85. Average execution time = 41.186 seconds.

Cumulative execution time = 2621.2945. Best *PBMF* index value = 203598



(a) (b)

Figure 4.28(a): Rice image clustered using K-means for $K = 3$.

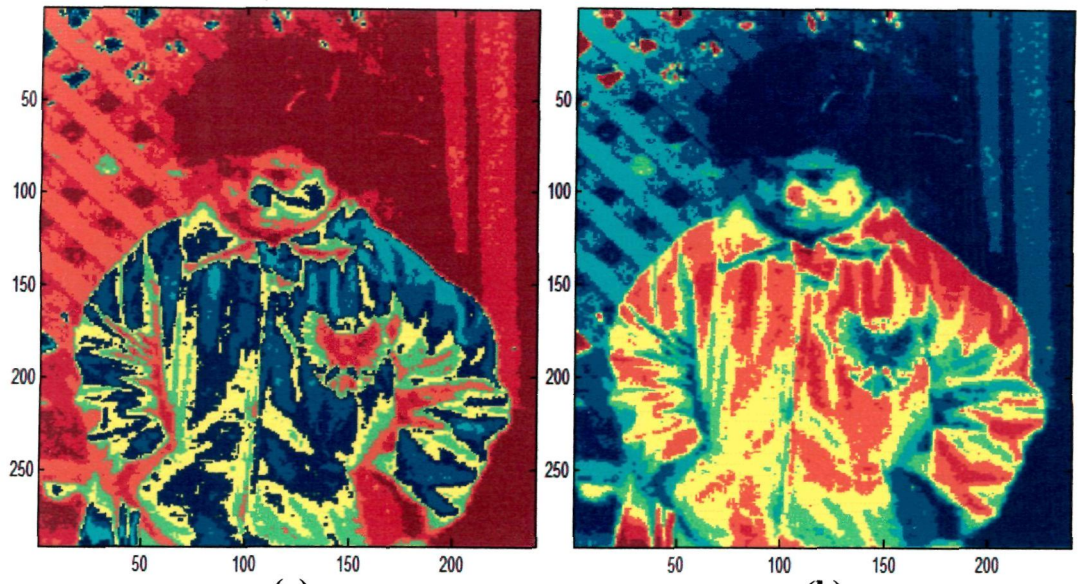
No. of iterations = 5. Average execution time = 0.22381 seconds.

Cumulative execution time = 16.9948. Best *PBM* index value = 46175.6

Figure 4.28(b): Rice image clustered using FCM for $c = 3$.

No. of iterations = 22. Average execution time = 3.4222 seconds.

Cumulative execution time = 164.1814. Best *PBMF* index value = 54430.1



(a) (b)

Figure 4.29(a): Pout image clustered using K-means for $K = 9$.

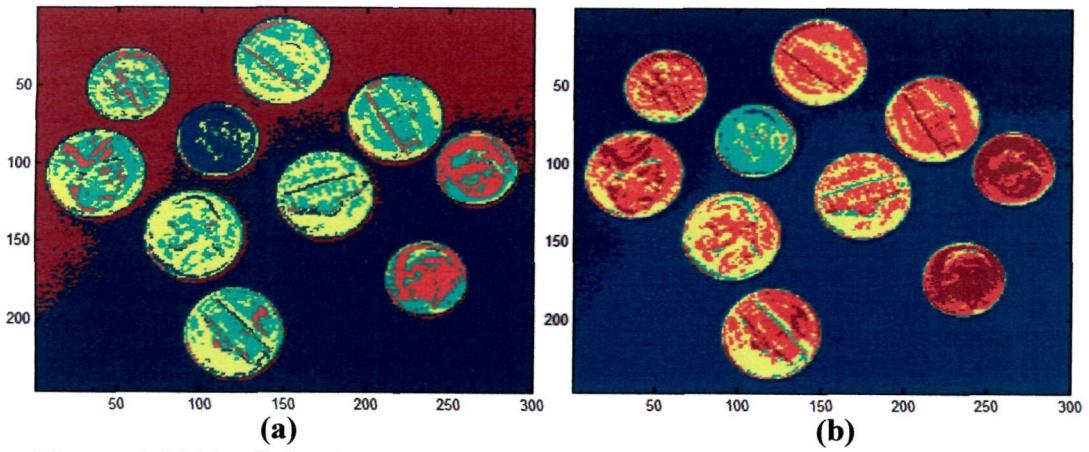
No. of iterations = 19. Average execution time = 2.1599 seconds.

Cumulative execution time = 74.5961. Best *PBM* index value = 32083.5

Figure 4.29(b): Pout image clustered using FCM for $c = 9$.

No. of iterations = 144. Average execution time = 72.3401 seconds.

Cumulative execution time = 2243.5209. Best *PBMF* index value = 40707



(a) (b)

Figure 4.30(a): Coins image clustered using K-means for $K = 6$.

No. of iterations = 14. Average execution time = 1.1461 seconds.

Cumulative execution time = 78.2023. Best *PBM* index value = 260387

Figure 4.30(b): Coins image clustered using FCM for $c = 6$.

No. of iterations = 113. Average execution time = 38.6004 seconds.

Cumulative execution time = 1732.0642. Best *PBMF* index value = 308375

From all the clustering results on the images considered, it can be clearly comprehended that clustering is indeed a very effective tool for image segmentation.

4.7.3 Text Clustering

In this subsection, experimental results of text document clustering using the K-means clustering algorithm are presented. For the representation of the text documents in the collection of text documents considered, the Vector Space Model representation of the documents [35] using the TF-IDF weighting scheme [97] [98] was used. Since the TF-IDF matrix of text documents can grow very large with thousands of dimensions for each pattern vector representing a text document in the text collection, the FCM algorithm was not used in this case, as it has already been seen that FCM is computationally costly. Moreover the FCM algorithm is also heavily dependant on the fuzzifier parameter m and the choice of a good value of m is based on the problem under consideration. So, deciding for a good value of m in every case, for different types of text collections can be a difficult task.

For the experiments on text clustering a simplified version of the 20 Newsgroups text data collection [99] obtained from the UCI machine learning repository [81] was used.

The 20 Newsgroups text collection data set consists of 20000 messages taken from 20 Usenet newsgroups each corresponding to a different topic. The twenty newsgroups are:

comp.graphics *comp.os.ms-windows.misc* *comp.sys.ibm.pc.hardware* *comp.sys.mac.hardware* *comp.windows.x*	*rec.autos* *rec.motorcycles* *rec.sport.baseball* *rec.sport.hockey*	*sci.crypt* *sci.electronics* *sci.med* *sci.space*
misc.forsale	*talk.politics.misc* *talk.politics.guns* *talk.politics.mideast*	*talk.religion.misc* *alt.atheism* *soc.religion.christian*

The simplified subset used in the experiment is composed of 10 articles each from each of the above 20 newsgroups. That is a total of 200 text documents are there in

the set. The preprocessing of the set of documents was done using a set of 700 stop-words and 39 special characters to be filtered out from the documents. After that the set of text documents, when converted using the vector space model technique and TF-IDF weighting scheme resulted in a 200×10602 sized 2-dimensional data matrix which when saved on disk consumed 18.1 Mega Bytes of disk space. Thus the resulting data set that was used in the experiment is quite a large data set. Hereafter it is referred to as *20-mini-newsgroups* data set.

A plot of the data set is shown below:

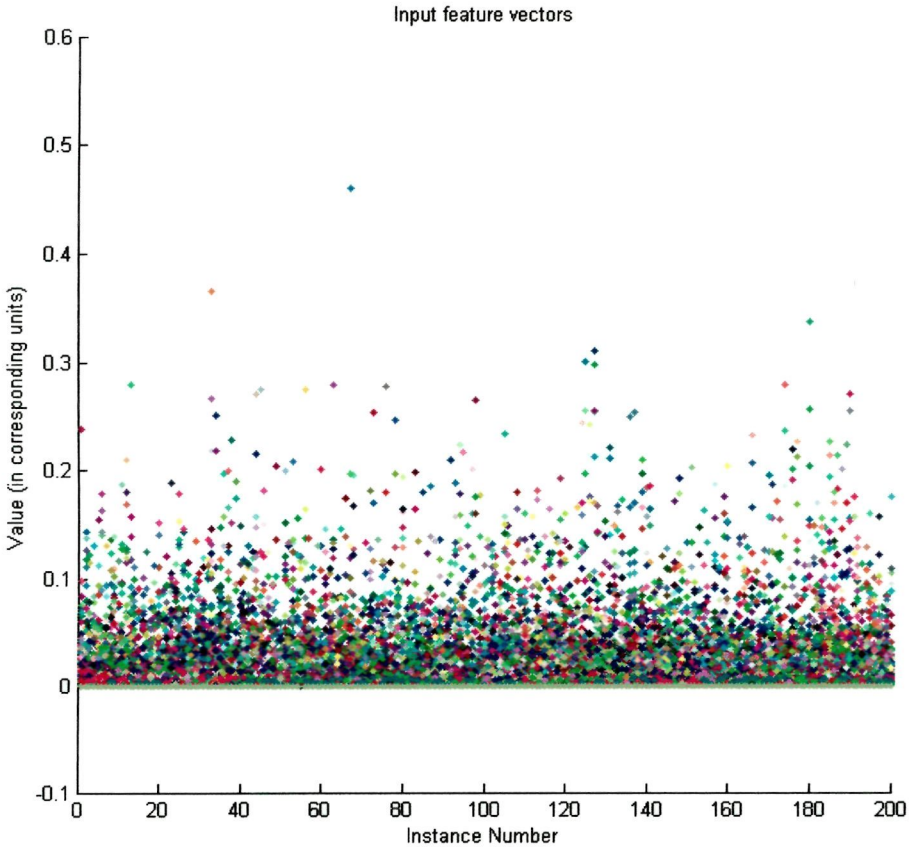


Figure 4.31: Instance number vs. input feature vector values' plot for the 20-mini-newsgroups data set

From the plot it can be easily seen that the 20 clusters are quite overlapping and linearly inseparable. The results of application of K-means algorithm on this data set are now presented next. As there are 20 newsgroups so the value of K was taken as 20. Figure 4.20 shows the summary of the best clustering (as decided by the optimum value of *PBM*) obtained in a series of 50 trials of the K-means algorithm on the 20-mini-newsgroups data set. The cosine distance was used as the proximity measure for

the K-means algorithm and also for the evaluation of value for the *PBM* index.

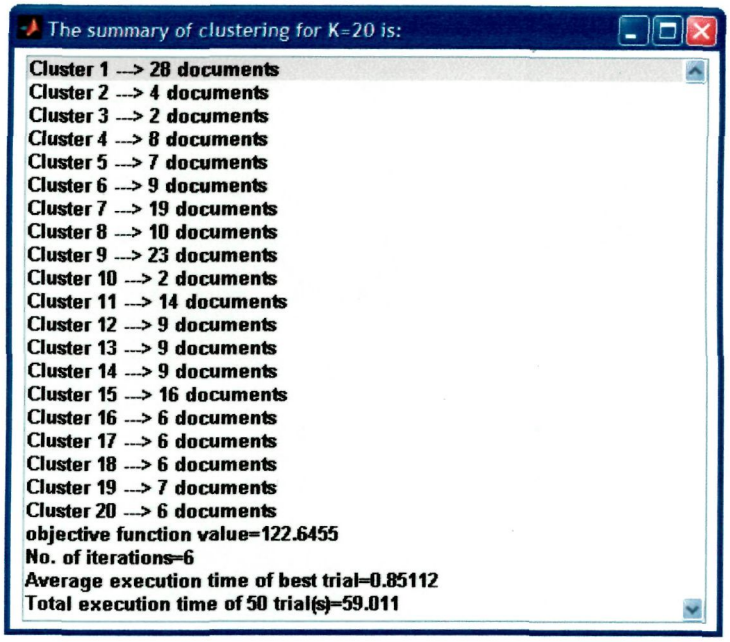


Figure 4.32: Summary output window (from PatternWiz) for K = 20 for the 20-mini-newsgroups data set. Best *PBM* index value = 0.00272557

As can be seen from the results, the sizes of most of the clusters are closer to the ideal value of 10. Moreover the iteration count and execution time (in seconds) required by the best trial of the K-means algorithm is also quite less.

The silhouette plot for the 20 clusters is shown below:

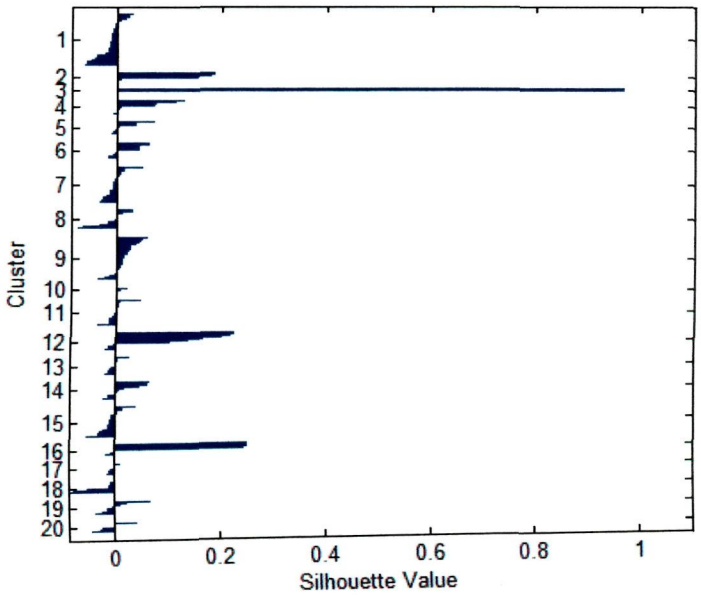


Figure 4.33: Silhouette plot for clustering result obtained by K-means for K = 20 for the 20-mini-newsgroups data set

From the silhouette plot, however, it can be inferred that the clusters formed are not quite well separated, as indicated by the negative silhouette values, and the degree of similarity between the individual text documents in each cluster is also low as indicated by a low silhouette value of 0 to 0.4 for almost all the clusters. A glimpse of the cluster assignments for the individual documents is shown below:

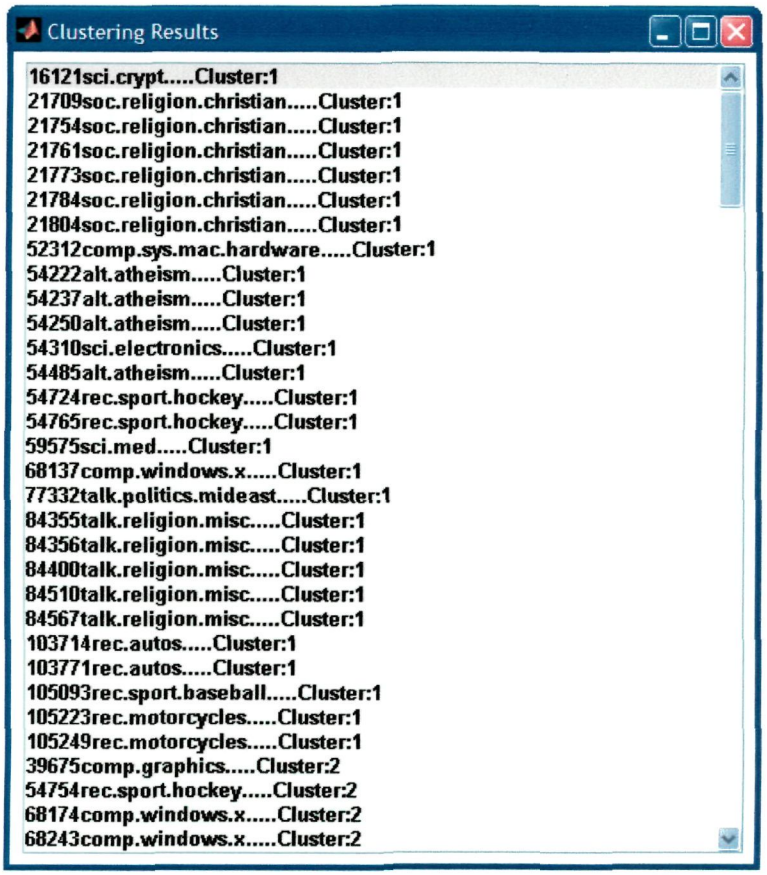


Figure 4.34: Glimpse of the cluster assignments obtained by K-means for $K = 20$ for the 20-mini-newsgroups data set

From this experiment on text clustering conducted using K-means it can thus be inferred that the usual clustering methodology is not very much suitable for text clustering. For the classification of text documents in a more meaningful way, text document classification should involve the discovery of semantic, lexical and ontological relations in the texts of a given sets of documents. The vector space model representation of text documents do not consider such important relations in the text documents which leads to a very low relevance score for relevant documents. Moreover, the vector representations of documents tend to use all the words in the

text documents after removing the stop-words, regardless of their meaning and semantic relations with other words in the collection of documents. This leads to thousands of dimensions in the vector representation of documents. It is however well known that only a very small number of words/terms in documents have distinguishable power to classify documents. Thus these problems need to be solved in an efficient manner for better clustering results for text documents. As already mentioned in subsection 2.6.3, these problems can be addressed very well by using WordNet lexical categories and WordNet ontology as a result of which a well structured document vector space of low dimensionality can be created so that better performances by the clustering algorithms can be attained.

To summarize this section on experimental results of clustering, a comparison of K-means and FCM clustering algorithms is presented below:

Table 4.15: Comparison of K-means and Fuzzy c-means clustering algorithms

Name of algorithm	K-means	Fuzzy c-means
Type of algorithm	Hard, partitional, non-deterministic, polythetic, non-incremental	Fuzzy, pseudo-partitional, non-deterministic, polythetic, non-incremental
Inputs	No. of clusters K	No. of clusters c, fuzzifier parameter m
Shape of clusters	Hyper-spherical	Non-convex
Search Technique	Localized	Localized
Nature of convergence	Convergence to local minima of sum of squared error objective function.	Convergence to local minima of sum of squared error objective function.
Time complexity*	$O(nKdt)$	$O(ncdt)$
Advantages	Fast and easy to implement, suitable for medium to large data sets	Good at dealing with fuzziness of clusters, applicable to medium and large data sets
Disadvantages	Dependent on initial centroids, Not well suited for data with fuzzy overlapping clusters.	Dependent on initial centroids and m, Computationally a little bit costly

*n = no. of points, K, c = no. of clusters, d = no. of dimensions, t = no. of iterations

4.8 Chapter Summary

In this chapter first a brief overview on clustering, similarity measures and different taxonomical representations of clustering are presented. Then discussions on two prominent clustering algorithms namely the K-means hard clustering algorithm and the Fuzzy c-means (FCM) fuzzy clustering algorithm are presented. After that, some popular clustering validity assessment indices for hard and fuzzy clustering are discussed. Further, some experiments on the applications of the K-means and FCM clustering algorithms for pattern recognition on numeric, image and text data are presented. Experiments on numeric and image data showed good performances by both the K-means and FCM clustering algorithms with the K-means algorithm being less costly than FCM. Experimental results on validation of clustering obtained by the two algorithms showed that the *PBM* validity index and its fuzzy version *PBMF* gives the best performance among all the validity indices experimented. Experiments on text clustering revealed the inability of conventional clustering approaches in efficiently utilizing the inherent semantic relationships of words/terms in text documents, for better meaningful results. In general, all the experiments showed that both the algorithms are very well suited for solving a wide array of clustering problems, from various problem and data domains.

CHAPTER 5

An Efficient Deterministic K-means Clustering Algorithm

This chapter presents an efficient deterministic version of the widely used K-means clustering algorithm. To illustrate the efficiency of the proposed algorithm it was compared with K-means through experiments on a number of benchmark data sets and experimental results so obtained are presented in this chapter.

5.1 Introduction

The classical K-means algorithm, discussed in section 4.4, starts with K initial centroids (or seeds) chosen at random inside the hyper-volume containing the pattern set X, then it assigns each data point to the nearest centroid, recalculates the cluster centroids by taking the mean of all points in each cluster, and repeats the process until the K centroids do not change or negligible changes occurs in two successive iterations. The K-means algorithm is a greedy algorithm for minimizing *SSE* hence it may not converge to the global optimum. The performance of K-means algorithm thus strongly depends on the choice of seeds. But the choice of good seeds is not easy. Thus to obtain optimal clustering using classical K-means, an uncertain number of trials of K-means with varying number of iterations is needed to be applied to the data set, each trial starting with a randomly determined set of seeds. In this chapter, thus a fast deterministic clustering algorithm is presented which is based on the classical K-means clustering algorithm. It was developed as one of the principal objectives of the present research. The proposed approach uses a deterministic unit hyper-block based

division technique, for seed initialization for K-means algorithm, so as to overcome the major shortcomings of K-means, viz., random selection of seeds, uncertainty about the number of trials and iterations of K-means required and convergence to one of several local minima. The approach is presented in detail in section 5.4. In section 5.2, a brief outline of some related work on seed initialization of K-means is presented. In section 5.3, the shortcomings of K-means algorithm are enlisted in brief. In section 5.5 experimental results are presented for enunciating the efficiency of the proposed approach. Finally, in section 5.6, a summary of the chapter is presented.

5.2 Related Work

The primary aim of any K-means based improved clustering approach, is to find an effective method for selection of good seeds. Several methods have been developed by many researchers working in the field of clustering for obtaining good seeds for K-means. Bradley and Fayyad [100] proposed an algorithm that refines seeds by analyzing the distribution of data and estimating the modes of the joint probability density of the data. It is a technique that begins by randomly breaking the data into 10, or so, subsets. Then it performs a K-means clustering on each of the subsets, all starting with the same set of initial centroids chosen using Forgy's method. The Forgy's method chooses K instances of the data set at random as seeds and assigns the rest of the instances to the cluster represented by the nearest seed. Su and Dy [101] presented a deterministic seed initialization method based on principal component analysis based divisive hierarchical approach. It obtains the seeds by calculating the mean of each cluster produced by splitting the data set at each step. Beginning with one cluster the splitting is carried out to K clusters iteratively by the evaluation and minimization of the within-cluster *SSE*. Khan and Ahmad [102] proposed a Cluster Center Initialization Algorithm (*CCIA*) to solve the cluster initialization problem. The algorithm starts by calculating mean and standard deviation for data attributes, and then separates the data with normal curve into certain partition. *CCIA* uses K-means and density-based multi scale data condensation to observe the similarity of data patterns before finding out the seeds. Arai and

Barakbah [103] presented a hierarchical K-means algorithm that obtains the optimal seeds by applying classical K-means on the data for a predefined number of trials, recording the set of final centroids in each trial, and then applying a hierarchical algorithm like, Centroid Linkage, on the recorded sets of final centroids of all trials. Xinhua et al. [104] presented a technique of obtaining seeds based on self-adoptively selecting best density radius. The density radius r of a d-dimensional pattern is the radius of the hyper-sphere of patterns in its close vicinity. The technique starts with r set to an initial value and evaluates densities of all points. Thereafter it sets r within the range $0.8 \times n/K$ and $0.7 \times n/K$ in several steps, using the heuristic that, "if the largest density (calculated using current r) in all points is bigger than $0.8 \times n/K$, then r subtracts a length of step (e.g. 0.01) and if the largest density is smaller than $0.7 \times n/K$, then r adds on a length of step." In each step, it calculates the density of each pattern and sorts the patterns according to their densities. It then selects the pattern with highest density as the first centroid and calculates its distance with the pattern with next highest density. If the distance is smaller than D (which is taken as a certain multiple of r), then the latter one is left out, otherwise it is selected as the next centroid. After this the next pattern in the order of density is selected and its distances from all the centroids obtained till now are calculated. If the distances turn smaller than D, then it is left out, otherwise it is selected as the next centroid. All other centroids are determined in the same manner. Davidson and Satyanarayana [105] showed how bootstrap averaging with K-means can produce results comparable to clustering all of the data but in lesser time. The approach is superficially similar to that of [100] but uses an alternative to randomly choosing starting centroids. It starts with sub-sampling the training data, then clusters each sub-sample, clusters the resultant cluster centroids many times, to generate refined initial seeds for K-means. Arthur and Vassilvitskii [106] proposed a way of initializing K-means by choosing random seeds with very specific probabilities that is $\Theta(\log K)$-competitive with the optimal clustering. Belal and Daoud [107] proposed an algorithm which finds a set of medians extracted from the dimension with maximum variance to initialize the starting clusters for K-means. The idea of the algorithm is to find the dimension with maximum variance, sorting it, dividing it into a set of groups of data points, then

finding the median for each group and then using the corresponding data points to initialize the K-means algorithm. Likas et al. [108] presented a global K-means algorithm that dynamically adds one cluster centroid at a time through a deterministic global search procedure consisting of *n* executions of the K-means algorithm from suitable initial positions. Hung et al. [109] proposed a method that obtains the seeds from a simplified data set. The method has three phases. In phase I the original data set is partitioned into blocks and patterns lying in each block are determined. Each block unit, called a unit block (UB) must contain at least one pattern. UBs not containing any pattern are discarded. In phase II, the centroid of each unit block (CUB) is then computed by taking the mean of all points in a UB. All the computed CUBs form a reduced data set that represents the original data set. In phase III, the reduced data set is clustered by applying K-means to obtain the seeds for clustering the complete data set using K-means.

5.3 Shortcomings of K-means Clustering Algorithm

Though the classical K-means algorithm has many advantages like, ease in implementation, approximately linear time complexity, applicability to medium and large-sized data sets, lesser requirement for the user to specify input parameters and determinism after the choice of seeds, the algorithm has several inherent shortcomings as discussed below:

1) The algorithm is highly dependent on the initial choice of seeds.
2) Because K-means is a greedy algorithm, a choice of bad seeds may lead the algorithm to converge to sub-optimal solutions.
3) The algorithm is non-deterministic in the initiation phase prior to the selection of seeds. This property of K-means may not very desirable at times, because to obtain optimal clustering using K-means, an uncertain number of trials of K-means are needed to be applied to the dataset, with each trial starting with a different, randomly determined set of seeds and requiring different numbers of iterations in each trial.
4) It is unable to identify clusters with arbitrary shapes, ultimately imposing

hyper-spherical clusters on the data. This is its major limitation.

5) It is also unable to handle outliers properly.

Figure 5.1 shows an example of sub-optimal clustering that is often encountered with K-means. As can be seen, the data set has not been properly clustered into 4 distinct clusters, due to suboptimal convergence resulting from bad choice of seeds.

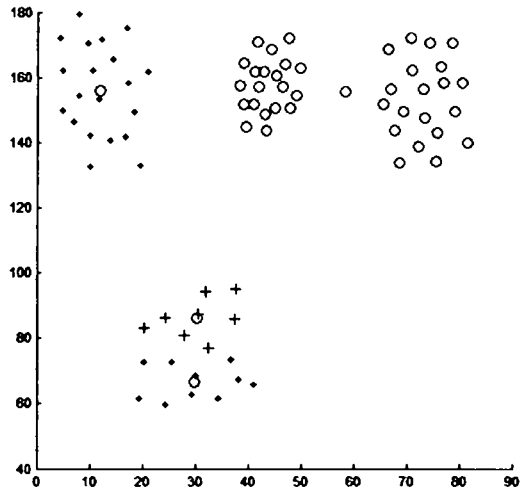


Figure 5.1: Graphical representation of sub-optimal clustering obtained by the K-means clustering algorithm

Figure 5.2 shows a pictorial representation of outliers. As can be seen from the figure, the outliers are data points which lie far apart from all other points in the data set. Thus the assignment of such observations to one of the several clusters in the data set is a difficult task.

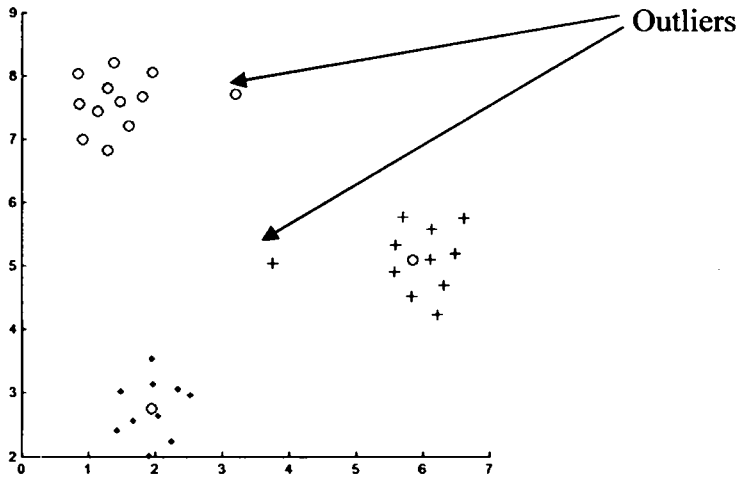


Figure 5.2: Outliers encountered in data clustering

5.4 The proposed HBDKM Clustering Algorithm

Most of the improved variants of K-means focus on finding better seeds so that faster and optimal convergence can be achieved by reducing the number of iterations. However, spending too much time for seed calculation seems contradictory against the simplicity and speed of the classical K-means algorithm, even though the direct application of classical K-means does not guarantee any accuracy. Thus in the light of the disadvantages of K-means discussed in section 5.3, any good improvement of K-means clustering algorithm should take care of four things primarily:

1) Fast and deterministic computation of seeds.
2) Validation of the seeds for their fitness.
3) Minimizing the number of iterations of K-means, in case K-means needs to be applied for several trials on the data set.
4) Assurance of clustering quality.

Considering these points, a new improved clustering algorithm named HBDKM (Hyper-Block division based Deterministic K-means) is proposed in this section. A *hyper-block* is a hyper-rectangular block containing a subset of points in the multidimensional feature space (the hyper-volume). A *hyper-rectangle* is the generalization of a *rectangle* for higher dimensions. The unit hyper-block-based partitioning technique used by the proposed approach is similar to the approach described in [109]. The proposed approach also takes motivations from the grid based clustering methodology [110]. It takes care of the above four improvement considerations and addresses the first three shortcomings of the K-means clustering algorithm viz., random selection of seeds, uncertainty about the number of iterations and/or trials of K-means and convergence to some local minima.

The proposed HBDKM algorithm consists of two phases, Phase I and Phase II. Phase I is the initialization phase that involves the fast and deterministic computation of seeds and checking of their fitness. Phase II is the clustering phase in which the classical K-means algorithm is applied only 'once' on the whole data set using the optimal set of centroids obtained in Phase I. The complete algorithm is now presented next:

Phase I

Step 1): Obtain the number of clusters K.

Step 2): Set $bpr = \left\lceil \left(\sqrt{K}\right) \right\rceil$.

Repeat Steps 3, 4, 5, 6 while $bpr \leq K$

Step 3): Partition the dataset X into bpr^d equally sized unit hyper-blocks and determine for each pattern $\mathbf{x}_j$, the coordinates (corresponding to each dimension of X) of the hyper-block to which $\mathbf{x}_j$ belongs, using the procedure described in Table 5.1.

Step 4): Calculate the centroids $\overline{\mathbf{x}}_k$'s of top K dense unit hyper-blocks using the procedure described in Table 5.2. The density of a unit hyper-block is simply the number of patterns in that hyper-block.

Step 5): For the set of K centroids obtained in the Step 5 evaluate the value of the fitness measure, the crisp *PBM* validity index (described in section 4.6.1) and record the value obtained.

Step 6): Set $bpr = bpr + 1$.

Step 7): Select the best set of centroids *cen_opt* from all the computed sets of centroids based on the maximal value of the *PBM* validity index obtained in Step 6.

Phase II

Step 8): Apply K-means algorithm on the original data set only once, using the optimal set of centroids *cen_opt* and output the result of clustering.

As can be seen from the algorithm, Phase I, consists of six steps, Step 1 through Step 6. Step 1 obtains the number of clusters K from the user and Step 2 calculates the quantity *bpr* which is called the *block partition ratio* that determines the number of splits to be done along each dimension. Step 1 and Step 2 are executed only once. In Step 3, the hyper-volume of the data set is partitioned into bpr^d equal sized unit hyper-blocks using the procedure described in Table 5.1 and for each pattern, the coordinates of the hyper-blocks to which it belongs, are determined. In Step 4, the centroids of top K dense unit hyper-blocks are computed using the procedure described in Table 5.2, to obtain a candidate set of K seeds. In Step 5, this set of K

centroids is evaluated for fitness by calculating the *PBM* validity index. In Step 6 *bpr* is incremented by 1. Step 7 obtains the optimal set of centroids *cen_opt* corresponding to the maximum value of the *PBM* validity index, by comparing the values of the *PBM* index for all the $K - \lceil \sqrt{K} \rceil + 1$ candidate sets for initial centroids.

Table 5.1: Procedure to partition the hyper-volume of original data set *X* into bpr^d unit hyper-blocks

```
procedure partition_data_set(X, bpr)
  for each dimension X(:, m) of the data set, where
  m = 1, ..., d do
    / / Find the maximum and minimum values along
    / / each dimension X(:, m)
    Max(m) = the maximum value of dimension X(:, m)
    Min(m) = the minimum value of dimension X(:, m)
    / / Calculate the interval of segment partition for
    / / dimension X(:, m)
    Intrv_of_Seg(m) = (Max(m) - Min(m)) / bpr
  end for
  for each pattern x_i do
    / / Calculate the unit hyper-block UHB to which
    / / pattern x_j belongs
    for each dimension X(:, m) of x_j named X(j, m), do
      Point_in_Dim(j,m) = ⌈ (X(i,m) - Min(m)) / Intrv_of_Seg(m) ⌉
      if Point_in_Dim(j,m) == 0
        Point_in_Dim(j,m) = 1
      end if

    end for
  end for
  / / Point_in_Dim is a n × d matrix whose each row
  / / contains the d coordinates of each unit hyper-block
  / / to which x_j belongs
end partition_data_set
```

Phase I is repeated for only a specific number of times $K - \lceil \sqrt{K} \rceil + 1$, and thus only $K - \lceil \sqrt{K} \rceil + 1$ candidate sets of centroids are obtained for evaluation, using the

PBM validity index. The choice of $bpr = \lceil \sqrt{K} \rceil$ is an intuitive reasonable choice for a data set having K clusters.

Table 5.2: Procedure to compute centroids of top *K* dense unit hyper-blocks

```
procedure compute_k_centroids(X, Point_in_Dim, K)
  / / Find all unique instances of rows of Point_in_Dim
  / / and store in upid and store the number of rows in upid
  / / in len
  for l = 1, ..., len do
    / / Calculate the indices of rows in Point_in_Dim
    / / containing upid(l, :) and store in idx
    Points_in_Block(l) = X(idx, :)
    / / Sort the array of hyper-blocks Points_in_Block in
    / / descending order according to the densities of
    / / hyper-blocks
  end for
  if len ≥ K
    / / Compute the centroids of top K dense hyper-blocks
    for j = 1, ..., K do
      CUHB(j, :) = mean of Points_in_Block(j)
    end for
  else
    / / Compute the centroids of all hyper-blocks
    for j =1, ..., len do
      CUHB(j, :) = mean of Points_in_Block(j)
    end for
    / / Set the remaining centroids to zeros
    for j = len +1, ..., K do
      CUHB(j, :) = A vector of all zeros of length d
    end for
  end if
end compute_k_centroids
```

Phase II, consists of only one step, Step 8. In Phase II actual clustering is done on the original data set by applying the K-means algorithm only once, using the optimal set of initial centroids *cen_opt* obtained in Step 7. At the end of Step 8 the result of clustering is given as output. Since the centroids are optimal, i.e., closer to the actual centroids thus only a few iterations of K-means are consumed for convergence.

Figure 5.3 shows the optimal initial centroids obtained by the HBDKM clustering algorithm using hyper-block division for a data set having four clusters.

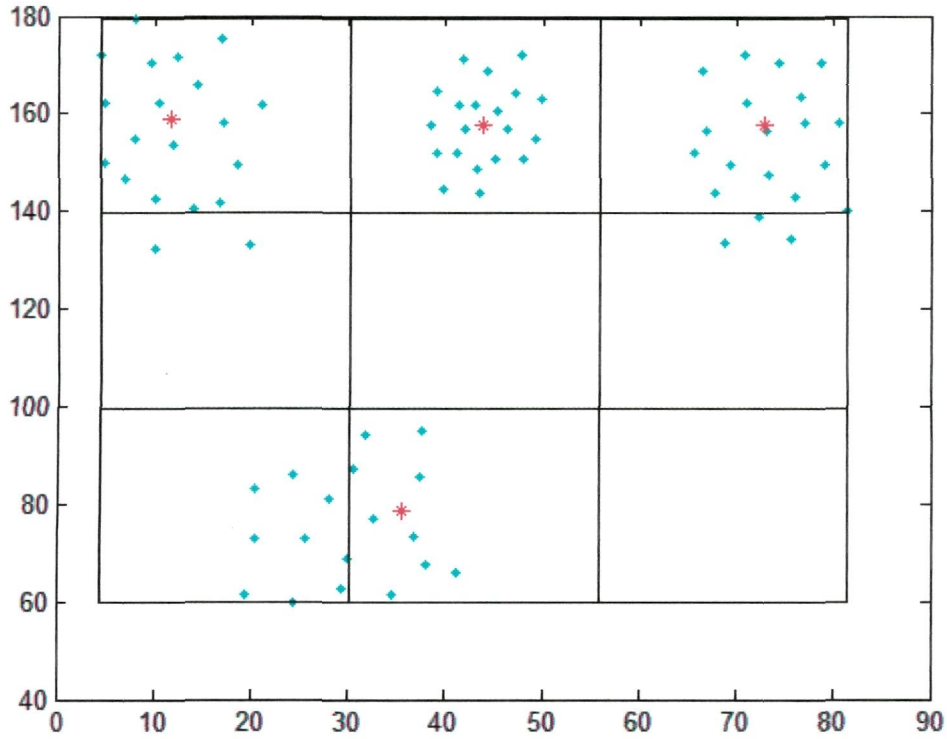


Figure 5.3: Optimal initial centroids obtained by the HBDKM clustering algorithm using hyper-block division for a data set having four clusters

It can now be seen that the proposed approach involves no randomness that incurs with classical K-means clustering algorithm. Unlike K-means, in this approach the choice of good seeds is done deterministically, rather than resorting to random techniques. The approach generates the candidate sets of seeds by incrementing the partitioning resolution from $\lceil \sqrt{K} \rceil$ to K by reducing the size of hyper-blocks after each iteration of steps of Phase I; a methodology similar to that used in grid based data clustering. The algorithm needs to check only a small deterministic number of good candidate sets of seeds, which is dependent only on K and not on the data set. It obtains the final seeds by evaluating an efficient and popular clustering validation measure, the *PBM* validity index, thus the quality of clustering is also assured. It needs to apply K-means algorithm on the full data set only once rather than for an uncertain number of trials as in the case of the classical K-means approach. The average execution time taken by the whole process is thus comparatively quite less. The final outcome of clustering by HBDKM algorithm is also highly optimal. A comparative analysis of HBDKM algorithm with the K-means algorithm, for the

demonstration of its effectiveness over K-means, is now presented in the next section.

5.5 Experimental Results and Discussion

To demonstrate the better performance of the proposed HBDKM clustering algorithm over the K-means clustering algorithm, a comparative study was carried out by applying the two algorithms on all the twelve benchmark data sets described in subsection 4.7.1. For both the algorithms the validation of clustering was done using the efficient *PBM* validity index. The experimental results obtained using the two algorithms, for all the data sets are presented in Table 5.3 and Table 5.4 and in Figures 5.4 through 5.12. In Table 5.3, the values obtained by the HBDKM algorithm for the *PBM* validity index for the optimal set of initial centroids *cen_opt* for each of the data sets, are presented for the demonstration of their 'goodness'. In Table 5.4, the iteration counts and average execution times of the two algorithms are compared. Due to the problem of random selection of seeds, the K-means algorithm was run for 50 trials on each of the data sets and the iteration counts and average execution times corresponding to the first best trial with optimal clustering (indicated by the lowest value of misclassification error percentage or highest value of the *PBM* index), encountered during the 50 trials, are presented in Table 5.4. The total execution times of all trials of K-means for each of the data sets are presented in Table 4.2. For the HBDKM clustering algorithm also the average execution times were obtained by executing the algorithm for 50 trials on all twelve data sets.

Both the clustering algorithms, K-means and HBDKM, obtained the same set of optimal values for the *PBM* validity index (as shown in Table 5.3 under the '**For final clustering**' column) for their corresponding optimal final clustering results for all the data sets. Only after attaining matching optimal values for the *PBM* validity index for the final clustering results obtained by the algorithms, the comparison of the number of iterations and average execution times was done between the two algorithms and those results are presented in Table 5.4. The misclassification error percentages were also obtained as same for both the algorithms for all the labeled data sets, upon attaining the equality in values of the *PBM* validity index.

Table 5.3: Comparison of values obtained for the *PBM* index by the HBDKM clustering algorithm for the optimal set of initial centroids, *cen_opt*, and for the final clustering results, for the actual number of clusters in the data sets

Data set	Actual no. of clusters	*PBM* index value obtained by HBDKM	
		For initial centroids	For final clustering
Iris	3	19.7571	25.1754
Cancer	2	6.7579	142.822
Wine	3	394819.3	473716
SODAR1	3	30908.3729	31105.1
SODAR2	3	51658.5949	56229.6
Data_3_2	3	12.4940	16.0331
Data_5_2	5	17.8383	18.7947
Data_6_2	6	624.5969	626.418
Data_9_2	9	4.9872	4.97797
Data_10_2	10	293.7252	300.15
Data_4_3	4	949.4388	941.875
Ruspini	4	22414.9014	24005

Table 5.4: Comparison of no. of iterations and average execution times of one trial of HBDKM and K-means algorithms for obtaining the final optimal clustering results, for the actual number of clusters in all the data sets

Data set	Actual no. of clusters	No. of iterations		Average execution time (in seconds)	
		HBDKM	K-means (First best trial)	HBDKM	K-means (First best trial)
Iris	3	4	7	0.0039139	0.0014647
Cancer	2	5	6	0.025527	0.0094869
Wine	3	4	8	0.027207	0.0031219
SODAR1	3	2	8	0.0022149	0.0018439
SODAR2	3	2	4	0.0023725	0.00094103
Data_3_2	3	2	4	0.0022614	0.00092164
Data_5_2	5	3	8	0.005138	0.0021571
Data_6_2	6	2	6	0.007455	0.0027303
Data_9_2	9	5	16	0.057021	0.016875
Data_10_2	10	6	9	0.030828	0.0064658
Data_4_3	4	2	6	0.0066208	0.002571
Ruspini	4	2	4	0.0029469	0.0011199

From the results presented in Tables 5.3 and 5.4 it can be inferred that the proposed HBDKM clustering algorithm is quite efficient in obtaining very good initial centroids for K-means clustering. More importantly, it obtains the seeds in a deterministic way. That is, the set of optimal seeds obtained by the HBDKM algorithm is fixed for a given data set. Further, the values of the *PBM* validity index, for both *cen_opt* and for final clustering results for a given data set, also remain the same and the number of iterations of K-means is also fixed. The HBDKM algorithm is thus fully deterministic, and the final result of clustering is also very optimal. However, the proposed approach, as can be seen from Table 5.4, can be a little more computationally costly than the best trial of the non-deterministic K-means clustering algorithm. But it should be noted that the total execution times of all the random number of trials of K-means for each data set can be much more than the execution time of one deterministic trial of HBDKM algorithm. This can be confirmed by comparing the execution times of one trail of HBDKM algorithm presented in Table 5.4 with the total execution times of all trials of K-means as presented in Table 4.2.

Figures 5.4 through 5.12 show the results obtained for the various data sets, using the HBDKM algorithm, in graphical manner. Each figure corresponding to a data set shows the optimal initial centroids and the best final clustering obtained by the HBDKM algorithm for the data set. For all the data sets in 2-dimensional and 3-dimensional space, the hyper-block partitioning is also shown. Graphical results are not shown for data sets with $d > 3$.

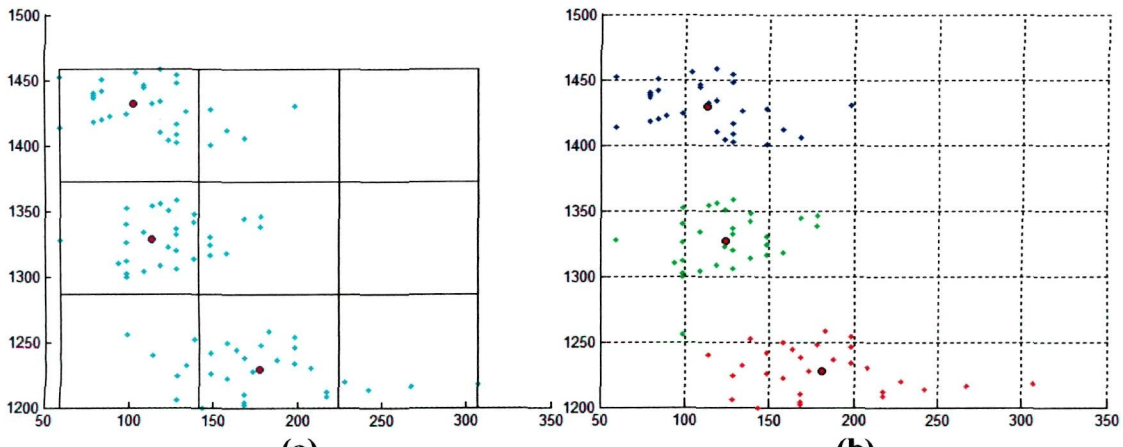


Figure 5.4: (a) Optimal initial centroids obtained for the SODAR1 data set with *bpr* = 3, (b) Final clustering obtained for SODAR1 data set

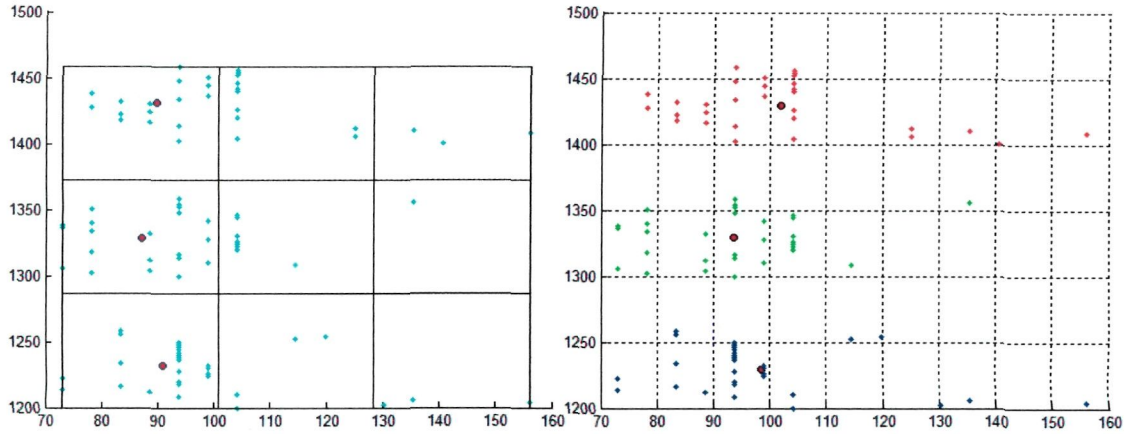


Figure 5.5: (a) Optimal initial centroids obtained for the SODAR2 data set with *bpr* = 3, (b) Final clustering obtained for SODAR2 data set

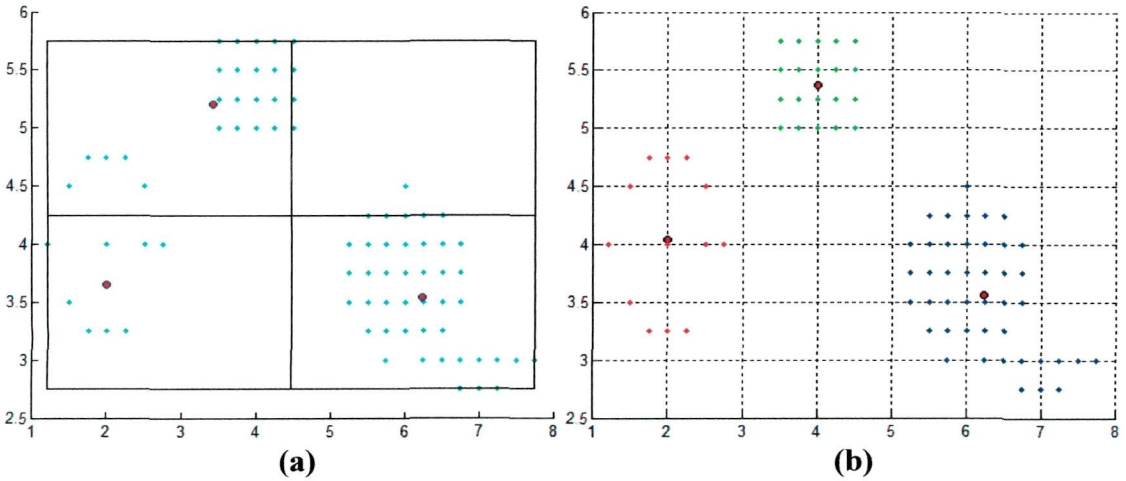


Figure 5.6: (a) Optimal initial centroids obtained for the Data_3_2 data set with *bpr* = 2, (b) Final clustering obtained for Data_3_2 data set

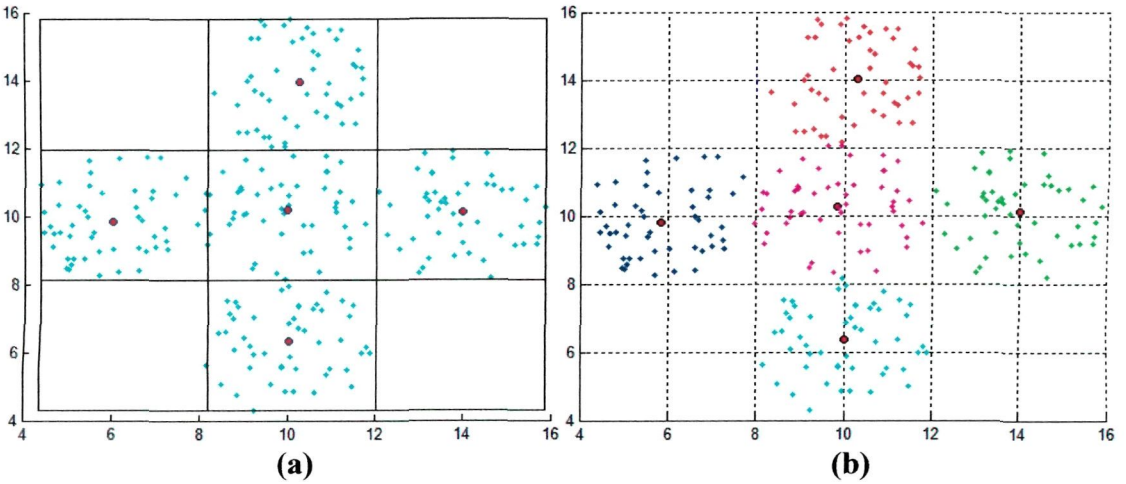


Figure 5.7: (a) Optimal initial centroids obtained for the Data_5_2 data set with *bpr* = 3, (b) Final clustering obtained for Data_5_2 data set

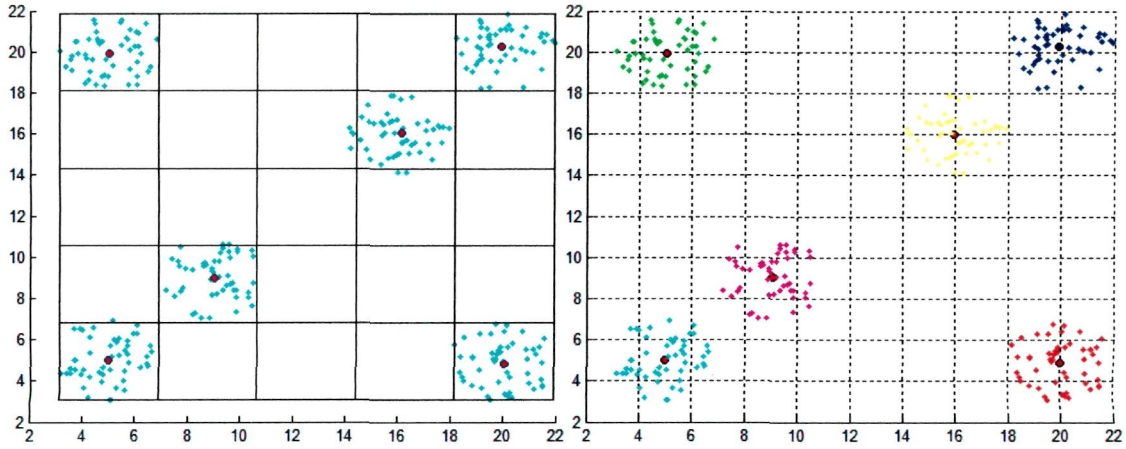


Figure 5.8: (a) Optimal initial centroids obtained for the Data_6_2 data set with *bpr* = 5, (b) Final clustering obtained for Data_6_2 data set

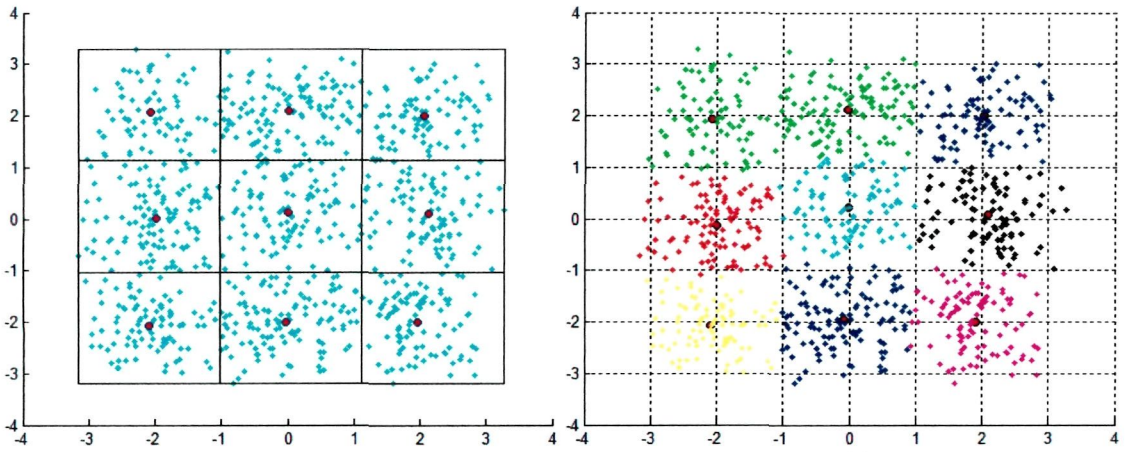


Figure 5.9: (a) Optimal initial centroids obtained for the Data_9_2 data set with *bpr* = 3, (b) Final clustering obtained for Data_9_2 data set

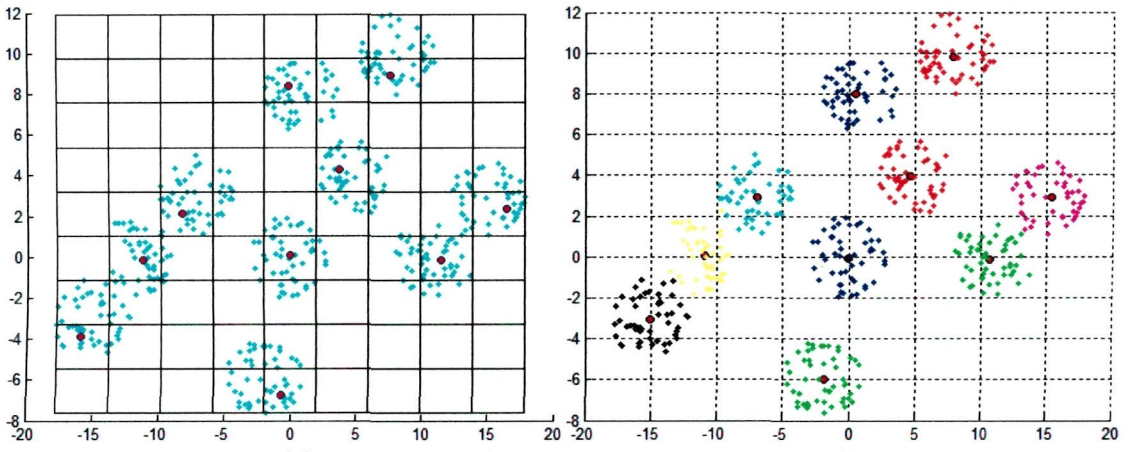
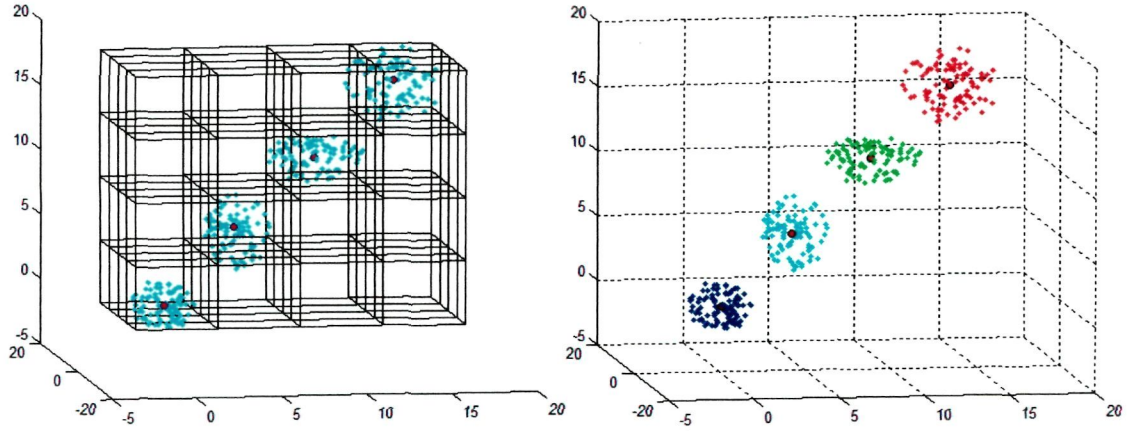
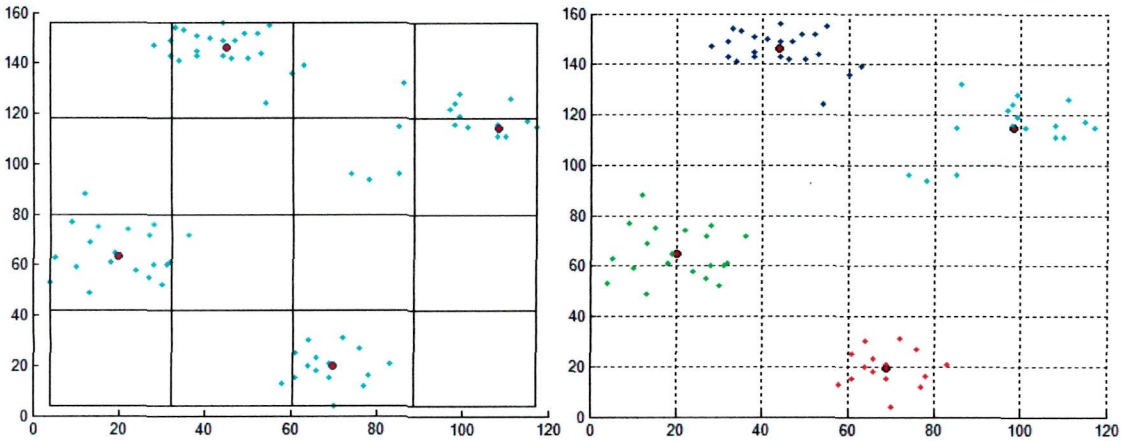


Figure 5.10: (a) Optimal initial centroids obtained for the Data_10_2 data set with *bpr* = 9, (b) Final clustering obtained for Data_10_2 data set



(a) (b)

Figure 5.11: (a) Optimal initial centroids obtained for the Data_4_3 data set with *bpr* = 4, (b) Final clustering obtained for Data_4_3 data set



(a) (b)

Figure 5.12: (a) Optimal initial centroids obtained for the Ruspini data set with *bpr* = 4, (b) Final clustering obtained for Ruspini data set

From the Figures 5.4 through 5.12 it can be inferred that the initial centroids that are obtained by the proposed HBDKM algorithm are indeed very good as in almost all of the cases the seeds were obtained quite near to the final centroids of the clusters.

To compare the time complexity of the HBDKM algorithm with K-means algorithm, first the time complexity of the K-means algorithm $O(nKdt)$ is analyzed in detail. The time complexity of one iteration of K-means can be decomposed into three parts [59]:

(i) The time required to assign the patterns to their closest clusters is $O(nKd)$.

(ii) The time required to calculate the candidate centroids is $O(nd)$.

(iii) The time required to calculate the error function is $O(nd)$.

Thus the total time complexity of one iteration of K-means is $O(n(K + 2)d)$. Since in general K-means requires at least a few iterations for convergence so in general the time complexity of one trial of K-means is expressed as $O(nKdt)$ for t iterations by omitting the constant 2.

For the time complexity of the HBDKM algorithm the following points need to be considered:

(i) The time complexity of exploring a data set to find the maximum and minimum values along each dimension is $O(n)$.

(ii) The time complexity of a d-ary division of a data set into bpr^d hyper-blocks is $O\left(\log_d bpr^d\right)$ [109].

(iii) For computing statistical information for every unit hyper-block, determining patterns belonging to each hyper-block, calculating the centroids of top K dense unit hyper-blocks, the time complexity is $O(n)$.

(iv) In Phase I of the HBDKM algorithm, the data set is partitioned for $K - \lceil\sqrt{K}\rceil + 1$ times.

(v) In Phase II of the HBDKM algorithm the K-means algorithm is executed once. One trial of K-means has the time complexity $O(nKdt)$.

Thus the total time complexity of the HBDKM algorithm turns out to be:

$$K - \lceil\sqrt{K}\rceil + 1 \times \left(O(n) + O\left(\log_d bpr^d\right) + O(n)\right) + O(nKdt)$$

For data sets with $d, bpr \ll n$, the complexity turns out to be:

$$\left(K - \lceil\sqrt{K}\rceil + 1\right) \times \left(O(n) + O(n)\right) + O(nKdt)$$
$$= O\left(\left(\left(K - \lceil\sqrt{K}\rceil + 1\right) \times 2\right)n\right) + O(nKdt)$$

The complexity of HBDKM algorithm is thus approximately linear for data sets with $d, bpr \ll n$, and is comparable with the complexity of the K-means clustering algorithm. Also, the experimental results show that the degree of certainty of good clustering from the HBDKM algorithm is very high in comparison to the non-deterministic classical K-means algorithm where there is no guarantee that in a specified number of trials, the K-means algorithm will output the most optimal clustering for a given data set.

From all the results presented in Table 5.3 and Table 5.4 and in Figures 5.4 through 5.12 and from the time complexity analysis, it can thus be inferred that the HBDKM clustering algorithm is fully deterministic and it indeed performs better that the non-deterministic hard clustering algorithm K-means. It should also be noted that hard clustering is ideally suitable to clustering of data sets with well separated and dense clusters. The results obtained for the HBDKM algorithm reinforce this fact by showing its appealing performance for such data sets. The performance of HBDKM algorithm can be further improved by using a kernel version of the *PBM* index named *kPBM* index, which is described in detail in chapter 7.

5.6 Chapter Summary

In this chapter a fully deterministic version of the widely used K-means clustering algorithm named HBDKM clustering algorithm is presented. The algorithm finds better initial centroids for clustering compared to those obtained by the K-means algorithm in a random manner. To illustrate the efficiency of the proposed algorithm it was compared with the K-means algorithm through several experiments on a number of benchmark data sets and from the experimental results it can be concluded that the HBDKM algorithm shows superior clustering performance over K-means clustering algorithm in terms of the determination of initial centroids, number of trials and iterations and total execution time.

CHAPTER 6

An Efficient Deterministic psFCM Clustering Algorithm

This chapter presents an efficient deterministic version of the psFCM clustering algorithm [111]. The chapter also presents experimental results, obtained by applying the proposed algorithm on a number of benchmark data sets, to illustrate its efficiency over the pshFCM [112], psFCM and FCM clustering algorithms.

6.1 Introduction

The Fuzzy c-means algorithm (presented in section 4.5), like its hard version the K-means algorithm, belongs to the family of local search algorithms, which searches for the solution optimum using a greedy (hill-climbing) approach. As such, the local search algorithms, like FCM, often fail to search for the global optimum. Also, the execution of FCM algorithm frequently involves the creation of a large number of membership matrices and candidate cluster matrices. The FCM algorithm is thus a computationally intensive method. The execution time of the FCM algorithm can be reduced, if a good set of initial cluster centroids is chosen, because the algorithm will take less number of iterations to find the actual cluster centroids. It is however difficult to select a good set of initial cluster centroids randomly [112]. Thus, in this chapter, an efficient deterministic fuzzy clustering algorithm is presented that obtains better initial centroids through partitioning of the data set using a variation of the *k*-d tree space-partitioning data structure [113]. Like the HBDKM algorithm described in chapter 5, the proposed fuzzy clustering algorithm, presented in this chapter, also

obtains the initial centroids in a deterministic way. Thus the overall clustering obtained by the proposed algorithm is also deterministic. The algorithm is presented in detail in section 6.4. In section 6.2, a brief outline of some related work on seed initialization of FCM is presented. In section 6.3, the *k*-d tree data structure is explained along with its variants. In section 6.5 experimental results for enunciating the efficiency of the proposed approach is presented. Finally, in section 6.6, a summary of the chapter is provided.

6.2 Related Work

The FCM clustering algorithm is the most widely used and researched approach in both theoretical and practical applications of fuzzy logic to data clustering. Many FCM based clustering algorithms have been proposed in the literature. Cheng et al. [114] proposed a multistage random sampling FCM algorithm. It is based on the assumption that a small subset of a data set of feature vectors can be used to approximate the cluster centroids of the complete data set. With this assumption, FCM is used to compute the cluster centroids of a small randomly selected subset of the original data set. After obtaining the cluster centroids of this small subset, the subset is merged with an additional small, randomly selected subset of the remaining unprocessed feature vectors to form a large subset to be processed by FCM. The previously calculated cluster centroids are used for the initialization of the fuzzy partition matrix of the newly formed set. The above procedure is repeated until the size of the feature vector matrix used in calculations is large enough to approximate the actual cluster centroids of the full data set. The resulting cluster centroids are then used for the initialization of the fuzzy partition matrix for FCM when it is applied to the original data set. Hung and Yang [111] proposed an efficient FCM clustering algorithm called psFCM (*partition simplification* FCM) algorithm. It is divided into two phases. In Phase I, the dataset is first divided into some hyper-blocks using the *k*-d tree data structure. All patterns in a hyper-block are then replaced by the centroids of the patterns. In this way the original dataset is drastically cut down to a simplified dataset comprising of small number of hyper-blocks' centroids. The FCM algorithm is

then applied on this simplified dataset to find the actual initial cluster centroids for the complete dataset. In Phase II, the FCM algorithm is applied on the complete data set using the optimal seeds obtained from Phase I. Begum et al. [112] also presented a *k*-d tree based divisive technique for seed initialization for the FCM clustering algorithm. The approach has three phases. Phase I partitions the data set into several hyper-blocks using the *k*-d tree partitioning method and then computes the centroids of all the obtained hyper-blocks using an approach similar to that in Phase I of [111]. In Phase II a set of *c* centroids are selected at random from the set of all computed centroids obtained in Phase I and are evaluated using the popular, fuzzified *PBM* validity index, *PBMF*. This process is continued for a specific number of times for different randomly selected sets and the set of seeds corresponding to the highest value of *PBMF* index is chosen as the optimal set of seeds for clustering the complete data set. Using these seeds, the FCM algorithm is then applied on the complete data set, to obtain the final clustering in Phase III.

6.3 The *k*-d tree Data Structure

The *k*-d tree data structure is a popular space-partitioning data structure for organizing points in a *k*-dimensional space. It is a data structure that subdivides the hyperspace into hyper-rectangular cells (called here as hyper-blocks) through the recursive application of some *splitting rule*. The choice of splitting rule affects the shape of hyper-blocks and the structure of the resulting tree. The *k*-d tree is a generalization of the simple 1-dimensional binary search tree in which every node is associated with one of the *k* dimensions. In a *k*-d tree, every non-leaf node of the tree can be thought of as implicitly generating a splitting hyperplane that divides the hyperspace of *k*-dimensional patterns into two parts, known as subspaces. Points to the left of this hyperplane represent the left sub-tree of that node and points to the right of the hyperplane represent the right sub-tree. The root node of the tree represents the entire *k*-dimensional data set [113] [115] [116] [117]. The idea of *k*-d trees was first introduced by Bentley [113] in 1975. As per the original definition, if a *k*-dimensional data set is represented as a *k*-d tree, then each pattern in the data set is

stored as a node in the tree. Each node contains two pointers, which are either null or point to another node in the *k*-d tree. Each pointer can be considered as specifying a sub-tree. Each node at a particular depth in the tree is associated with a discriminator dimension (not necessarily stored as a field in the node), which is an integer in the interval [1, *k*] corresponding to the *k* dimensions. The original notion of a *k*-d tree was however modified by Friedman et al. in [116] by introducing a *k*-d tree which stores data only in the leaf nodes, often as small collections of patterns from the data set called buckets (here hyper-blocks). The two notions were later termed by Bentley as a homogeneous and a nonhomogeneous *k*-d tree respectively, in [117]. Thus a homogenous *k*-d tree is a binary tree in which every node stores a *k*-dimensional data pattern from the data set whereas a nonhomogenous *k*-d tree is a binary tree which stores data points only in the leaf nodes, often as collections of several *k*-dimensional points from the data set, in hyper-blocks. A non-homogeneous *k*-d tree is also sometimes referred to as a *k*-d trie or a pseudo *k*-d tree [118].

Given a list of *n*, *k*-dimensional points, the following algorithm will construct a *k*-d tree containing those points.

Table 6.1: Procedure to construct a *k*-d tree

```
procedure split (pointList, depth)
  // as per some splitting rule select a discriminator dimension and return a
  // discriminator value for that dimension (usually median or midpoint)
end split
procedure kdtree (pointList, depth)
  if pointList is empty
    return null;
  else
    discriminator = split(pointList, depth)
    // Create a tree node, node and construct sub-trees
    node.discriminator = discriminator;
    node.leftChild = kdtree(points in pointList less than discriminator, depth+1);
    node.rightChild = kdtree(points in pointList greater than or equal to
                             discriminator, depth+1);
    return node;
  end if
end kdtree
```

The procedure **split** has been defined in several ways since its first definition by Bentley in [113]. In the next subsection some popular splitting rules are presented

including the original splitting rule presented in [113] by Bentley.

6.3.1 Splitting Rules

In this subsection, some well-known splitting rules for the construction of the *k*-d tree space partitioning data structure are presented. The splitting rule used for the construction of *k*-d tree structure by the proposed algorithm of this chapter, is also presented. A splitting rule obtains the discriminator dimension, i.e., the dimension which should be split into two subspaces, and the discriminator value *discriminator* for each node in a *k*-d tree.

The original split rule: The original splitting rule for *k*-d tree construction proposed by Bentley [113] chooses the discriminator dimension for each node on the basis of its depth in the tree. The root node is defined to be at *depth* 0. The discriminator dimension is calculated as *depth* mod $k + 1$; this formula results in cycling through the *k* dimensions. The discriminator value for a chosen discriminator dimension is given by the coordinate median of the points in that dimension.

Standard split: Instead of cycling through the dimensions, this rule selects the discriminator dimension for each node to be the one for which the current *pointList* have the maximum spread value (difference between the maximum and minimum values along the dimension). The discriminator value is chosen to be the coordinate median of the points in that dimension. Friedman, Bentley and Finkel introduced this splitting rule in their definition of the optimized *k*-d tree in [116]. This rule is now referred to as the standard splitting rule.

Midpoint split: As per this rule, the splitting hyperplane passes through the center of the hyper-block and bisects the longest side of the hyper-block. If there are many sides of equal length, any one may be chosen first, say the one with the lowest coordinate index. The rule differs from the standard split in the way that it uses the midpoint, rather than the median, as the *discriminator*. Thus the splitting hyperplane need not always pass through a data point [115].

Sliding-midpoint split: In this rule, first a midpoint split is attempted, by considering a hyperplane passing through the center of the hyper-block and bisecting the hyper-block's longest side. If the data points lie on both sides of the splitting

hyperplane then the splitting hyperplane remains here. However, if all the data points lie to one side of the splitting hyperplane, then splitting hyperplane 'slides' towards the data points until it encounters the first point. A child leaf hyper-block containing this single point is created, and the algorithm reiterates on the remaining points. In all splits the rule uses the midpoint, rather than the median, as the *discriminator* [115].

Hybrid split: This rule is a hybridization of the original split rule, the standard split rule and the midpoint split rule. According to this rule, initially a sorted one dimensional array *disc_dim_nos* of the k dimension numbers, 1 to k, is computed by sorting the numbers in decreasing order according to the spread values (difference between the maximum and minimum values along a dimension) of the k dimensions for the complete data set. The rule then selects the discriminator dimension numbers for the nodes at various depths by cycling through the k dimension numbers from the sorted list *disc_dim_nos*. At a particular depth, *depth*, the index of the discriminator dimension number in *disc_dim_nos* is given by *depth* mod $k + 1$. After the selection of the discriminator dimension the rule continues with the splitting using the midpoint split rule. The root node is defined to be at *depth* 0.

From the description it can be easily seen that the hybrid split rule is designed to be computationally fast by:

- restricting the spread value calculation to only one instance, unlike that in the standard, midpoint and sliding-midpoint rules which require computation after each split for choosing the next dimension to be split.
- using the simple rule for choice of the discriminator dimension based on *depth* from *disc_dim_nos*, similar to that of the original split rule.
- avoiding the costly calculation of mean and using the midpoint instead.

The proposed algorithm presented in this chapter thus uses this hybrid split rule.

6.4 The proposed DpsFCM Clustering Algorithm

In this section the proposed DpsFCM (*deterministic* psFCM) clustering algorithm is presented. The proposed DpsFCM algorithm consists of two phases which are analogous to that of the HBDKM algorithm discussed in chapter 5. Phase I is the

initialization phase that involves the fast and deterministic computation of seeds and checking of their fitness. Phase II is the clustering phase in which the FCM algorithm is applied only 'once' on the whole data set using the optimal set of centroids obtained in Phase I. The complete algorithm is now presented next:

Phase I

Step 1): Obtain the number of clusters c.

Step 2): Set maximum depth of partitioning, $maxd = \lceil \log_2(c) \rceil + 1$.

Step 3): Compute the spread values (difference between the maximum and minimum values along a dimension) of the k dimensions for the complete data set. Create the sorted one dimensional array *disc_dim_nos* of the k dimension numbers, 1 to k, by sorting the numbers in decreasing order according to the spread values.

Step 4): Partition the dataset X by simulating the creation of a nonhomogeneous k-d tree up to the maximum depth *maxd*, starting at *depth* 0, via hybrid split, using the array *disc_dim_nos*, so that an $M \times (k+1)$ matrix T is obtained. The matrix T contains M ($\leq 2^{maxd}$) centroids $\bar{\mathbf{x}}_m$'s and the M counts of patterns n_m's for all the M non-empty hyper-blocks generated temporarily in the process of partitioning of X into a k-d tree with M leaf nodes. Each leaf node represents a hyper-block of k-dimensional patterns from the original data set. (Procedure given in Table 6.2).

If $M \geq c$ then

Step 5): Sort the M centroids in T, in decreasing order according to the values n_m's of the densities of their corresponding hyper-blocks and choose top c centroids and store them in optimal centroid set *cen_opt*1.

Step 6): Obtain another set of optimal initial centroids from T using a procedure described in Table 6.3 and store it in optimal centroid set *cen_opt*2. [In each iteration of the procedure, it calculates the pair-wise distances of each pair of centroids present in T, sorts the distances in increasing order, computes the mean of two closest centroids in the list of centroids and removes the two centroids. It then adds the mean of the

centroids as a new centroid in the list of centroids. After M - c iterations the optimal centroid set *cen_opt*2 is obtained].

Step 7): Evaluate the value of the fitness measure, the fuzzy *PBM* validity index, *PBMF* (described in section 4.6.2) for the optimal centroid set *cen_opt*1 and store the value obtained in *pbmfval*1.

Step 8): Evaluate the value of the fuzzy *PBM* validity index, *PBMF* for the optimal centroid set *cen_opt*2 and store the value obtained in *pbmfval*2.

Step 9): Compare the two values *pbmfval*1 and *pbmfval*2 and select the optimal centroid set corresponding to the larger of the two values as the final optimal centroid set *cen_opt*.

Else set *cen_opt* as an empty set and terminate

Phase II

Step 10): Apply FCM algorithm on the original data set only once, using the optimal set of centroids *cen_opt* obtained in Step 9 and output the result of clustering.

As can be seen from the algorithm, Step 1 of Phase I obtains the value of c. Step 2 calculates the maximum depth of partitioning *maxd*. Step 3 obtains the sorted array *disc_dim_nos*, by sorting the k dimensions numbers in decreasing order according to the spread values. In Step 4, the hyper-volume of the data set X is partitioned using the procedure described in Table 6.2 as a result of which the original dataset X is drastically cut down to a very simplified form T comprising of only a small number ($\leq 2^{maxd}$) of hyper-blocks' centroids. In Step 5, the centroids in T are sorted according to the densities of the M hyper-blocks and the c centroids corresponding to the top c dense hyper-blocks are stored in the optimal centroid set *cen_opt*1. In Step 6, another optimal centroid set *cen_opt*2 is obtained using the procedure described in Table 6.3. In steps 7 and 8 the *PBMF* index values for the two centroid sets *cen_opt*1 and *cen_opt*2 are calculated and in Step 9, the centroid set corresponding to the higher value of the *PBMF* validity index is chosen as the optimal centroid set *cen_opt*. Step 5 through Step 9 are executed only if the number of leaf nodes M (the number of rows in T) is greater than or equal to c else the algorithm sets *cen_opt* to an empty matrix and terminates.

Table 6.2: Procedure to partition a data set X by simulating the creation of a nonhomogeneous k-d tree up to a maximum depth *maxd* starting at *depth* = 0

```
Procedure kdtreepart (X, depth, maxd, disc_dim_nos)
  // Calculate the number of rows n and number of columns k, of X
  if n == 0
    T = an empty matrix
    return T;
  elseif n >1 and depth < maxd
    // Select the discriminator dimension based on depth by cycling through all
    // the values in disc_dim_nos. Select the discriminator value as the midpoint
    disc_dim = disc_dim_nos(depth mod k + 1)
    // Find the maximum and minimum values along the dimension disc_dim
    max_disc_dim = the maximum value of dimension X(:, disc_dim)
    min_disc_dim = the minimum value of dimension X(:, disc_dim)
    discriminator = (min_disc_dim + max_disc_dim)/2
    X1 = points in X less than discriminator
    X2 = points in X greater than or equal to discriminator
    // Calculate the number of rows s1 of X1 and the number of rows s2 of X2
    if s1 > 0
      T1 = kdtreepart(X1, depth+1, maxd, disc_dim_nos);
    end if
    if s2 > 0
      T2 = kdtreepart(X2, depth+1, maxd, disc_dim_nos);
    end if
    // Calculate the number of rows st1 of T1 and the number of rows st2 of T2
    if st1 > 0 and st2 > 0
       T= a matrix obtained by concatenating T1 and T2
    elseif st1 > 0 and st2 == 0
       T=T1;
    elseif st1 == 0 and st2 > 0
       T=T2;
    end if
    return T;
  else
    if n >1
      T(1, 1:k) = mean of X
    else
      T(1, 1:k) = X
    end if
    T(1, k +1) = n
    return T
  end if
end kdtree
```

In Step 10 of Phase II the FCM clustering algorithm is applied on the complete data set by initializing the pseudopartition matrix of the FCM algorithm using the optimal initial centroids *cen_opt* obtained in Step 9 of Phase I.

Table 6.3: Procedure to obtain the optimal centroid set *cen_opt2* from the $M \times (k+1)$ matrix *T*

```
procedure optcenset2 (T)
  / / Initialize the matrix of optimal centroid set cen_opt2
  cen_opt2 = T(:,1:k) / / an M × k matrix
  dif = M - c
  for i = 1, ..., dif do
    / / Calculate the number of rows ncen in the matrix cen_opt2
    / / Create a matrix dist_cen with ncen(ncen-1)/2 rows and 3 columns
    ind = 1
    for ii = 1, ..., ncen do
      for j = 1, ..., ncen do
        if ii < j
          dist_cen(ind, 1) = ii
          dist_cen(ind, 2) = j
          dist_cen(ind, 3) = Euclidean distance between cen_opt2(ii, :)
                             and cen_opt2(j, :)
          ind = ind + 1
        end if
      end for
    end for
    / / Sort the rows of dist_cen in ascending order according to the distance
    / / values in the 3rd column, of the centroid pairs
    cen1 = cen_opt2(dist_cen(1, 1), :) / / Top two closest centroids are
    cen2 = cen_opt2(dist_cen(1, 2), :) / / stored in cen1 and cen2
    / / Delete the two centroids cen1 and cen2 from the matrix cen_opt2
    cen = mean of the centroids, cen1 and cen2
    cen_opt2 = a matrix obtained by concatenating cen_opt2 and cen
  end for
  return cen_opt2
end optcenset2
```

It can now be seen that the proposed DpsFCM algorithm involves no randomness, as the choice of good seeds is done deterministically rather than resorting to random techniques. The algorithm generates only two candidate sets of seeds and chooses the better one by evaluating the efficient *PBMF* validity index, thus the quality of clustering is also ensured. It needs to apply FCM algorithm on the full data

set only once rather than for an uncertain number of times. The average execution time taken by the whole process is thus comparatively quite less. The final outcome of clustering by the DpsFCM algorithm is also highly optimal.

6.5 Experimental Results and Discussion

To demonstrate the better performance of the proposed DpsFCM algorithm over the pshFCM, psFCM and FCM algorithms, a comparative study was carried out by applying the four algorithms on all the twelve benchmark data sets described in subsection 4.7.1. For all the algorithms the validation of clustering was done using the efficient *PBMF* validity index. For the sake of uniform comparison the same partitioning procedure, presented in Table 6.2, was also used for the pshFCM and psFCM algorithms. The experimental results obtained using the algorithms are presented in Tables 6.4 through 6.8 and in Figures 6.1 through 6.9. In Table 6.4, the values obtained by the four algorithms for the *PBMF* validity index, corresponding to their optimal set of initial centroids, are compared and in Table 6.5, the values obtained by the four algorithms for the *PBMF* validity index, corresponding to their final optimal clustering results, are compared. In Table 6.6 and Table 6.7, the iteration counts and average execution times (respectively) of the four algorithms are compared. Due to the non-deterministic nature of the pshFCM, psFCM and FCM algorithms, the algorithms were run for 50 trials on each of the data sets. For FCM, the iteration counts and average execution times corresponding to the first best trial with optimal clustering (indicated by the lowest value of misclassification error percentage or highest value of the *PBMF* index), encountered during the 50 trials, are presented and for the pshFCM and psFCM algorithms the iteration counts and average execution times corresponding to the first best trial with optimal seeds (indicated by the highest value of the *PBMF* index), encountered during the 50 trials, are presented. The total execution times of all trials for the pshFCM, psFCM and FCM algorithms for each of the data sets are presented in Table 6.8. For the DpsFCM clustering algorithm also, the average execution times (as presented in Table 6.7) were obtained by executing the algorithm for 50 trials on all twelve data sets.

Table 6.4: Comparison of values obtained for *PBMF* validity index by DpsFCM, pshFCM and psFCM clustering algorithms for their corresponding optimal set of initial centroids, for the actual number of clusters in all data sets

Data set	Actual no. of clusters	*PBMF* index value obtained for initial centroids by		
		DpsFCM	pshFCM	psFCM
Iris	3	22.5997	24.8601	13.8843
Cancer	2	98.8271	176.7426	92.0861
Wine	3	293386.9594	331575.3169	278771.5960
SODAR1	3	29198.8489	35213.9468	28361.3310
SODAR2	3	56576.6911	56576.6911	30472.2335
Data_3_2	3	17.4287	17.9150	16.3502
Data_5_2	5	22.9278	23.7360	22.8135
Data_6_2	6	619.6013	554.0484	619.6217
Data_9_2	9	6.853	7.2971	6.7458
Data_10_2	10	308.7863	305.6905	310.3326
Data_4_3	4	954.9403	954.9403	800.6757
Ruspini	4	21278.9258	24463.3192	21880.5954

Table 6.5: Comparison of values obtained for *PBMF* validity index by DpsFCM, pshFCM and psFCM clustering algorithms for final clustering results, for the actual number of clusters in all data sets

Data set	Actual no. of clusters	*PBMF* index value obtained for final clustering by		
		DpsFCM	pshFCM	psFCM
Iris	3	28.2205	28.2205	28.2204
Cancer	2	167.964	167.964	167.964
Wine	3	553537	553537	553537
SODAR1	3	34745.1	34745	34745.1
SODAR2	3	58914.4	58914.4	58914.3
Data_3_2	3	16.6513	16.6513	16.6513
Data_5_2	5	22.9332	22.9332	22.9332
Data_6_2	6	638.347	638.347	638.347
Data_9_2	9	6.78327	6.78327	6.78327
Data_10_2	10	345.917	345.917	345.918
Data_4_3	4	960.79	960.79	960.79
Ruspini	4	25196.7	25196.7	25196.7

The misclassification error percentages were obtained as same for all the four algorithms. For all the four algorithms and for the *PBMF* index, m was chosen as 1.5.

Table 6.6: Comparison of iteration counts of one deterministic trial of DpsFCM with those for the first best trials of pshFCM, psFCM and FCM algorithms for obtaining the final clustering for the actual number of clusters in all the data sets

Data set	Actual no. of clusters	No. of iterations			
		DpsFCM	pshFCM	psFCM	FCM
Iris	3	17	19	25	27
Cancer	2	9	10	9	11
Wine	3	25	28	35	26
SODAR1	3	8	10	9	14
SODAR2	3	5	5	4	10
Data_3_2	3	5	6	4	11
Data_5_2	5	22	22	21	25
Data_6_2	6	4	5	4	11
Data_9_2	9	23	27	20	38
Data_10_2	10	14	16	14	27
Data_4_3	4	4	4	4	9
Ruspini	4	6	6	5	7

Table 6.7: Comparison of average execution times of one trial of DpsFCM, pshFCM, psFCM and FCM algorithms for obtaining the final optimal clustering results, for the actual number of clusters in all the data sets

Data set	Actual no. of clusters	Average execution time (in seconds)			
		DpsFCM	pshFCM	psFCM	FCM
Iris	3	0.010108	0.014212	0.012593	0.011445
Cancer	2	0.017161	0.030664	0.015378	0.014367
Wine	3	0.017368	0.02183	0.0197	0.013246
SODAR1	3	0.0049287	0.007666	0.0051118	0.0047126
SODAR2	3	0.0041282	0.0050324	0.0038854	0.0036931
Data_3_2	3	0.0040322	0.0039549	0.0031309	0.0028061
Data_5_2	5	0.02786	0.032812	0.026412	0.023696
Data_6_2	6	0.014587	0.0091864	0.0059684	0.014881
Data_9_2	9	0.16861	0.21524	0.14172	0.2169
Data_10_2	10	0.084626	0.080755	0.059425	0.095032
Data_4_3	4	0.010908	0.019866	0.00755	0.011454
Ruspini	4	0.0045701	0.006333	0.0035463	0.0029079

Table 6.8: Comparison of total execution times of 50 trials for pshFCM, psFCM and FCM algorithms, for the actual number of clusters in all the data sets

Data set	Actual no. of clusters	Total execution time (in seconds)		
		pshFCM	psFCM	FCM
Iris	3	0.73454	0.56267	0.57225
Cancer	2	1.5056	0.7689	0.77189
Wine	3	1.0611	0.98498	0.78153
SODAR1	3	0.37563	0.26468	0.21005
SODAR2	3	0.33214	0.19427	0.17542
Data_3_2	3	0.33485	0.15655	0.28596
Data_5_2	5	1.8181	1.3231	1.3952
Data_6_2	6	3.4853	1.98	1.6639
Data_9_2	9	15.5528	10.2675	11.8722
Data_10_2	10	8.3531	5.9341	6.1877
Data_4_3	4	1.182	0.56814	0.81325
Ruspini	4	0.34937	0.1986	0.25091

Figures 6.1 through 6.9 show the results obtained by the DpsFCM algorithm, in graphical manner. Each figure corresponding to a data set shows the optimal initial centroids and the best final clustering obtained by the algorithm for the data set. For all the data sets in 2-dimensional and 3-dimensional space, the *k*-d tree partitioning is also shown. Graphical results are not shown for data sets with $d > 3$

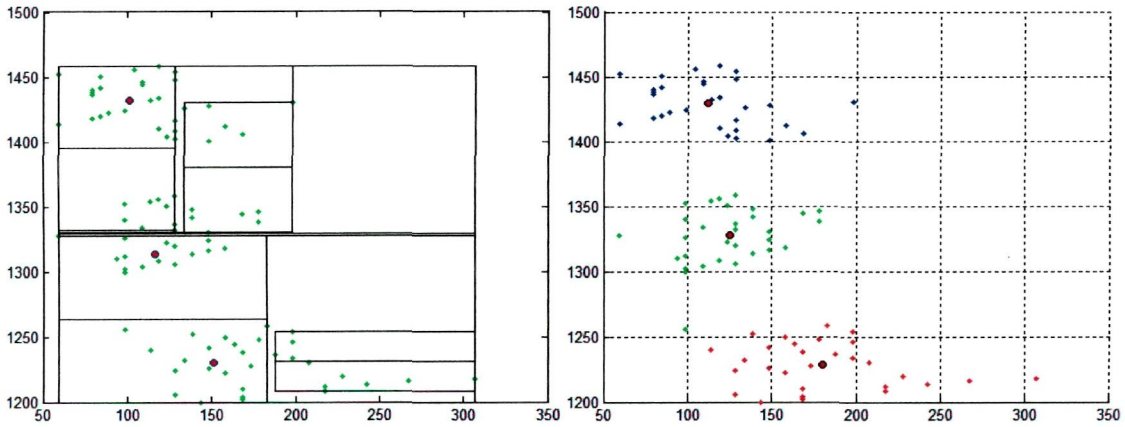


Figure 6.1: (a) Optimal initial centroids obtained for the SODAR1 data set with *maxd* = 3, (b) Final clustering obtained for SODAR1 data set

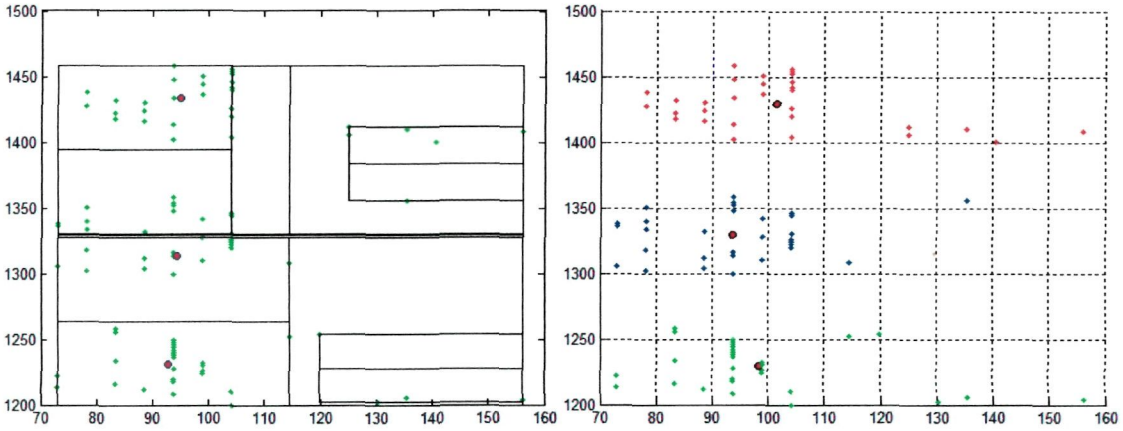


Figure 6.2: (a) Optimal initial centroids obtained for the SODAR2 data set with *maxd* = 3, (b) Final clustering obtained for SODAR2 data set

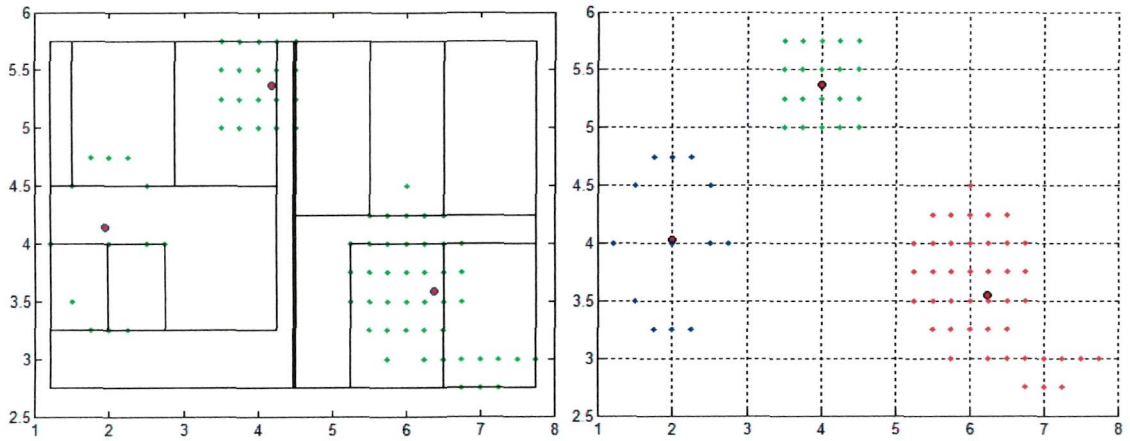


Figure 6.3: (a) Optimal initial centroids obtained for the Data_3_2 data set with *maxd* = 3, (b) Final clustering obtained for Data_3_2 data set

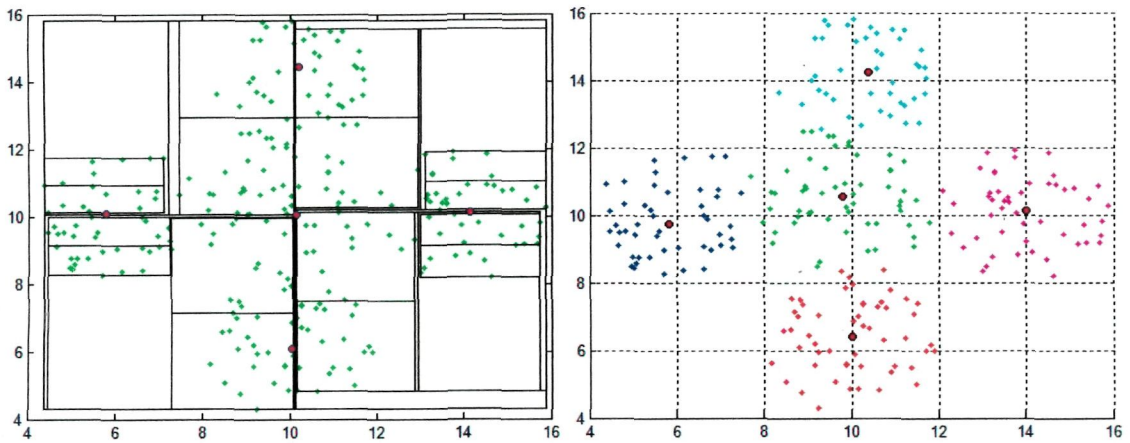
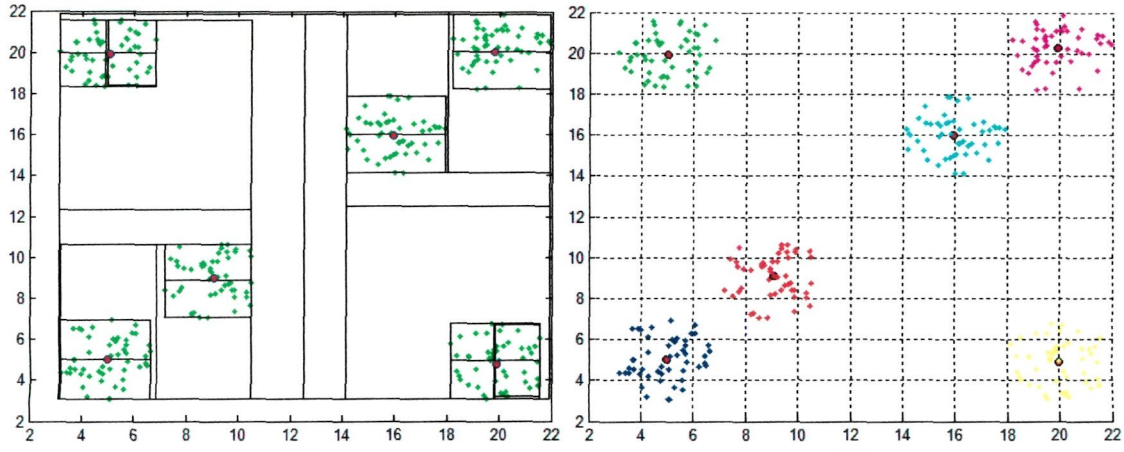
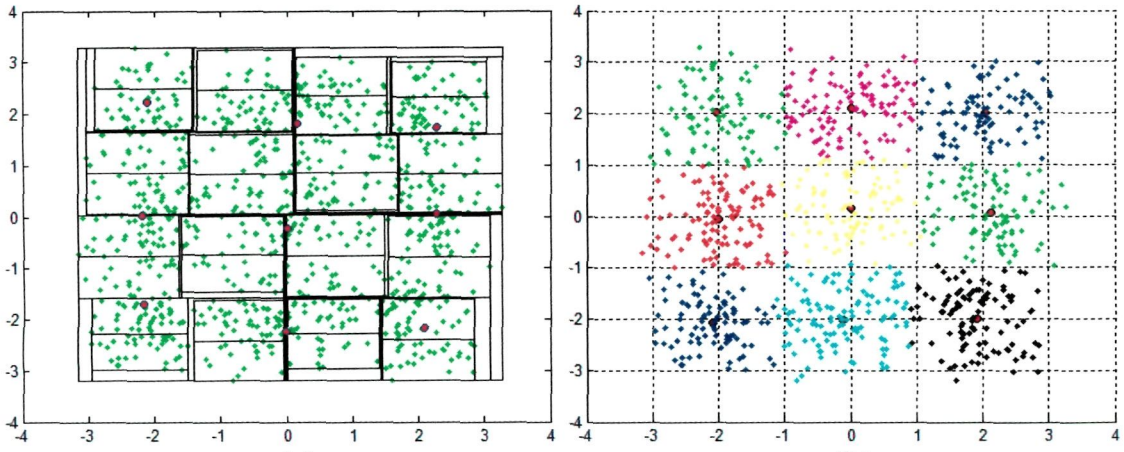


Figure 6.4: (a) Optimal initial centroids obtained for the Data_5_2 data set with *maxd* = 4, (b) Final clustering obtained for Data_5_2 data set



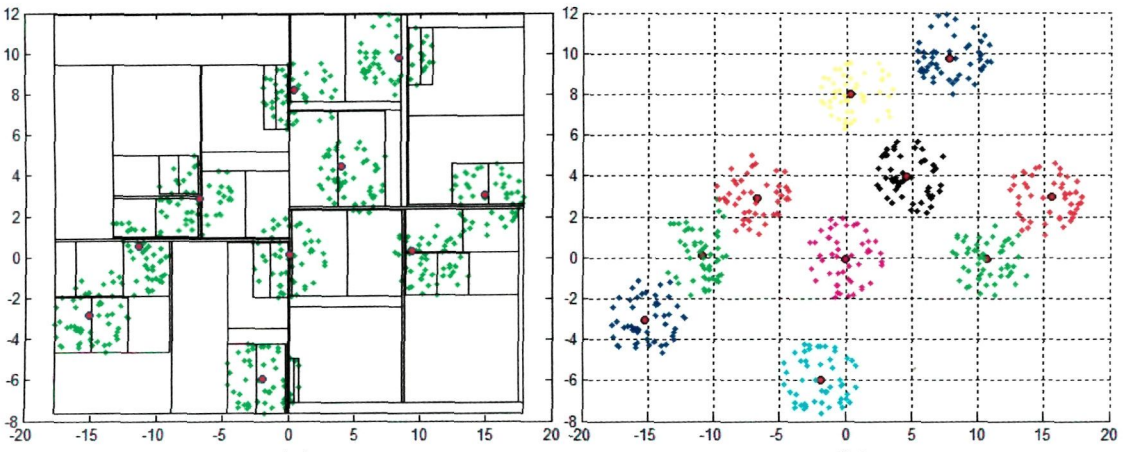
(a) **(b)**

Figure 6.5: (a) Optimal initial centroids obtained for the Data_6_2 data set with *maxd* = 4, (b) Final clustering obtained for Data_6_2 data set



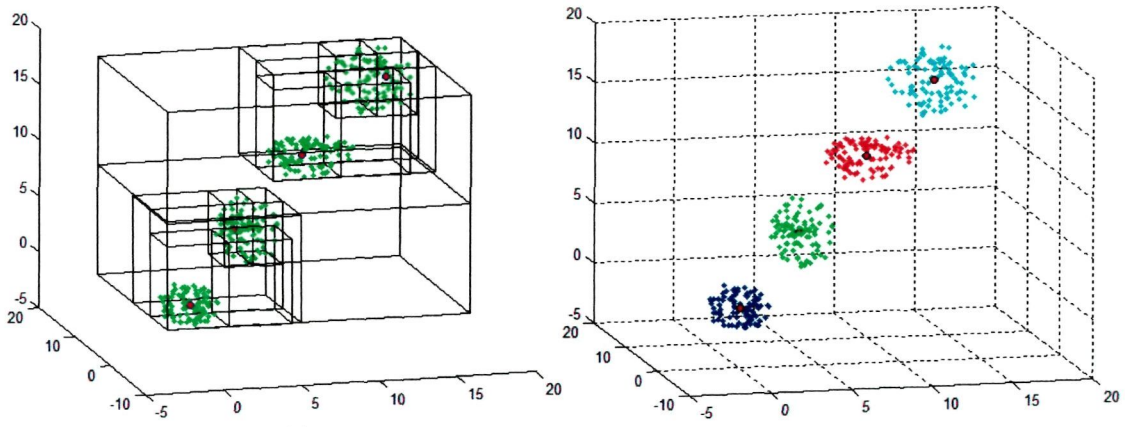
(a) **(b)**

Figure 6.6: (a) Optimal initial centroids obtained for the Data_9_2 data set with *maxd* = 5, (b) Final clustering obtained for Data_9_2 data set



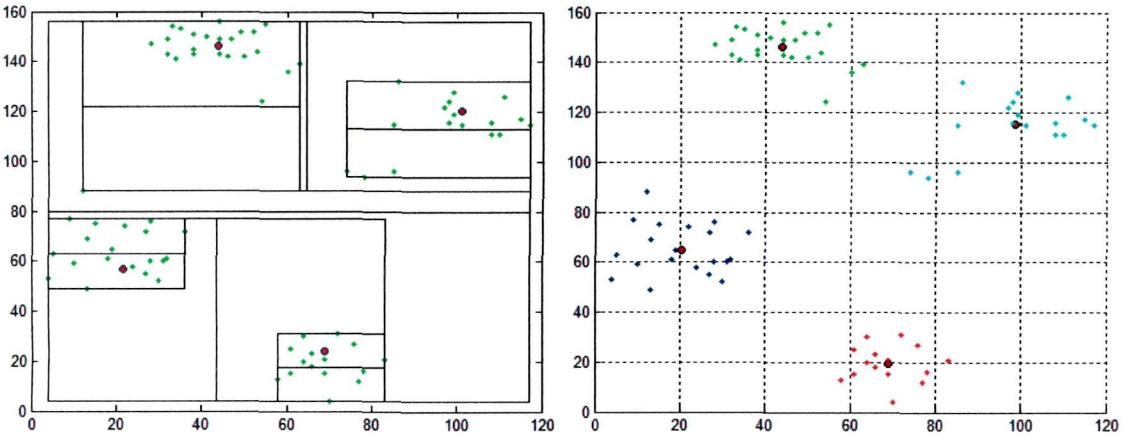
(a) **(b)**

Figure 6.7: (a) Optimal initial centroids obtained for the Data_10_2 data set with *maxd* = 5, (b) Final clustering obtained for Data_10_2 data set



(a) (b)

Figure 6.8: (a) Optimal initial centroids obtained for the Data_4_3 data set with *maxd* = 3, (b) Final clustering obtained for Data_4_3 data set



(a) (b)

Figure 6.9: (a) Optimal initial centroids obtained for the Ruspini data set with *maxd* = 3, (b) Final clustering obtained for Ruspini data set

From the results presented in Table 6.4, it can be clearly inferred that the proposed DpsFCM algorithm is quite efficient in obtaining very good initial centroids for FCM clustering compared to the non-deterministic psFCM and pshFCM clustering algorithms. Figures 6.1 through 6.9 clearly show that the initial centroids that are obtained by the proposed DpsFCM algorithm are indeed very good and in almost all of the cases the seeds are obtained quite near to the final centroids of the actual clusters. It is important to note that the DpsFCM algorithm obtains the seeds in a deterministic way unlike the psFCM and pshFCM algorithms. Thus, the set of optimal seeds obtained by the DpsFCM algorithm is fixed for each data set. As such, for the DpsFCM algorithm, the values of the *PBMF* validity index, for both *cen_opt* and for final clustering results for a given data set, also remain the same and the number of

iterations of the only FCM trial required, is also fixed. The DpsFCM algorithm is thus fully deterministic, and the final result of clustering is also very optimal (as shown in Table 6.5). From Table 6.6 it can be easily inferred that the numbers of iterations required by the DpsFCM algorithm are also very optimal and are comparable to the iteration counts of best trials of pshFCM, psFCM and FCM algorithms. For the non-deterministic algorithms, pshFCM, psFCM and FCM, the iteration counts presented in Table 6.6 can frequently vary due to their inherent random nature, but for DpsFCM, the iteration counts are fixed. From Table 6.7, it can be seen that the DpsFCM algorithm is comparable in terms of average execution time with the non-deterministic pshFCM, psFCM and FCM clustering algorithms. For some data sets like Iris, Data_9_2 and Data_4_3, the DpsFCM algorithm executes much faster because of the requirement of lesser number of iterations for convergence. For other data sets, one deterministic trail of the DpsFCM algorithm takes slightly more time than the other algorithms. However, from Table 6.8 it can be seen that the total execution times of all the random number (here 50) of trials required for obtaining the best clustering by each of non-deterministic algorithms pshFCM, psFCM and FCM, for each of the data sets, can be much more than the average execution time of one deterministic trial of the proposed DpsFCM clustering algorithm.

For obtaining the time complexity of the DpsFCM clustering algorithm, the following main time complexities need to be considered:

(i) The time complexity of exploring a data set to find the maximum and minimum values along each dimension is $O(n)$.

(ii) The time complexity of creating the array *disc_dim_nos* is $O(k \log k)$.

(iii) The time complexity of partitioning of a data set by building a k-d tree is $O(kn \log n)$ [116].

(iv) The time complexity of computing the first set of seeds is $O(c \log c)$.

(v) The time complexity of computing the second set of seeds is $O(c^3)$.

(vi) In Phase II of the DpsFCM algorithm the FCM algorithm is executed once. One trial of FCM has the time complexity $O(nckt)$.

Thus the total time complexity of the DpsFCM algorithm turns out to be:

$$O(n)+O(k\log k)+O(kn\log n)+O(c\log c)+O(c^3)+O(nckt)$$

For data sets with $k, c \ll n$, the complexity turns out to be:

$$O(n) + O(k\, n \log n) + O(n\, c\, k\, t)$$

The DpsFCM algorithm is thus slightly costly than the FCM algorithm, but is comparable in complexity with the other two algorithms, pshFCM and psFCM, which also use the *k*-d partitioning technique. The experimental results however, enunciate that the degree of certainty of good clustering is very high for the DpsFCM algorithm compared to all the other non-deterministic algorithms, pshFCM, psFCM and FCM, where there is no guarantee that in a specified number of trials, the algorithms will output the most optimal clustering for a given data set.

From all the results presented in Tables 6.4 through 6.8 and Figures 6.1 through 6.9 and from the time complexity analysis, it can thus be inferred that the DpsFCM deterministic clustering algorithm indeed performs better than the other non-deterministic clustering algorithms, with which it was compared. The performance of DpsFCM algorithm can be further improved by using a kernel version of the *PBMF* validity index named *kPBMF* index, which is described in detail in chapter 7.

6.6 Chapter Summary

In this chapter a fully deterministic version of the psFCM clustering algorithm is presented. The algorithm finds better initial centroids for clustering compared to the pshFCM, psFCM and FCM clustering algorithms. To illustrate the efficiency of the proposed algorithm it was compared with the pshFCM, psFCM and FCM algorithms through several experiments on a number of benchmark data sets and from the experimental results it can be concluded that the DpsFCM algorithm shows superior clustering performance over the other clustering algorithms in terms of the determination of initial centroids, number of trials and iterations and total execution time.

CHAPTER 7

Enhancement on the PBM Validity Index

This short chapter presents a proposed enhancement that is applicable to two efficient validity indices namely the *PBM* validity index and the *PBMF* validity index discussed in section 4.6 of chapter 4. Experimental results are presented to show the better performance of the proposed forms over the two existing forms of *PBM* index.

7.1 The Proposed Enhancement

The *PBM* and *PBMF* validity indices are both the squares of product of three quantities. For *PBM* index, the quantities are $1/K$, E_1/E_K and D_K and for its fuzzy version, the *PBMF* index, the quantities are $1/K, E_1/J_m(U,Z)$ and D_K. If the second quantities for both the indices, i.e., E_1/E_K and $E_1/J_m(U,Z)$ are inspected carefully, then it can be seen that the quantities are the ratio of the sum of intra cluster distances for the complete data set considered as a single cluster and the sum of intra cluster distances for the complete data set broken into K clusters, for hard clustering (corresponding to the case of *PBM* index) or c fuzzy clusters, for fuzzy clustering (corresponding to the case of *PBMF* index). The quantities are a measure of the compactness of a K or c cluster system, and ideally their values should be maximized. The denominator of these quantities decrease with increase in the number of clusters but the numerator remains fixed. In fact, the numerator E_1, i.e., the sum of distances of all observations of the data set from the geometric center of the complete data set, is a constant quantity for a given data set and was originally added to the definition of the *PBM* and *PBMF* indices by Pakhira et al. just to disallow the quantities from

getting too small [68]. But in the light of the latest superlative advances in floating point computation and processing capabilities by modern computers this problem of "values getting too small for computer processing" does not remain as a problem at all. Thus the numerator E_1 can be safely ignored from the second quantities of both the *PBM* and *PBMF* validity indices and the original forms of the *PBM* and *PBMF* indices can be modified to reduce computations in their evaluation without affecting their characteristics. The modified forms named *kPBM*(*K*) and *kPBMF*(*K*), where the prefix *k* means 'kernel' or 'core', can be obtained from *PBM*(*K*) and *PBMF*(*K*) as follows:

$$kPBM(K) = \frac{PBM(K)}{E_1^2}$$

i.e.

$$kPBM(K) = \left[\frac{1}{K} \times \frac{1}{E_K} \times D_K\right]^2$$

and,

$$kPBMF(K) = \frac{PBMF(K)}{E_1^2}$$

i.e.

$$kPBMF(K) = \left[\frac{1}{K} \times \frac{1}{J_m(U,Z)} \times D_K\right]^2$$

As can be seen from the above equations, the proposed kernel forms do not require the calculation of the constant quantity E_1 and thus require lesser computation in their evaluation. For data sets with large values for *n* and *d* the computation of the quantity E_1 can incur significant cost.

In the next section experimental justifications are provided to support the proposed modifications.

7.2 Experimental Results and Discussion

To demonstrate the better performance of the *kPBM* and *kPBMF* validity indices over the *PBM* and *PBMF* indices, the validity indices were evaluated on all the well known benchmark numeric data sets presented in section 4.6 of chapter 4. Some of the results obtained from the conducted experiments are presented in Table

7.1 through 7.4. In Table 7.1, the values of the *PBM* and *kPBM* indices obtained using the K-means clustering algorithm for two of the data sets Iris and Cancer, for K = 2, 3,...,10, are compared and in Table 7.2 the values of the *PBMF* and *kPBMF* indices obtained using the FCM clustering algorithm for the Iris and Cancer data sets, for c = 2, 3,...,10, are compared. This comparison is shown to demonstrate the preservation of the validation characteristics of *PBM* and *PBMF* indices by their respective proposed kernel forms *kPBM* and *kPBMF* indices.

Table 7.1: Comparison of values obtained for *PBM* and *kPBM* validity indices by K-means clustering algorithm for K = 2, 3,..., 10 for Iris and Cancer data sets

No. of clusters K	Iris		Cancer	
	PBM	*kPBM*	*PBM*	*kPBM*
2	19.9233	0.000234291	**142.822**	**5.07728e-6**
3	**25.1754**	**0.000296054**	103.126	3.66607e-6
4	20.7486	0.000243996	75.9747	2.70087e-6
5	11.8664	0.000139545	61.5145	2.18682e-6
6	12.4381	0.000146268	45.927	1.63269e-6
7	10.7152	0.000126007	35.0487	1.24597e-6
8	9.21186	0.000108328	27.6969	9.84613e-7
9	10.3237	0.000121403	24.3976	8.67324e-7
10	8.7435	0.000102821	23.1193	8.21881e-7

Table 7.2: Comparison of values obtained for *PBMF* and *kPBMF* validity indices by FCM clustering algorithm for c = 2, 3,..., 10 for Iris and Cancer data sets

No. of clusters c	Iris		Cancer	
	PBMF	*kPBMF*	*PBMF*	*kPBMF*
2	21.8314	0.00025673	**167.963**	**5.97103e-6**
3	**28.2204**	**0.000331862**	143.575	5.10404e-6
4	24.7414	0.000290951	120.485	4.28321e-6
5	20.7735	0.000244289	104.341	3.70929e-6
6	18.5807	0.000218502	93.2341	3.31444e-6
7	17.7894	0.000209198	84.0972	2.98962e-6
8	15.941	0.000187461	74.8518	2.66095e-6
9	14.8778	0.000174958	66.6795	2.37043e-6
10	13.8462	0.000162826	64.6411	2.29797e-6

In Table 7.3 the computation times for the *PBM* and *kPBM* validity indices, obtained by applying the K-means algorithm on all the data sets, are compared and in Table 7.4 the computation times for the *kPBM* and *kPBMF* validity indices, obtained using the FCM algorithm, are compared.

Table 7.3: Comparison of average computation times for *PBM* and *kPBM* validity indices using the K-means clustering algorithm

Data set	No. of clusters	Computation times (in seconds) obtained for	
		PBM	***kPBM***
Iris	3	0.00056572	0.00041532
Cancer	2	0.00094967	0.00067492
Wine	3	0.00078079	0.00054047
SODAR1	3	0.00035655	0.00029178
SODAR2	3	0.00036099	0.00030826
Data_3_2	3	0.0003578	0.0002855
Data_5_2	5	0.00074198	0.00061554
Data_6_2	6	0.00085885	0.00073185
Data_9_2	9	0.0014941	0.0013462
Data_10_2	10	0.0013673	0.0012752
Data_4_3	4	0.00082038	0.00067516
Ruspini	4	0.00039211	0.00033829

Table 7.4: Comparison of average computation times for *PBMF* and *kPBMF* validity indices using the FCM clustering algorithm. For FCM, $m = 1.5$

Data set	No. of clusters	Computation times (in seconds) obtained for	
		PBMF	***kPBMF***
Iris	3	0.00075841	0.00068939
Cancer	2	0.001572	0.0013744
Wine	3	0.0009119	0.00079143
SODAR1	3	0.00076183	0.00070904
SODAR2	3	0.00073419	0.00067556
Data_3_2	3	0.00066384	0.00059792
Data_5_2	5	0.0011398	0.0010754
Data_6_2	6	0.0017348	0.0016658
Data_9_2	9	0.0061079	0.0059705
Data_10_2	10	0.0038106	0.0037298
Data_4_3	4	0.001586	0.0014809
Ruspini	4	0.00075481	0.00068795

All the computation times presented in Table 7.3 and in Table 7.4 are the average of 100 computation times corresponding to the 100 evaluations of each of the four validity indices. The MATLAB 7 script '*pbm_vs_kpbm_et_kmeans.m*', included in the package PatternWiz (described in chapter 8), was used to obtain the computation times presented in Table 7.3 and the MATLAB 7 script '*pbmf_vs_kpbmf_et_fcm.m*', included in the package PatternWiz, was used to obtain the computation times presented in Table 7.4.

From Tables 7.3 and 7.4 it can be clearly seen that the calculation of the modified kernel forms of the *PBM* and *PBMF* indices requires lesser computation time. Further from Tables 7.1 and 7.2, it can also be confirmed that the *kPBM* and *kPBMF* indices preserve the original validation characteristics of the *PBM* and *PBMF* validity indices. Thus the proposed modifications are optimal and prove better substitutes for the *PBM* and *PBMF* validity indices.

7.3 Chapter Summary

In this chapter, enhanced forms of each of the validity indices *PBM* and *PBMF*, namely the *kPBM* and *kPBMF* validity indices, obtained by applying a common enhancement principle applicable to both the *PBM* and *PBMF* validity indices, are presented. The performances of the proposed forms are compared with the *PBM* and *PBMF* validity indices through experiments conducted by applying clustering algorithms on several standard data sets. The results showed the optimality of the proposed forms and higher suitability of the use of the proposed forms over the standard existing forms.

CHAPTER 8

PatternWiz - A software package for pattern recognition

This chapter presents a description of the software package PatternWiz, which was developed as one of the primary objective of the research. The package was developed in MATLAB 7 by integrating together the optimized MATLAB implementations of all the clustering algorithms and soft computing approaches investigated in the research so as to obtain a valuable research aid for researchers working in the area of pattern recognition. MATLAB is a high-level language and interactive environment that enables one to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and FORTRAN [119]. The package PatternWiz principally offers functionalities for the study and comparison of the working of different clustering algorithms and soft computing approaches against standard benchmark data and other custom data.

8.1 Functions and Features

The software package PatternWiz can be used to perform two broad categories of functions related to pattern recognition, namely:

1) Hard and Fuzzy Logic based Clustering.
2) Optical Character Recognition of digits (Digit Recognition) from distorted images, using Artificial Neural Network.

Based on the three different data domains, numeric, image and text, PatternWiz makes available three different types of clustering functionalities:

- Clustering of numeric data.
- Clustering of image data.
- Clustering of collections of raw text documents.

The functions are now described below in relation to the package PatternWiz.

Hard and Fuzzy Logic based Clustering

The package PatternWiz is a graphical user interface (GUI) based package, thus it provides a convenient visual analysis environment to facilitate clustering and clustering validation. The package supports the use of the KM (K-means) and HBDKM (Hyper Block Division based K-means) clustering algorithms for hard clustering and for fuzzy clustering it supports the use of the FCM (Fuzzy c-means), DpsFCM (Deterministic partition simplification FCM), psFCM (partition simplification FCM) and the pshFCM (partition simplification hybrid FCM) clustering algorithms. The algorithms are described in detail in the previous chapters. Since the evaluation of clustering results is very important in cluster analysis, so all the 12 validity indices discussed thus far are also incorporated in PatternWiz. The 12 validity indices included in PatternWiz are listed below:

- For hard clustering algorithms (K-means and HBDKM)
  Pakhira Bandyopadhyay Maulik (PBM) Index
  kernel Pakhira Bandyopadhyay Maulik (kPBM) Index
  Xie-Beni (XB) Index
  Dunn's Index (DI)
  Separation and Compactness (SC) index
  Davies-Bouldin (DB) Index
- For fuzzy clustering algorithms (FCM, DpsFCM, psFCM and pshFCM)
  The Fuzzy PBM (PBMF) Index
  kernel Fuzzy Pakhira Bandyopadhyay Maulik (kPBMF) Index
  The Fuzzy Xie-Beni (XB) Index
  The Partition Coefficient (PC) Index
  The Monotonic Partition Coefficient (MPC) Index
  The Classification Entropy (CE) Index

PatternWiz provides two different modes for clustering of input data – 'Single Clustering' and 'Multiple Clustering'. In 'Single Clustering' mode, the user chooses a particular value for the number of clusters (and any other additional parameter values, like *m* for Fuzzy clustering) and also specifies the number of trials and continues with the clustering of the data set, whereas in 'Multiple Clustering' mode, the user can provide a range of values for number of clusters with a range difference of 8, like 2 to 10 or 7 to 15, and other parameter values, and continue with the clustering of the data set for multiple values for number of clusters. The restriction of range difference to 8 is to avoid the excess generation of output visualization windows for clustering results. For 'Multiple Clustering' mode, PatternWiz also offers a useful feature to plot the values of the validity indices for their comparison thus the 'Multiple Clustering' mode is particularly useful for the performance evaluation of the validity indices.

Though both the modes 'Single Clustering' and 'Multiple Clustering' are supported for all the three types of data, care must be taken in the selection of number of trails for 'Single Clustering' and in the selection of the range of number of clusters for large numeric data sets, large images and large text collections, because for large input data the modes may be quite time consuming.

Clustering of numeric data

For clustering of a numeric data set, the package PatternWiz requires the input numeric data set file to be in the following format:

feature1, feature2, feature3, ..., class label

Thus an input numeric data set file is a 'tab' or 'comma' delimited text file containing numeric tabular data so that rows denote multidimensional data points and the columns denote dimensions. All the values in the file should be numeric, without special characters and missing values. Most of the well-known benchmark numeric data sets are available in this format, like the data sets from UCI Machine Learning repository. The class label column may or may not be present.

For example, if a numeric data set has the following five 3-dimensional features:

Feature 1: (0.05, -0.87, 0.22)

Feature 2: (0.05, 1.11, 0.85)

Feature 3: (3.29, 4.27, 5.91)

Feature 4: (5.89, 7.57, 4.71)

Feature 5: (12.61, 10.32, 9.76)

And the corresponding true class labels for the features, Feature1 through Feature 5 are 1, 1, 2, 2 and 3 respectively, then the contents of the text data file containing these points can be:

0.05	-0.87	0.22	1
0.05	1.11	0.85	1
3.29	4.27	5.91	2
5.89	7.57	4.71	2
12.61	10.32	9.76	3

The true class labels can also be present in the very first column of the data file.

PatternWiz also offers another useful and interesting feature of generation of a custom 2-dimensional data set visually. The feature requires the MATLAB Image Processing Toolbox [120] to be installed.

Use of all the 6 clustering algorithms and 12 validity indices are supported by PatternWiz for clustering of numeric data.

Clustering of image data

PatternWiz supports the use of all the 6 clustering algorithms for image clustering. However, out of the 12 validity indices the Dunn's Index *DI* is not made available for images as its computation gets quite costly. The input image file can be in any valid image file format that is supported by the '*imread*' function of MATLAB [121].

Clustering of collections of raw text documents

PatternWiz makes available only the K-means clustering algorithm for clustering of text documents. Like the case of images, for text clustering also, the Dunn's Index *DI* is not made available because of its computational cost. Given a collection of text documents, PatternWiz first preprocesses the text contents of each of the text files in the collection by the removal of stopwords and other unnecessary symbols. It refers a file named, 'stopwords.txt' containing 700 stopwords, that can be modified for better results by the addition of more stopwords and unnecessary text elements that may be removed during preprocessing. The outcome of preprocessing is a very large multidimensional TF-IDF matrix whose rows represent documents and columns represent the TF-IDF weights of the terms in the text documents. This TF-IDF matrix can then be clustered using K-means for a specified number of clusters.

After the time taking generation of the TF-IDF matrix, the user may save the generated matrix into a 'tab' delimited text file for future use. Along with this the user *must* save the file names of the files processed. The pair of text files will then represent a preprocessed TF-IDF text data. When loading the preprocessed TF-IDF text data at a later stage, the user then needs to select the pair of text files as prompted by PatternWiz. However, the user may also cluster the TF-IDF matrix separately, if the user clusters the TF-IDF data file as a purely numeric data set. For this the user will have to choose 'Numeric data' in the 'Data domain selection window' (Figure 8.2) of PatternWiz and choose the TF-IDF text data file as the numeric input data file. As numeric data, the TD-IDF matrix can be clustered using all the 6 clustering algorithms, but the Euclidean distance measure will be used as the proximity measure rather than the cosine distance measure in the usual case.

PatternWiz supports the visualization of hyper-block partitioning by the HBDKM clustering algorithm, and the visualization of k-d tree partitioning by the DpsFCM, psFCM and pshFCM clustering algorithms for the 2-dimensional and 3-dimensional data sets. Also after each clustering, PatternWiz displays a lot of details like the execution times, objective function values, validity indices' values, number of iterations required etc., for the clustering performed. For labeled data sets, PatternWiz also displays the detailed misclassification error percentages, for the clusters formed.

Optical Character Recognition of digits (digit recognition) from distorted images

Apart from clustering, PatternWiz also provides another useful pattern recognition function in the form of optical character recognition of digits (called in short as digit recognition) from distorted images of printed and handwritten digits. The recognition is done using the hamming neural network described in detail in chapter 3. For the hamming neural network the Ramp and Hard Limit transfer functions are used. The functions are defined below:

$$\text{Ramp function:} \quad f(x) = \begin{cases} 1, x \geq 1 \\ x, 0 < x < 1 \\ 0, x \leq 0 \end{cases}$$

$$\text{Hard Limit function:} \quad f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

8.2 Usage

To start working with the package PatternWiz, the user needs to choose the PatternWiz application directory as the 'Current Directory' of MATLAB and then type 'PatternWiz' in the MATLAB command window ('PatternWiz.fig' file should not be double-clicked, as it is a MATLAB figure file). The user may also open the 'PatternWiz.m' file by double-clicking and press 'F5' key to run the file. The user is then presented with the following GUI window:

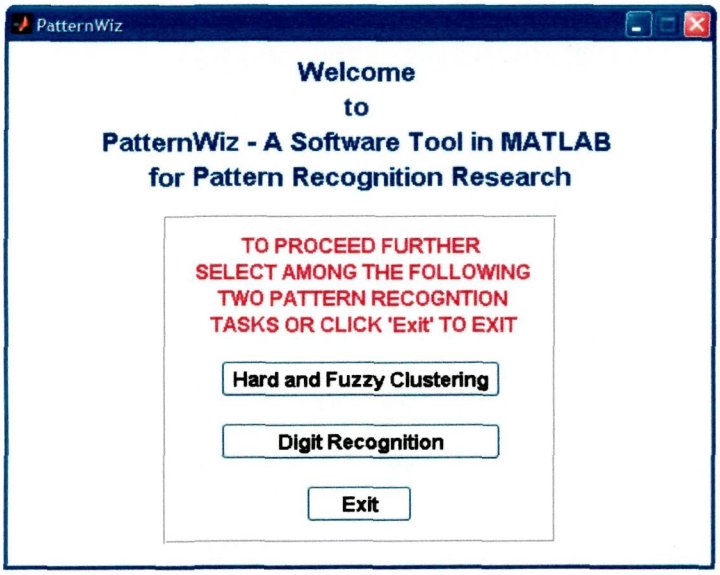


Figure 8.1: PatternWiz welcome window

From the main window, the user then needs to select 'Hard and Fuzzy Clustering' or 'Digit Recognition' as per choice. If the user selects 'Hard and Fuzzy Clustering' then the user is presented with the following window for selecting the data domain.

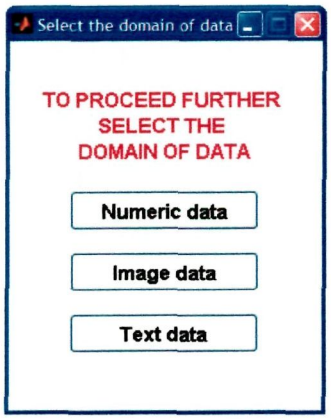


Figure 8.2: Data domain selection window of PatternWiz

From the 'Data domain selection window', the user then needs to select 'Numeric Data', 'Image Data' or 'Text Data' as per choice. Sample windows with an input data loaded for each of the three cases, are now shown next in respective manner:

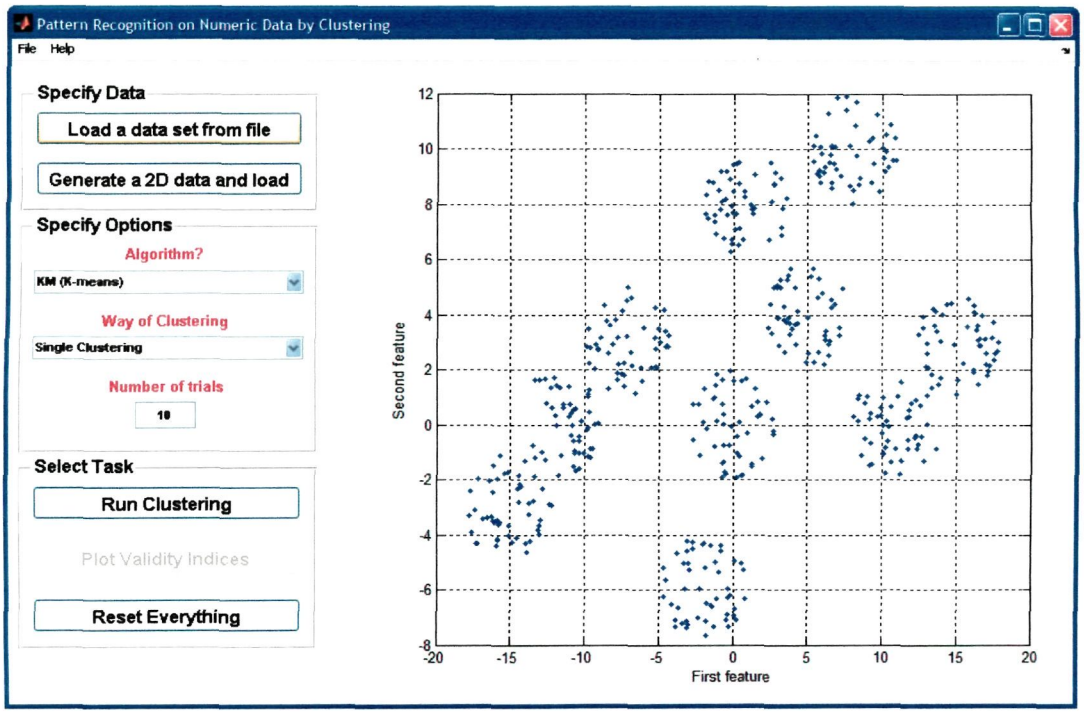


Figure 8.3: PatternWiz numeric data clustering window

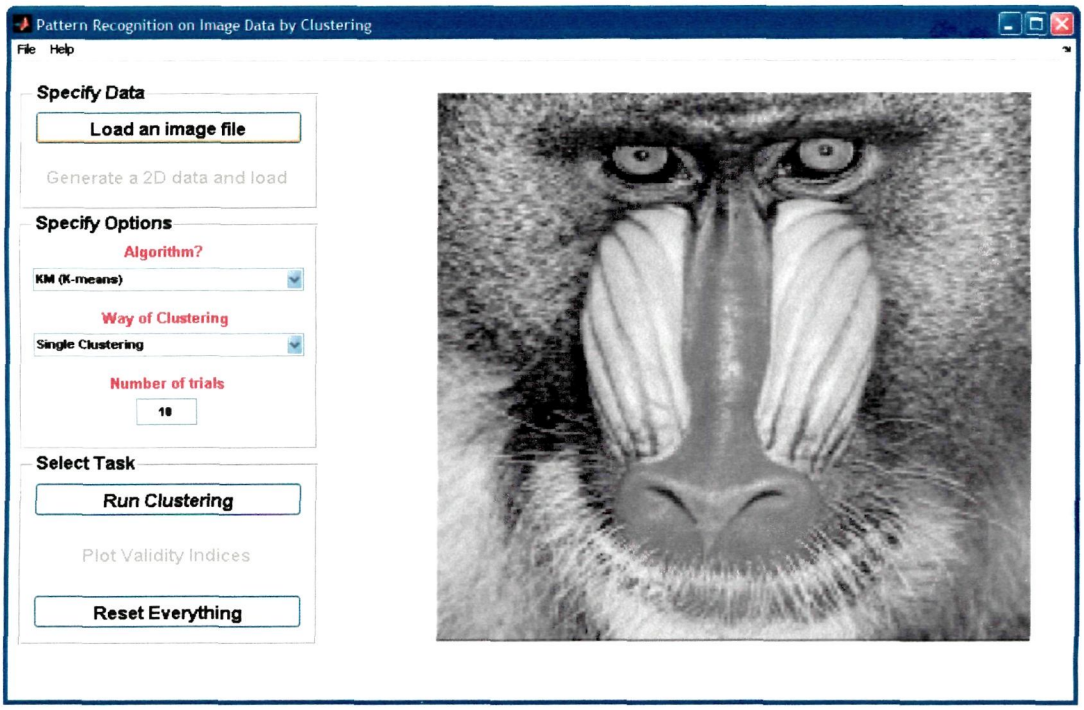


Figure 8.4: PatternWiz image data clustering window

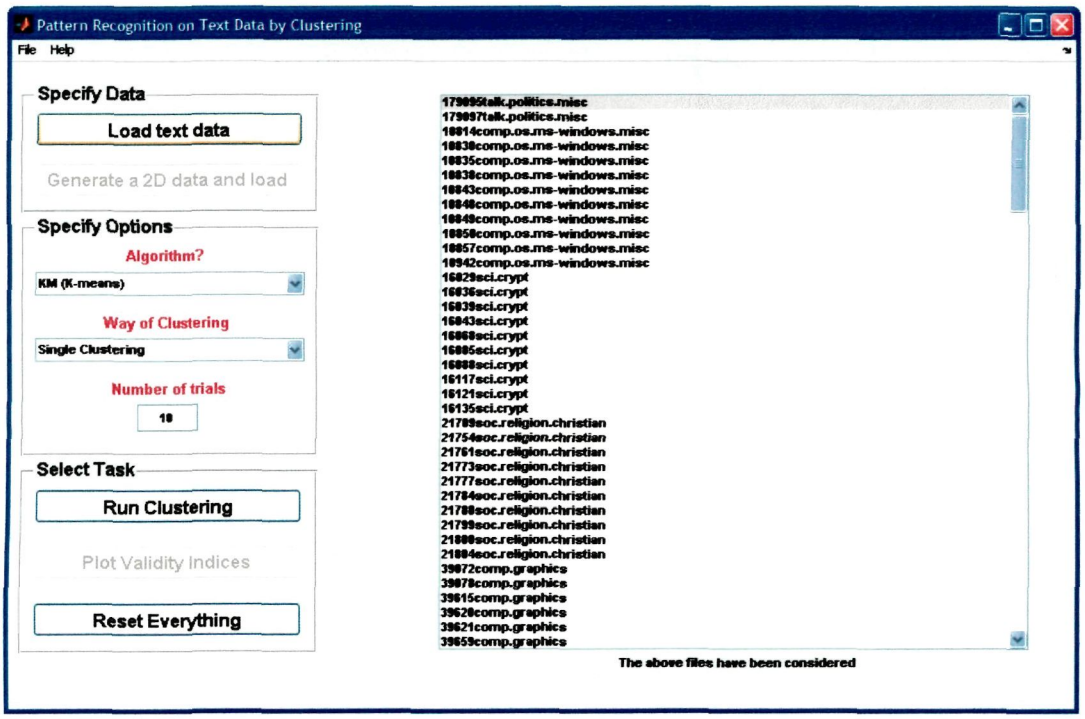


Figure 8.5: PatternWiz text data clustering window

From the above three figures it can be clearly seen that the selection of choices for clustering is quite easy. Usage of most of the GUI components is self explanatory. In all the three cases the user needs to perform three basic things - load the input data, select the clustering algorithm and select the mode of clustering. If the mode of clustering is 'Single clustering', the user needs to specify the number of trials and click the 'Run Clustering' command button and then provide the number of clusters and other input parameter values (like *m* for fuzzy clustering algorithms). In 'Multiple Clustering' mode the user needs to specify the 'Range of number of clusters' with a range difference of 8 as explained earlier, and then click the 'Run Clustering' command button. After these instructions to PatternWiz, the package will cluster the input data and present the results in graphical form along with the other necessary information, like execution time, objective function values etc as discussed earlier.

In all the three cases the 'Run Clustering' command button provides one click access to all the clustering and validation visualization features of PatternWiz.

For 'Multiple Clustering' mode, the values obtained for the validity indices after clustering can be plotted for comparison by clicking the command button 'Plot Validity Indices'. For 'Single Clustering' mode, the button remains disabled.

The 2-dimensional custom data generation feature of PatternWiz works as follows: The user first needs to click the 'Generate a 2D data and load' command button, available in the numeric data clustering window. This brings up a prompt asking for the presence of Image Clustering Toolbox of MATLAB which is needed for the feature to work. Upon selecting 'Yes', PatternWiz checks for the presence of the toolbox and on success, prompts the user to specify the X and Y axes' ranges. The user has to then specify the maximum and minimum value ranges along the two dimensions. After this a figure with a blank 2-dimensional Cartesian axes system is displayed, on which the user can draw the points visually. When finished with all but the last point the user needs to plot the last point with a mouse *double click* for the plotting to end. An illustration of the use of the feature is shown below, figuratively:

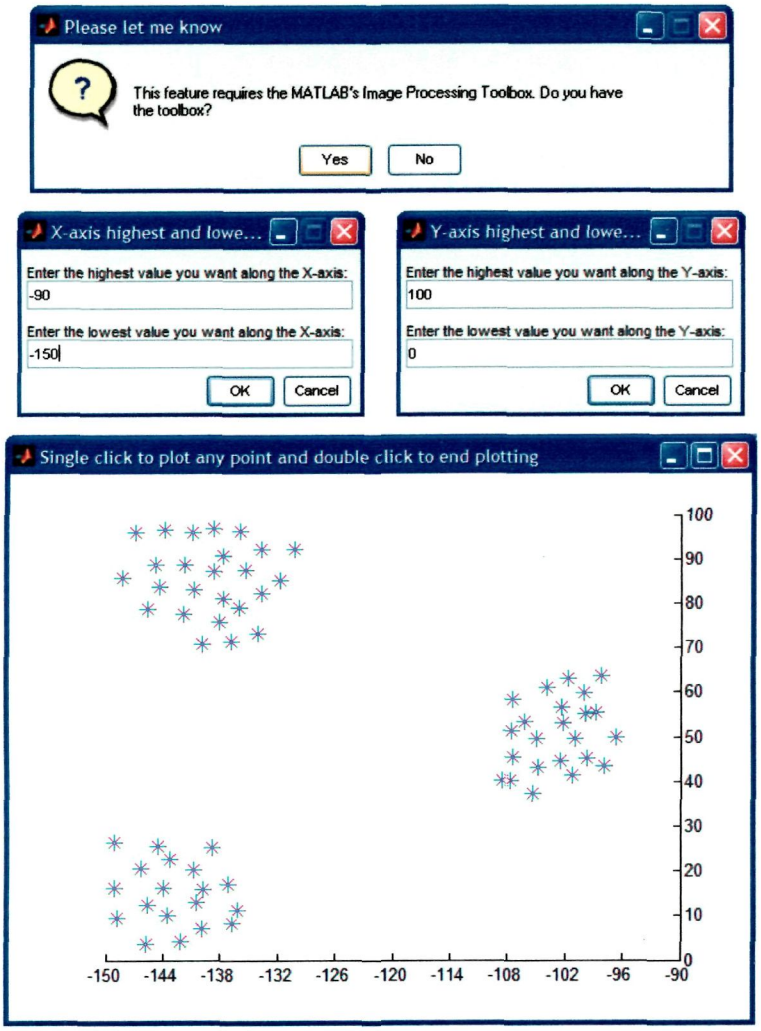


Figure 8.6: PatternWiz prompts and data plotting window for custom 2D numeric data set generation

If the user selects 'Digit Recognition' in the PatternWiz welcome window then the main window of digit recognition is displayed after a splash screen. A sample of that window, with a set of template images loaded, is shown below:

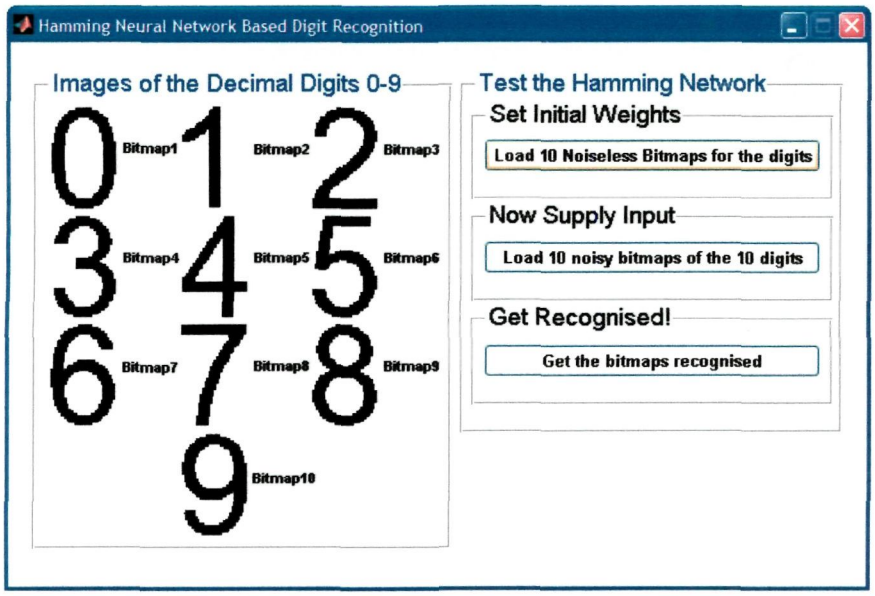


Figure 8.7: Hamming neural network based digit recognition window of PatternWiz

After loading a template image set, the user needs to load a set of images to be recognized. On pressing the command button 'Get the bitmaps recognised' the neural network recognizes the loaded images and outputs the recognition results in the MATLAB command window and a prompt of completion is shown to the user:

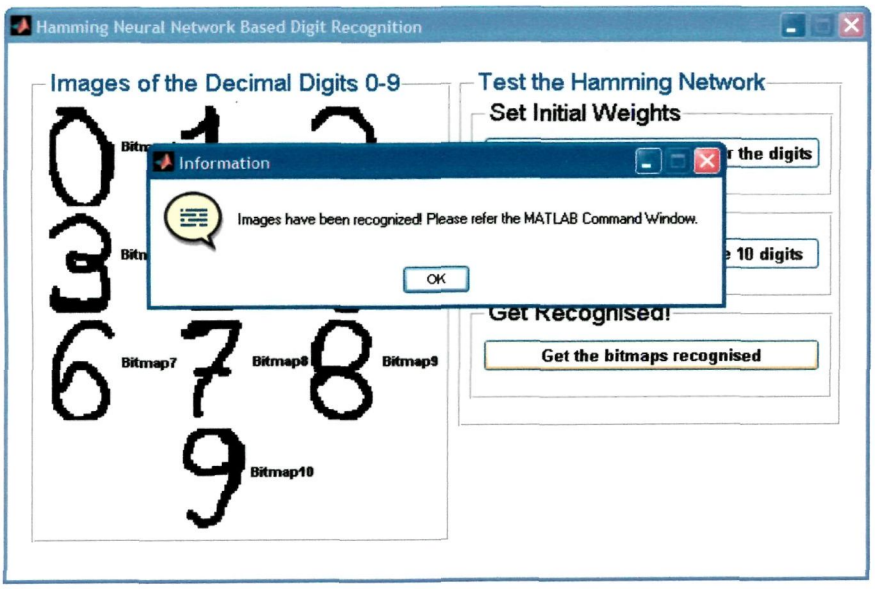


Figure 8.8: Sample digit recognition completion prompt of PatternWiz

8.3 Miscellaneous details

In this section some miscellaneous details about the package PatternWiz are provided. At first some important notes related to PatternWiz are presented below:

Note 1: The Statistics Toolbox of MATLAB needs to be installed for viewing the Silhouette plot for higher dimensional data from the numeric and text data domains. Without this toolbox the software will work but Silhouette plots cannot be displayed.

Note 2: The Image Processing Toolbox of MATLAB needs to be installed for the 'getpts()' function that is used in the custom 2D data set generation. Without this the software will work but the custom 2D data set generation feature will be unavailable.

Note 3: The package also uses the following three contributed code files (in modified or unmodified forms) by other authors:

- 'contourvis.m': The contour visualization code for visualization of fuzzy clustering results. The code was written by adopting some code from unlicensed 'clusteval.m' code file of Clustering Toolbox by Janos Abony available at: http://www.mathworks.com/matlabcentral/fileexchange/7486-clustering-toolbox
- 'valid_errorate.m' and 'ind2cluster.m': The pair of code files for computing percentage of error rates for every cluster if true class labels are given. For the package PatternWiz, the 'valid_errorate.m' code file was modified to include the file name of the input data set file also, for display in the MATLAB command window along with the error values. The code files are included in the package CVAP: Cluster Validity Analysis Platform (cluster analysis and validation tool) distributed under BSD License by Kaijun WANG. CVAP is available at: http://www.mathworks.com/matlabcentral/fileexchange/14620

The authors of the above code files are acknowledged and thanked for their contribution.

The package PatternWiz is implemented with the help of several functions and two scripts implemented in MATLAB. Those functions are now presented below with arguments' information and other details. For all the functions presented below, the following arguments variables have the same meanings. Function specific argument variables are described with the respective functions.

Argument variables with same meaning for all the functions:

X: the $n \times d$ data set matrix

K: the number of clusters (for hard clustering algorithms)

c: the number of clusters (for fuzzy clustering algorithms)

n: the number of data points in the data set X

d: the number of dimensions of the data set X

cen: the initial set of optimal centroids, a $K \times d$ matrix

shwv: the 'show visualization' argument for deciding on the display of partitioning in visual manner for the HBDKM, DpsFCM, psFCM and pshFCM algorithms

labels: the final cluster labels for points in X, a column vector of n rows

m: the fuzzifier value

FUNCTIONS

1) *result* = *kmeansalgo*(X, K, n, *cen*)

This function implements the K-means clustering algorithm. For random initialization *cen* must be chosen as an empty matrix. The output argument *result* is a structure with the following members:

result.labels = the final cluster labels on convergence, a column vector of n rows

result.iter = the number of iterations required for convergence

result.z = the $K \times d$ matrix of final cluster centroids obtained on convergence

result.obj = the final value of the objective function after convergence

2) *result* = *hbdkmeans*(X, K, n, d)

This function implements the HBDKM clustering algorithm presented in section 5.4. The output argument *result* is a structure with the following members:

result.labels = the final cluster labels on convergence, a column vector of n rows

result.iter = the number of iterations required for convergence

result.z = the $K \times d$ matrix of final cluster centroids obtained on convergence

result.obj = the final value of the objective function after convergence

result.cen_opt = the $K \times d$ matrix of optimal initial centroids

result.mpbm = the maximum *PBM* index value obtained for *cen_opt*

result.optbpr = the best block partition ratio (*bpr*) value corresponding to the best hyper-block partitioning

result.optivs = the *Intrv_of_Seg* array (defined in Table 5.1) corresponding to the best hyper-block partitioning

3) [*cen_opt mpbm optbpr optivs*] = *hbp*(*X*, *K*, *n*, *d*)

This function implements the hyper-block partitioning procedure described in Table 5.1 and also implements the procedure of Table 5.2. The function is called by the *hdbkmeans* function for obtaining the hyper-block partitioning of the data set *X*. As outputs, the function *hbp* returns the optimal initial centroids *cen_opt*, the maximum *PBM* index value obtained for *cen_opt*, *maxpbm*, the best block partition ratio value *optbpr* and the *Intrv_of_Seg* array corresponding to the best hyper-block partitioning.

4) *result* = *kmeanscosd*(*X*, *K*, *n*, *cen*)

This function implements the K-means clustering algorithm using the cosine distance measure. The input data set *X* is normalized first and in each iteration the set of centroids obtained is also normalized. The members of the output argument *result* are analogous to those of *result* of *kmeansalgo*.

5) *pbmval* = *evalpbm*(*X*, *K*, *n*, *cen*)

This function calculates the value of the *PBM* validity index. The function can be used for the evaluation of 'goodness' of a centroid set *cen* as optimal seeds or for validation of clustering results obtained using any hard clustering algorithm.

6) *kpbmval* = *evalkpbm*(*X*, *K*, *n*, *cen*)

This function calculates the value of the *kPBM* validity index. In place of the *evalpbm* function, this function also can be used for the evaluation of 'goodness' of a centroid set *cen* as optimal seeds or for validation of clustering results obtained using any hard clustering algorithm. The function *evalkpbm* is faster than

the function *evalpbm*.

7) *pbmval=evalpbmcosd(X, nrmX, K, n, cen)*

This function calculates the value of the *PBM* validity index using the cosine distance measure. The function is used in PatternWiz for validation of clustering results obtained by the K-means algorithm for text documents' collections. The input argument *nrmX* is the normalized form of the data matrix *X*. For normalization, the function *normalizearr* is used, which is described later.

8) *vindex = hvalidity(X, K, n, cen, labels, type_of_data)*

This function computes the values of the 6 hard clustering related validity indices discussed in chapter 4 in section 4.6.1. The *type_of_data* argument is set to empty string '' for the numeric data domain. For images it is set to 'image' and for text clustering it is set to 'text' so that by checking its value the computation of Dunn's Index can be avoided. The output argument *vindex* is a structure with 6 members, vindex.PBM, vindex.kPBM, vindex.XB, vindex.SC, vindex.DB, vindex.DI. The member names are self explanatory. For validation of results of text clustering the cosine distance measure is used and for the results of numeric data clustering and image clustering, the Euclidean distance measure is used.

9) *result = fcmeans(X, c, n, d, cen, m)*

This function implements the FCM clustering algorithm. For random initialization *cen* must be chosen as an empty matrix. The output argument *result* is a structure with the following members:

result.U = the final fuzzy membership function matrix, a $n \times c$ matrix

result.d = the $n \times c$ matrix of distances of each point from every centroid

result.z = the $c \times d$ matrix of final cluster centroids obtained

result.iter = the number of iterations required for convergence

result.obj = the final value of the objective function after convergence

10) *result = DpsFCM(X, c, n, d, m, shwv)*

This function implements the DpsFCM clustering algorithm presented in section 6.4. Thus the function also implements the procedure described in Table 6.3. For *k*-d partitioning the function calls the *kdtreepart* function described later.

shwv is set to 1 for displaying the *k*-d partitioning and 0 otherwise. The function calls the *fcmeans* function for clustering, thus the members of the output argument *result* are analogous to those of output argument *result* of *fcmeans*.

11) *result* = *psFCM*(*X*, *c*, *n*, *d*, *m*, *shwv*)

This function implements the psFCM clustering algorithm. For *k*-d partitioning the function calls the *kdtreepart* function described later. *shwv* is set to 1 for displaying the *k*-d partitioning and 0 otherwise. The function calls the *fcmeans* function for clustering, thus the members of the output argument *result* are analogous to those of output argument *result* of *fcmeans*.

12) *result* = *pshFCM*(*X*, *c*, *n*, *d*, *m*, *shwv*)

This function implements the pshFCM clustering algorithm. For *k*-d partitioning the function calls the *kdtreepart* function described later. *shwv* is set to 1 for displaying the *k*-d partitioning and 0 otherwise. The function calls the *fcmeans* function for clustering, thus the members of the output argument *result* are analogous to those of output argument *result* of *fcmeans*.

13) *T* = *kdtreepart*(*X*, *depth*, *maxd*, *disc_dim_nos*, *shwv*)

This function implements the *k*-d partitioning algorithm given in Table 6.2. Initially *depth* is set to 0 and *maxd* is set to $\lceil \log_2(c) \rceil + 1$. The value of *shwv* is supplied by *DpsFCM* or by *psFCM* or by *pshFCM* whichever calls *kdtreepart*. The output argument *T* is an $M \times (k+1)$ matrix such that $M \leq 2^{maxd}$. The number of dimensions in *X* is denoted by *k*.

14) *pbmfval* = *evalpbmf*(*X*, *c*, *n*, *cen*, *m*)

This function calculates the value of the *PBMF* validity index. The function can be used for the evaluation of 'goodness' of a centroid set *cen* as optimal seeds or for validation of clustering results obtained using any fuzzy clustering algorithm.

15) *kpbmfval* = *evalkpbmf*(*X*, *c*, *n*, *cen*, *m*)

This function calculates the value of the *kPBMF* validity index. In place of the *evalpbmf* function, this function also can be used for the evaluation of 'goodness' of a centroid set *cen* as optimal seeds or for validation of clustering results

obtained using any fuzzy clustering algorithm. The *evalkpbmf* function is faster than *evalpbmf* function.

16) *vindex* = *fvalidity*(*X*, *c*, *n*, *result*, *m*)

This function computes the values of the 6 fuzzy clustering related validity indices discussed in chapter 4 in section 4.6.2. The input argument *result* is usually set to the output (*result*) of the four fuzzy clustering algorithms FCM, DpsFCM, psFCM and pshFCM. The output argument *vindex* is a structure with 6 members, vindex.PBMF, vindex.kPBMF, vindex.XB, vindex.PC, vindex.MPC, vindex.CE. The member names are self explanatory.

17) *contourvis*(*X*, *c*, *cen*)

This function implements the contour map visualization routine for visualization of fuzzy clustering of 2-dimensional data as per values in the fuzzy membership matrix *U*. Contour map visualization obtained for the data set Data_9_2 using DpsFCM clustering algorithm for $m = 1.5$ is shown below:

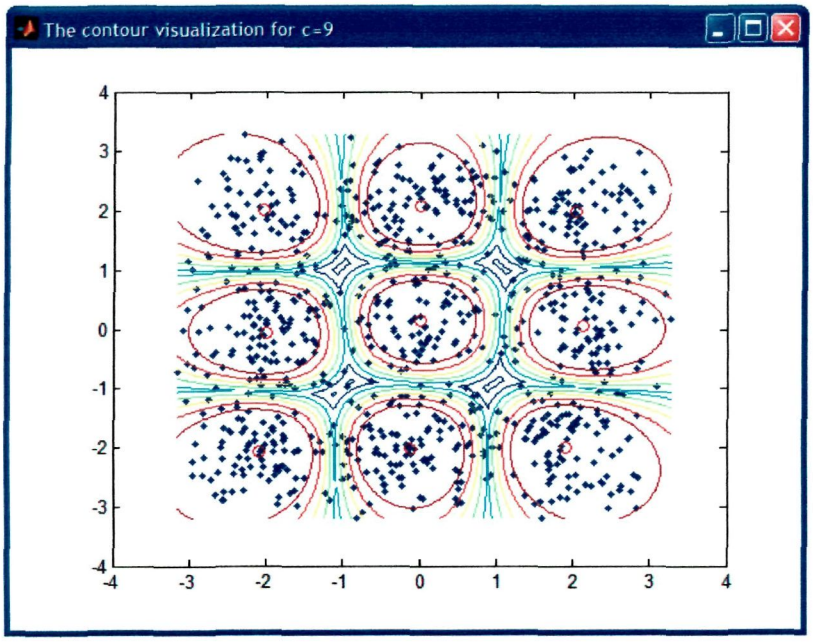


Figure 8.9: Contour visualization of fuzzy clustering for Data_9_2 data set

18) [*dist_cen* *dist_cen_mat*] = *cen_dist_mat*(*cen*)

This function computes the inter centroid distances of the *ncen* number of centroids in the matrix *cen* and returns the $\frac{ncen(ncen-1)}{2}$ distances among the

ncen centroids in *dist_cen*, a row vector, and also returns the distances in the *ncen* × *ncen* matrix *dist_cen_mat*.

19) *dist_cen* = *inter_cen_dist*(*cen*)

This function computes the inter centroid distances of the *ncen* number of centroids in the matrix *cen* and returns the $\frac{ncen(ncen-1)}{2}$ distances among the *ncen* centroids in *dist_cen*, a row vector.

20) *Rerror* = *valid_errorate*(*labels*, *truelabels*, *nof*)

This function computes the misclassification error percentages for every cluster if true class labels are given. It also returns the total percentage of misclassification error for a given clustering, in *Rerror*. The error rates of individual clusters are displayed in the MATLAB command window. The argument *truelabels* is a column vector of *n* rows which contains the true class labels retrieved from the input data set file. The argument *nof* denotes the name of the input data set file from which the *truelabels* are extracted. The value of *nof* can also be set to an empty string ''.

21) [*clusters*, *newlabels*] = *ind2cluster*(*labels*)

This function takes as its input the final class labels obtained from clustering i.e., *labels* and returns the *K* or *c* different sets of indices of the points in different clusters in the cell array *clusters* (an array of *K* or *c* elements). Along with this it also returns the corresponding *K* or *c* sets of cluster labels.

22) *out_arr* = *ramptransfunc*(*inp_arr*)

This function computes the ramp transfer function defined in section 8.1 for use in processing done by the hamming neural network for optical character recognition of decimal digits from distorted images. The input argument *inp_arr* is a 1 × 10 row vector as the number of images considered is 10 at a time.

23) *out_arr* = *hardlimtransfunc*(*inp_arr*)

This function computes the hard limit transfer function defined in section 8.1 for use in processing done by the hamming neural network for optical character recognition of decimal digits from distorted images. The input argument *inp_arr*

is a 1 × 10 row vector as the number of images considered is 10 at a time.

24) *normarr* = *normalizearr*(*arr*)

This function normalizes the input matrix *arr* so that the values in the matrix lie in the 0 to 1 range and returns the normalized form of the matrix *arr* in *normarr*.

25) *varargout* = *PatternWiz*(*varargin*)

This function can take variable number of input and output arguments and is usually invoked by simply typing *PatternWiz* in the MATLAB command window which brings up the PatternWiz welcome window as shown in Figure 8.1.

26) *varargout* = *PatternWizMain*(*varargin*)

This function can also take variable number of input and output arguments and is usually invoked by simply typing *PatternWizMain* in the MATLAB command window which brings up the PatternWiz numeric data clustering window as shown in Figure 8.3 (without any numeric data set loaded). It should be invoked if the user wants to do numeric data clustering only.

27) *varargout* = *DataOptions* (*varargin*)

This function can also take variable number of input and output arguments and is usually invoked by simply typing *DataOptions* in the MATLAB command window which brings up the Data domain selection window as shown in Figure 8.2. It should be invoked if the user wants to do data clustering only.

28) *varargout* = *hnnbdr*(*varargin*)

This function can also take variable number of input and output arguments and is usually invoked by simply typing *hnnbdr* in the MATLAB command window which brings up the hamming neural network based digit recognition window as shown in Figure 8.7 (without the images loaded). It should be invoked if the user wants to do hamming neural network based digit recognition only.

The four functions *PatternWiz*, *PatternWizMain*, *DataOptions* and *hnnbdr* also have associated MATLAB figure files with the same names. The user should not double click and open the 'fig' files to invoke these functions. Also, the user should not make any changes to these 'fig' files as this may also lead to erratic functioning.

SCRIPTS

1) *pbm_vs_kpbm_et_kmeans.m*

This MATLAB script compares the average computation times obtained for the two validity indices *PBM* and *kPBM* by evaluating their values for 100 times. For clustering the K-means algorithm is used.

2) *pbmf_vs_kpbmf_et_fcm.m*

This MATLAB script compares the average computation times obtained for the two validity indices *PBMF* and *kPBMF* by evaluating their values for 100 times. For clustering the FCM algorithm is used.

8.4 Chapter Summary

In this chapter, the functions, features, usage and other miscellaneous details of the software package PatternWiz are documented. For many of the important functionalities and features, the associated screenshots from the package are also presented. The development of the package PatternWiz was one of the primary objectives of the present research. It was developed using MATLAB 7 by integrating together the optimized MATLAB implementations of all the clustering algorithms and soft computing approaches investigated in the research so as to obtain a valuable research aid for researchers working in the area of pattern recognition. The package PatternWiz principally offers functionalities for the study and comparison of the working of different clustering algorithms and soft computing approaches against standard benchmark data and other custom data.

CHAPTER 9

Conclusions

This chapter concludes the thesis by summarizing the works, findings and contributions of the thesis. It also presents some directions of future research.

9.1 Summary of Works

This thesis investigated the effectiveness of various clustering algorithms and soft computing approaches from a pattern recognition perspective. The research work for the thesis began with the obtainment of a general overview on pattern recognition and on its paradigms and various approaches. A study work was then done on some prominent soft computing approaches like fuzzy logic and artificial neural networks and on their applications in pattern recognition. The K-means and FCM clustering algorithms were then analyzed thoroughly to understand their shortcomings. Since clustering is unsupervised, so a study and investigation was also done on various existing validity assessment indices for validation of clustering results. Based on the study on clustering algorithms, two new clustering techniques HBDKM and DpsFCM were developed and based on the study on validity indices the enhanced validity indices *kPBM* and *kPBMF* were obtained. A software package was then developed by integrating together the implementations of all the clustering algorithms and soft computing approaches investigated in the research so as to obtain a valuable research aid for researchers working in the area of pattern recognition. All these works are presented in the various chapters as follows:

Chapter 2 presents an overview of pattern recognition and its two different

paradigms namely supervised and unsupervised. It also presents some standard approaches of pattern recognition like, template matching, geometrical classification, statistical approach, syntactic approach and neural networks. Some applications of pattern recognition are also presented. Further a brief discussion on pattern recognition on data from some different domains is also presented in chapter 2.

In chapter 3, a discussion on soft computing approaches fuzzy logic and artificial neural networks is presented. A specific example of pattern recognition on images using a hamming neural network is also presented. The findings obtained from the experiment showed that the classification accuracy of a hamming network is fully dependant on the fact that how well the templates stored by it match the input pattern vectors. From the experiment it can also be reinforced and concluded that template matching is not the most effective approach to pattern recognition and for more sophisticated pattern recognition, neural networks with dynamic learning ability should be used.

In chapter 4 the details of the analytical work on clustering algorithms is presented. The chapter begins with a brief overview on clustering, similarity measures and different taxonomical representations of clustering. It then presents a discussion on two prominent clustering algorithms K-means and FCM and on some popular hard and fuzzy clustering validity assessment indices. Further, the chapter also presents several experimental results of experiments on the applications of the clustering algorithms for pattern recognition on numeric, image and text data.

In chapter 5 the HBDKM clustering algorithm is presented as an efficient deterministic version of the widely used K-means clustering algorithm. The chapter also presents experimental results obtained by applying the HBDKM algorithm on a number of benchmark data sets, to illustrate its efficiency over the K-means clustering algorithm.

In chapter 6 the DpsFCM clustering algorithm is presented as an efficient deterministic version of the psFCM clustering algorithm. The chapter also presents experimental results, obtained by applying the DpsFCM algorithm on a number of benchmark data sets, to illustrate its efficiency over the pshFCM, psFCM and FCM clustering algorithms.

In chapter 7 the enhanced validity indices *kPBM* and *kPBMF* are presented. The indices were obtained by applying a common enhancement applicable on both the *PBM* and *PBMF* validity indices. Experimental results are presented in the chapter to show the better performance of the proposed forms over the existing forms of the *PBM* validity index.

In chapter 8 the features and working of PatternWiz are presented. The software package was developed in MATLAB 7 as one of the primary objective of the research. The package includes a number of useful and interesting features to facilitate clustering of different types of data and the validation of clustering. It also integrates a module for the assessment of performance and efficiency of hamming neural network based pattern recognition on images. The module was developed for the identification of digits in distorted images of printed and handwritten digits.

9.2 Summary of Contributions

The main contributions of this thesis include the development of two new clustering algorithms HBDKM and DpsFCM, the formulation of *Hybrid split* rule for *k*-d tree partitioning, the formulation of enhanced clustering validity indices *kPBM* and *kPBMF* and the development of the software package *PatternWiz*. Other important contributions of the thesis include the studies and analysis of soft computing approaches, the studies and analysis of hard and fuzzy clustering algorithms and the studies and analysis of various hard and fuzzy clustering validity assessment indices.

The HBDKM clustering algorithm finds better initial centroids compared to the K-means clustering algorithm and obtains the same set of optimal initial centroids every time. As a result the algorithm is deterministic and obtains the same final clustering every time, for a given data set. The DpsFCM clustering algorithm is also deterministic and obtains the same set of better initial centroids every time unlike the other non-deterministic or stochastic algorithms, pshFCM, psFCM and FCM. Thus the resultant clustering is also obtained as same every time. The *hybrid split* rule described in subsection 6.3.1 is a hybridization of the *original split* rule, the *standard*

split rule and the *midpoint split* rule and was designed to obtain a computationally faster splitting rule for *k*-d tree partitioning, but in line with the standard rules. The *kPBM* and *kPBMF* validity indices evaluate faster than the corresponding *PBM* and *PBMF* validity indices by avoiding the computation of a constant term present in both the *PBM* and *PBMF* indices. The validation characteristics of the proposed versions however remain the same as the existing forms. The software package *PatternWiz* can be used as a useful research aid by researchers working in the area of pattern recognition.

9.3 Future Directions

In this thesis, the study and analysis related to the soft computing approaches was done by investigating fuzzy logic and artificial neural networks. As a future work, other soft computing approaches like genetic algorithms (GA), probabilistic reasoning (PR) and rough sets (RS) can be investigated for pattern recognition. Study and investigation of their hybridizations can also be done. Moreover apart from the fuzzy logic based clustering, other soft computing based clustering approaches may also be investigated.

The proposed HBDKM clustering algorithm uses a deterministic approach. As a future extension a non-deterministic version of the algorithm can be obtained by restricting the hyper-block partitioning process to only once and then it can be compared for performance with algorithms like K-means. Further the performance of the hyper-block partitioning technique itself may be investigated, by using it for other clustering algorithms which require the computations of good initial centroids for optimal convergence.

For the DpsFCM clustering algorithm also, a non-deterministic version can be obtained by allowing the *k*-d tree partitioning to change on different trials using different (may be random) values for maximum depth *maxd* and the performance may then be compared with other algorithms. Like hyper-block partitioning, the performance of the *k*-d tree partitioning technique also may be investigated by using it for other clustering algorithms.

The validity indices evaluate the goodness of clustering obtained for a given data set. Many validity indices however perform better for data sets with greater degree of regularity in the cluster shapes and orientation in the hyper-volume of the data set than for the data sets with noise and overlapping clusters. An extensive study thus needs to be undertaken on several validity indices for a better insight of their properties and validation performances. Since the performance study of validity indices in also dependant on the underlying algorithm used, hence as a future work, an investigation on the performances of the validity indices may also be done by applying the indices for validation of results obtained using various clustering algorithms for the same collection of data sets.

Bibliography

[1] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern classification and Scene Analysis*. second ed., New York: John Wiley & Sons, Inc., 2000.

[2] S. Watanabe, *Pattern Recognition: Human and Mechanical*. New York: John Wiley & Sons, Inc., 1985.

[3] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264-323, Sep. 1999.

[4] M.H.C. Law, M.A.T. Figueiredo, and A.K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1154-1166, Sep. 2004.

[5] M.G.H. Omran, A.P. Engelbrecht, and A. Salman, "An overview of clustering methods," *Intelligent Data Analysis*, vol. 11, no. 6, pp. 583-605. Dec. 2007.

[6] L.A. Zadeh, "Fuzzy logic, neural networks and soft computing," *Communications of the ACM*, vol. 37, no. 3, pp. 77-84, Mar. 1994.

[7] S.K. Pal, "Soft Computing Pattern Recognition: Principles, Integrations, and Data Mining," *New Frontiers in Artificial intelligence*, LNAI 2253, T. Terano, T. Nishida, A. Namatame, S. Tsumoto, Y. Ohsawa, and T. Washio, eds., Springer, 2001, pp. 261-271.

[8] J.P. Jesan, "The neural approach to pattern recognition," *Ubiquity*, vol. 2004, pp. 2-2, Apr. 2004; http://www.acm.org/ubiquity/views/v5i7_jesan.html.

[9] S. Mitra and T. Acharya, *Data Mining: Multimedia, Soft Computing, and Bioinformatics*. New York: John Wiley & Sons, Inc., 2003.

[10] A.K. Jain, R.P.W. Duin, and J. Mao, "Statistical Pattern Recognition: A Review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37, Jan. 2000.

[11] A.K. Pujari, *Data Mining Techniques*. first ed., Hyderabad: Universities Press (India) Pvt. Ltd., 2001.

[12] F.T. Allen, J.M. Kinser, and H.J. Caulfield, "A neural bridge from syntactic to statistical pattern recognition," *Neural Networks*, vol. 12, no. 2, pp. 519-526, Apr. 1999.

[13] S. Theodoridis and K. Koutroumbas, *Pattern recognition*. third ed., New York: Academic Press, Inc., 2006.

[14] G.J. Klir and B. Yuan, *Fuzzy Sets theory and Fuzzy Logic: Theory and Applications.* New Delhi: Prentice-Hall of India Pvt. Ltd., 2007.

[15] *Supervised and Unsupervised Pattern Recognition: Feature Extraction and Computational Intelligence.* E. Micheli-Tzanakou, ed., CRC Press, 2000.

[16] A.K. Jain and R.P.W. Duin, "Pattern Recognition," *The Oxford Companion to the Mind,* second ed., R.L. Gregory, ed., Oxford: Oxford University Press, 2004, pp. 698-703.

[17] L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition.* Springer, 1996.

[18] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* New York: John Wiley & Sons, Inc., 1973.

[19] K.S. Fu, *Syntactic Pattern Recognition and Applications.* Englewood Cliffs, N.J.: Prentice-Hall, 1982.

[20] T. Pavlidis, *Structural Pattern Recognition.* New York: Springer-Verlag, 1977.

[21] K.S. Fu., "A Step Towards Unification of Syntactic and Statistical Pattern Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 5, no. 2, pp. 200-205, Mar. 1983.

[22] L.I. Perlovsky, "Conundrum of combinatorial complexity," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 20, no. 6, pp. 666-670, Jun. 1998.

[23] R.J. Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches.* New York: John Wiley & Sons, Inc., 1992.

[24] K. Kpalma and J. Ronsin, "An Overview of Advances of Pattern Recognition Systems," *Vision Systems: Segmentation and Pattern Recognition,* G. Obinata and A. Dutta, eds., Vienna: I-Tech Education and Publishing, 2007, pp. 169-194.

[25] J. Liu, J. Sun, and S. Wang, "Pattern Recognition: An overview," *International Journal of Computer Science and Network Security,* vol. 6, no. 6, pp. 57-61, Jun. 2006.

[26] Y. Chang, B. Yang, and W. Yeh, "A generalized prime-number-based matrix strategy for efficient iconic indexing of symbolic pictures," *Pattern Recognition Letters*, vol. 22, no. 6-7, pp. 657-666, May 2001.

[27] A. Chakrabarty and B.S. Purkayastha, "An Analysis of Some Prime Generating Sieves," *The IUP Journal of Computer Sciences*, vol. 4, no. 1, pp. 16-26, Jan. 2010.

[28] U. Khurana and A. Koul, "Using Patterns to Generate Prime Numbers," *Proc. 3rd International Conference on Advances in Pattern Recognition* (ICAPR 2005), LNCS 3686, Springer, 2005, pp. 325-334.

[29] G. Greaves, *Sieves in number theory*. Berlin: Springer-Verlag, 2001.

[30] P. Pritchard, "A sublinear additive sieve for finding prime numbers," *Communications of the ACM*, vol. 24, no. 1, pp. 18-23, 1981.

[31] R.C. Gonzalez, and R.E. Woods, *Digital Image Processing.* second ed., Upper Saddle River, N.J.: Prentice-Hall, Inc., 2002.

[32] B. Jahne, *Spatio-Temporal Image Processing: Theory and Scientific Applications*. New York: Springer-Verlag, 1993.

[33] T.F. Gharib, M.M. Fouad, and M.M. Aref, "Fuzzy Document Clustering Approach using WordNet Lexical Categories," *Proc. International Joint Conference on Computer, Information, and Systems Sciences, and Engineering* (CIS2E 2008), Springer, 2008, pp. 181-186.

[34] D.R. Recupero, "A new unsupervised method for document clustering by using WordNet lexical and conceptual relations," *Information Retrieval.* vol. 10, no. 6, pp. 563-579, Dec. 2007.

[35] G. Salton, A. Wong, and C.S. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613-620, Nov. 1975.

[36] B.B., Wang, R.I. McKay, H.A. Abbass, and M. Barlow, "Learning Text Classifier using the Domain Concept Hierarchy," *Proc. International Conference on Communications, Circuits and Systems and West Sino Expositions*, IEEE Press, 2002, pp. 1230-1234.

[37] A. Hotho, S. Staab, and G. Stumme, "Wordnet improves Text Document Clustering," *ACM SIGIR Workshop on "Semantic Web"* (SWIR 2003), New York: ACM Press, 2003, pp. 541-544.

[38] J. Sedding and D. Kazakov, "WordNet-based text document clustering," *Proc. 3rd Workshop on Robust Methods in Analysis of Natural Language Data* (ROMAND 2004), Morristown, N.J.: Association for Computational Linguistics, 2004, pp. 104-113.

[39] S. Shehata, "A WordNet-Based Semantic Model for Enhancing Text Clustering," *Proc. 2009 IEEE International Conference on Data Mining Workshops* (ICDMW 2009), IEEE Computer Society, 2009, pp. 477-482

[40] G.A. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39-41, Nov. 1995.

[41] A. Chakrabarty, B.S. Purkayastha, and A. Roy, "Experiences in building the Nepali WordNet - insights and challenges," *Proc. 5th Global Wordnet Conference* (GWC 2010), New Delhi: Narosa Publishing House, 2010, pp. 192-197.

[42] I.P. Cabrera, P. Cordero, and M. Ojeda-Aciego, "Fuzzy Logic, Soft Computing, and Applications," *Bio-inspired Systems: Computational and Ambient intelligence: Part I*, LNCS 5517, J. Cabestany, F. Sandoval, A. Prieto, and J.M. Corchado, eds., Springer, 2009, pp. 236-244.

[43] L.A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338-353, Jun. 1965.

[44] D. Dubois and H. Prade, *Fuzzy Sets and Systems*. New York: Academic Press, Inc., 1988.

[45] G. B. Folland, *Real Analysis: Modern Techniques and Their Applications*. second ed., New York: John Wiley & Sons, Inc., 1999.

[46] L.A. Zadeh, "Preface", *Fuzzy logic technology and applications*. R. J. Marks II, ed., IEEE Technical Activities Board, 1994.

[47] S. Mitra and S.K. Pal, "Fuzzy sets in pattern recognition and machine intelligence," *Fuzzy Sets and Systems*, vol. 156, no. 3, pp. 381-386, Dec. 2005.

[48] R. Bellman, R. Kalaba, and L.A. Zadeh, "Abstraction and pattern classification," *Journal of Mathematical Analysis and Applications*, vol. 13, pp. 1-7, 1966.

[49] B. Yegnanarayana, *Artificial neural networks*. New Delhi: Prentice-Hall of India Pvt. Ltd., 2005.

[50] W.S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[51] K.N. Gurney, *An Introduction to Neural Networks*. London: Routledge, 1997.

[52] M.C. Purucker, "Neural network quarterbacking," *IEEE Potentials*, vol. 15, no. 3, pp. 9-15, 1996.

[53] A. Schmid, Y. Leblebici, and D. Mlynek, "Mixed analogue-digital artificial-neural-network architecture with on-chip learning," *IEEE Proceedings - Circuits, Devices and Systems*, vol. 146, no. 6, Dec. 1999, pp. 345-349.

[54] I. Meilijson, E. Ruppin, and M. Sipper, "Fast Computation in Hamming and Hopfield Networks," *Algorithms and Architectures*, C.T. Leondes, ed., New York: Academic Press, Inc., 1998. pp. 123-126.

[55] S. Liverani, "Bayesian Clustering of Curves and the Search of the Partition Space," Ph. D. thesis, University of Warwick, 2009.

[56] U. Maulik and S. Bandyopadhyay, "Performance Evaluation of Some Clustering Algorithms and Validity Indices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1650-1654, Dec. 2002.

[57] J. Fan and W. Xie, "Some notes on similarity measure and proximity measure," *Fuzzy Sets and Systems*, vol. 101, no. 3, pp. 403-412, Feb. 1999.

[58] P.E. Green and V.R. Rao, "A Note on Proximity Measures and Cluster Analysis," *Journal of Marketing Research*, vol. 6, no. 3, pp. 359-364. Aug. 1969.

[59] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, N.J.: Prentice-Hall, Inc., 1988.

[60] A. Vakali and G. Pallis, *Web Data Management Practices - Emerging Techniques and Technologies*. Idea Group Publishing, 2007.

[61] M.R. Anderberg, *Cluster Analysis for Applications*. New York: Academic Press, Inc., 1973.

[62] J. McQueen, "Some methods for classification and analysis of multivariate observations," *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, 1967, pp. 281-297.

[63] M. Khalilian, N. Mustapha, MD.N. Suliman, and MD.A. Mamat, "A Novel K-Means Based Clustering Algorithm for High Dimensional Data Sets," *Proc. International MultiConference of Engineers and Computer Scientists* (IMECS 2010), vol. 1, 2010, pp. 503-507.

[64] J.C. Dunn, "Some recent investigations of a new fuzzy partition algorithm and its application to pattern classification problems," *Journal of Cybernetics*, vol. 4, no. 2, pp. 1-15, 1974.

[65] J.C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press, 1981.

[66] E.H. Ruspini, "A new approach to clustering," *Information and Control*, vol. 15, no. 1, pp. 22-32, 1969.

[67] M.-S. Yang, K.-L. Wu, J.-N. Hsieh, and Jian Yu, "Alpha-Cut Implemented Fuzzy Clustering Algorithms and Switching Regressions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 3, pp. 588-603, Jun. 2008.

[68] M.K. Pakhira, S. Bandyopadhyay, and U. Maulik, "Validity index for crisp and fuzzy clusters," *Pattern Recognition*, vol. 37, no. 3, pp. 487-501, Mar. 2004.

[69] F. Klawonn, V. Chekhtman, and E. Janz, "Visual Inspection of Fuzzy Clustering Results," *Advances in soft computing: engineering design and manufacturing*, J.M. Benitez, O. Cordon, F. Hoffmann, and R. Roy, eds., Springer, 2003. pp. 65-76.

[70] K.S. Cheng and S.M. Pan, "A New Clustering Method for the Distribution Analysis of Hearing Neurons," *Proc. World Congress on Biomedical Engineering and Medical Physics*, Springer, 2006, pp. 2296-2299.

[71] M.J.A. Berry and G. Linoff, *Data Mining Techniques for Marketing, Sales, and Customer Support*. New York: John Wiley & Sons, Inc., 1997.

[72] M. Halkidi, Y. Batistakis, and M. Varzirgiannis, "On Clustering Validation Techniques," *Journal of Intelligent Information Systems*, vol. 17, no. 2-3, pp. 107-145, Dec. 2001.

[73] X.L. Xie and G. Beni, "A Validity Measure for Fuzzy Clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 841-847, Aug. 1991.

[74] J.C. Dunn., "Well separated clusters and optimal fuzzy-partitions," *Journal of Cybernetics*, vol. 4, pp. 95-104, 1974.

[75] A.M. Bensaid, L.O. Hall, J.C. Bezdek, L.P. Clarke, M.L. Silbiger, J.A. Arrington, and R.F. Murtagh, "Validity-guided (re)clustering with applications to image segmentation," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, pp. 112-123, May 1996.

[76] D.L. Davies and D.W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. pami-1, no. 2, pp. 224-227, Apr. 1979.

[77] N.R. Pal and J.C. Bezdek, "On cluster validity for the fuzzy c-means model," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, pp. 370-379, Aug. 1995.

[78] J.C. Bezdek, "Numerical Taxonomy with Fuzzy Sets", *Journal of Mathematical Biology*, vol. 1, no. 1, pp. 57-71, May 1974.

[79] R.N. Dave, "New measures for evaluating fuzzy partitions induced through c-shells clustering," *Proc. SPIE conference on Intelligent Robots and Computer Vision X: Algorithms and Techniques*, SPIE, 1992, vol. 1607, pp. 406-414.

[80] J.C. Bezdek, "Cluster validity with fuzzy sets," *Cybernetics and Systems*, vol. 3, no. 3, pp. 58-73, 1973.

[81] A. Frank and A. Asuncion, (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[82] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179-188, 1936.

[83] S. Choudhury, "Tropospheric VHF Propagation Studies Using Ground-Based In-situ and Acoustic Remote Sensing Technique," Ph. D. thesis, University of Calcutta, 2000.

[84] S. Bandyopadhyay and U. Maulik, "Genetic clustering for automatic evolution of clusters and application to image classification," *Pattern Recognition*, vol. 35, no. 6, pp. 1197-1208, Jun. 2002.

[85] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern Recognition*, vol. 32, no. 9, pp. 1455-1465, Sep. 2000.

[86] S. Bandyopadhyay and S.K. Pal, *Classification and Learning using Genetic Algorithms: Applications in Bioinformatics and Web Intelligence*. Heidelberg, Germany: Springer-Verlag, 2007.

[87] S. Bandyopadhyay and U. Maulik, "Nonparametric Genetic Clustering: Comparison of Validity Indices," *IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews*, vol. 31, no. 1, pp. 120-125, Feb. 2001.

[88] S. Bandyopadhyay, C.A. Murthy, and U. Maulik, "Pattern Classification Using Genetic Algorithms," *Pattern Recognition Letters*, vol. 16, pp. 801-808, Aug. 1995.

[89] E.H. Ruspini, "Numerical methods for fuzzy clustering," *Information Sciences: an International Journal*, vol. 2, no. 3, pp. 319-350, Jul. 1970.

[90] X.-J. Kong and X.-J. Tong, "Adaptive Algorithms for Weight of the Feature Weighted of FCM," *Proc. 5th International Conference on Image and Graphics* (ICIG 2009), IEEE Computer Society, 2009, pp. 501-505.

[91] A.R. Rao and V.V. Srinivas, *Regionalization of Watersheds: An Approach Based on Cluster Analysis*. Netherlands: Springer, 2008.

[92] A.R. Rao and V.V. Srinivas, "Some problems in regionalization of watersheds," *Water Resources Systems - Water Availability and Global Change*, S. Franks, G. Bloschl, M. Kumagai, K. Musiake, and D. Rosbjerg, eds., Wallingford, Oxfordshire, UK: IAHS Press, 2003, pp. 301-308.

[93] L.G. Shapiro and G.C. Stockman, *Computer Vision*. Upper Saddle River, N.J.: Prentice-Hall, Inc., 2001.

[94] D. Malyszko and S.T. Wierzchon, "Standard and Genetic k-means Clustering Techniques in Image Segmentation," *Proc. 6th International Conference on Computer Information Systems and Industrial Management Applications* (CISIM 2007), IEEE Publications, 2007, pp. 299-304.

[95] M.G.H. Omran, "Particle Swarm Optimization Methods for Pattern Recognition and Image Processing," Ph.D. thesis, University of Pretoria, 2005.

[96] R.H. Turi, "Clustering-based Color Image Segmentation," PhD Thesis, Monash University, Australia, 2001.

[97] Tf-idf weighting. 2010. [Online]. Available: http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html

[98] G. Salton and C. Buckley, *Term Weighting Approaches in Automatic Text Retrieval*, Technical Report. TR87-881, Cornell University, 1987.

[99] UCI Machine Learning Repository: Twenty Newsgroups Data Set. 2010. [Online].Available: http://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups

[100] P.S. Bradley and U.M. Fayyad, "Refining Initial Points for K-Means Clustering," *Proc. 15th International Conference on Machine Learning* (ICML 1998), San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 91-99.

[101] T. Su and J. Dy, "A Deterministic Method for Initializing K-means Clustering," *Proc. 16th IEEE International Conference on Tools with Artificial Intelligence* (ICTAI 2004), IEEE Computer Society, 2004, pp. 784-786.

[102] S.S. Khan and A. Ahmad, "Cluster center initialization algorithm for K-means clustering," *Pattern Recognition Letters*, vol. 25, no. 11, pp. 1293-1302, Aug. 2004.

[103] K. Arai and A.R. Barakbah, "Hierarchical K-means: an algorithm for centroids initialization for K-means," *Reports of the Faculty of Science and Engineering*, Saga University, vol. 36, no. 1, pp. 25-31, 2007.

[104] Y. Xinhua, Y. Kuan, and D. Wu, "A k-means Clustering Algorithm based on Self-Adoptively Selecting Density Radius," *International Journal of Computer Science and Network Security*, vol. 6, no. 8, pp. 43-47, Aug. 2006.

[105] I. Davidson and A. Satyanarayana, "Speeding up k-means Clustering by Bootstrap Averaging," *Proc. IEEE ICDM 2003 Workshop on Clustering Large Data Sets*, IEEE Publications, 2003, pp. 16-25.

[106] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," *Proc. 18th annual ACM-SIAM symposium on Discrete algorithms* (SODA 2007), Society for Industrial and Applied Mathematics, 2007, pp. 1027-1035.

[107] M. Belal and A. Daoud, "A New Algorithm for Cluster Initialization," *Proc. of World Academy of Science, Engineering and Technology*, vol. 4, World Academy of Science, Engineering and Technology, 2005, pp. 74-76.

[108] A. Likas, N. Vlassis, and J.J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451-461, Feb. 2003.

[109] M.-C. Hung, J. Wu, J.-H. Chang, and D.-L. Yang, "An Efficient k-Means Clustering Algorithm Using Simple Partitioning," *Journal of Information Science and Engineering*, vol. 21, no. 6, pp. 1157-1177, Nov. 2005.

[110] W.-K. Liao, Y. Liu, and A. Choudhary, "A Grid-based Clustering Algorithm using Adaptive Mesh Refinement," *Proc. 7th Workshop on Mining Scientific and Engineering Datasets* (MSD04), 2004, pp.61-69.

[111] M.-C. Hung and D.-L. Yang, "An Efficient Fuzzy C-Means Clustering Algorithm," *Proc. IEEE International Conference on Data Mining* (ICDM 2001), IEEE Computer Society, 2001, pp. 225-232.

[112] S.A. Begum, B.S. Purkayastha, A. Chakrabarty, and T. Som, "An efficient psFCM Clustering Algorithm," *Proc. IEEE International Advance Computing Conference* (IACC 2009), IEEE Publications, 2009, pp. 383-389.

[113] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509-517, Sep. 1975.

[114] T.W. Cheng, D.B. Goldgof, and L.O. Hall, "Fast Fuzzy Clustering," *Fuzzy Sets and Systems*, vol. 93, no. 1, pp. 49-56, Jan. 1998.

[115] S. Maneewongvatana and D.M. Mount, "It's okay to be skinny, if your friends are fat," *Proc. 4th Annual CGC Workshop on Computational Geometry*, 1999.

[116] J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Transactions on Mathematical Software.* vol. 3, no. 3, pp. 209-226, Sep. 1977.

[117] J.L. Bentley, "Multidimensional Binary Search Trees in Database Applications," *IEEE Transactions on Software Engineering*, vol. se-5, no. 4, pp. 333-340, Jul. 1979.

[118] kd-trie. [Online]. Available: http://www.worldlingo.com/ma/enwiki/en/kd-trie

[119] MATLAB - The Language Of Technical Computing. 2010. [Online]. Available: http://www.mathworks.com/products/matlab/

[120] Image Processing Toolbox 7.0. 2010. [Online]. Available: http://www.mathworks.com/products/image/

[121] imread. 2010. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/techdoc/ref/imread.html

Appendix A

Definitions of Terms

This appendix lists the terms frequently used in this thesis.

Artificial neural network: A closely interconnected network of simple processing units called artificial neurons arranged in the form of layers of processing units: input layer, hidden layer and output layer. The units in input layer are connected to the units in hidden layer, which are again connected to units in the output layer.

Cluster: A set of similar patterns.

Clustering: A process of partitioning a data set of n patterns into K subsets, called clusters.

Connection: A means of passing input from one neuron to another in an artificial neural network.

Connection weight: A numerical label associated with a connection which is a measure to reflect the strength of the connection between the neurons and the amount of information flow between them.

Convergence: Culmination of a process with a final result.

Crisp set: A conventional set wherein the degree of membership of any element in the set can be either 0 or 1.

Data set: A data set is a collection of data, usually presented in tabular form. Each column represents a particular feature. Each row corresponds to a pattern in the data set.

Digit recognition: Optical character recognition of printed and handwritten digits in distorted images.

Distance measure: A distance that is calculated to determine the similarity between two patterns in a data set.

Exemplar pattern vector: A vector of example patterns used in the training of a neural network.

Fuzzy logic: In broad sense, one of the major techniques of soft computing that offers tolerance to imprecision and suboptimality for solving problems involving real-life ambiguous situations. In narrow sense, a generalization of the traditional multivalued logical systems that aims at a formalization of approximate reasoning.

Global optimum: The optimum solution among all possible solutions of a combinatorial optimization problem.

Hard computing: The traditional computing approach where the primary considerations are precision, certainty and rigor. This form of computing strives for exactness and full truth.

Hyper-block: A hyper-block is a hyper-rectangular block containing a subset of points in the multidimensional feature space. A *hyper-rectangle* is the generalization of a *rectangle* for higher dimensions.

Hyper-plane: The generalization of the concept of *plane* for higher dimensions.

Image segmentation: The classification of a digital image into a set of segments by assigning a label to every pixel such that pixels with the same label share some common visual characteristics.

***k*-d tree**: A popular space partitioning data structure for organizing points in a *k*-dimensional space, in hyper-rectangular cells.

Learning: The process of obtaining an appropriate set of connection weights for an artificial neural network so that the network solves the task given to it in some optimal sense.

Local minimum: A point where the value of a function is lower than the value at any other point within some neighborhood. Local minimum need not be (but may be) a global minimum.

Objective function: A function as a criterion of goodness, that is to be maximized or minimized to obtain a solution for a combinatorial optimization problem.

Pattern: A single object or data point that is usually denoted by a feature vector $(x_1, x_2, x_3, \ldots, x_d)$ in a d-dimensional feature space.

Pattern recognition: A scientific discipline which uses machine intelligence for the classification of a given set of patterns into meaningful classes.

Seeds: The initial set of centroids of clusters for the K-means and FCM algorithms and other similar clustering algorithms.

Soft computing: A collection of methodologies that aims to simulate human like decision making by exploiting the tolerance for impression, uncertainty, approximate reasoning and partial truth to achieve tractability, robustness and low-cost solutions.

Validity index: A measure that is evaluated to assess the fitness of partitioning obtained by clustering a data set. Usually the optimal value of a validity index indicates the best clustering.

WordNet: A WordNet is a lexical reference system wherein a collection of nouns, verbs, adjectives and adverbs of a language are grouped into sets of cognitive synonyms (synsets), each expressing a distinct lexical concept or sense. These synsets are interlinked by means of conceptual-semantic and lexical relations.

Appendix B

Definitions of Symbols

This appendix lists the symbols frequently used in this thesis.

$\mathbf{x}$	A feature vector
X	A data set
n	Number of points in the data set
d	The number of dimensions or features of a pattern or feature vector
$\Re$	Set of all real numbers
K or c	Number of clusters or classes
$\{x, y, \ldots.\}$	Set of elements $x, y, \ldots.$
$\mu_A(\mathbf{x})$	Membership degree of $\mathbf{x}$ in fuzzy set A
$[a, b]$	Closed interval of real numbers between a and b
$(a, b]$	Left open, right closed interval of real numbers between a and b
w	Weight vector of an artificial neuron
ξ	Exemplar pattern vector
T	Threshold level
U	Partition matrix
$[x_{ij}]$	A matrix
μ_{ij}	Membership grade of a pattern in a cluster
$\mathbf{x}_j$	The j^{th} pattern in data set X
C_k	k^{th} cluster
$D(\mathbf{x}, \mathbf{y})$	Distance between points $\mathbf{x}$ and $\mathbf{y}$
$\mid a \mid$	Modulus or absolute value of a
$\|P\|$	The magnitude of a vector P
$\| \cdot \|$	Distance measure
z_k	A cluster centroid
$\mathcal{P}$	A fuzzy pseudopartition
m	The fuzzifier parameter

$J_m(\mathcal{P})$	Fuzzy objective or criterion function
I	An index set (a set whose members index or label members of another set)
$\propto$	Infinity
V_i	The i^{th} value of a validity index
U_m	The m^{th} partition of a data set X
bpr	Block partition ratio
maxd	The maximum depth value for k-d tree partitioning for the DpsFCM clustering algorithm

Appendix C

List of Publications

A. Journals

1) A. Chakrabarty and B.S. Purkayastha, "An Analysis of Some Prime Generating Sieves," *The IUP Journal of Computer Sciences*, vol. 4, no. 1, pp. 16-26, Jan. 2010. (Published),
Available at SSRN: http://ssrn.com/abstract=1539354

2) A. Chakrabarty and B. S. Purkayastha, "A kernel PBM index for fuzzy clustering of numeric data", AU International Journal of Science & Technology, India, vol. 5, no. 3, pp. 94-98, 2010. ISSN: 0975-2773 (Published)

B. Conferences

1) S.A. Begum, B.S. Purkayastha, A. Chakrabarty, and T. Som, "An efficient psFCM Clustering Algorithm," *Proc. IEEE International Advance Computing Conference* (IACC 2009), IEEE Publications, 2009, pp. 383-389. (Published),
Available at: http://dx.doi.org/10.1109/IADCC.2009.4809220

2) A. Chakrabarty, B.S. Purkayastha, and A. Roy, "Experiences in building the Nepali WordNet - insights and challenges," *Proc. 5th Global Wordnet Conference* (GWC 2010), New Delhi: Narosa Publishing House, 2010, pp. 192-197. (Published),
Available at:
www.cfilt.iitb.ac.in/gwc2010/pdfs/35_napali_wordnet__Chakrabarty.pdf

3) A. Chakrabarty and B.S. Purkayastha, "On improvement of the PBM validity index for numeric data clustering," *Proc. National Conference on Current Trends in Computer Science* (CTCS 2010), Silchar, 22-24 Feb 2010. (Accepted)

Appendix D

PatternWiz Software Package CD