University of Alberta

Development of Partially Supervised Kernel-based Proximity Clustering Frameworks and Their Applications

by

Daniel Graves

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Digital Signal and Image Processing

Department of Electrical and Computer Engineering

Spring 2011
Edmonton, Alberta

Abstract. The focus of this study is the development and evaluation of a new partially supervised learning framework. This framework belongs to an emerging field in machine learning that augments unsupervised learning processes with some elements of supervision. It is based on proximity fuzzy clustering, where an active learning process is designed to query for the domain knowledge required in the supervision. Furthermore, the framework is extended to the parametric optimization of the kernel function in the proximity fuzzy clustering algorithm, where the goal is to achieve interesting non-spherical cluster structures through a non-linear mapping. It is demonstrated that the performance of kernel-based clustering is sensitive to the selection of these kernel parameters. Proximity hints procured from domain knowledge are exploited in the partially supervised framework.

The theoretic developments with proximity fuzzy clustering are evaluated in several interesting and practical applications. One such problem is the clustering of a set of graphs based on their structural and semantic similarity. The segmentation of music is a second problem for proximity fuzzy clustering, where the aim is to determine the points in time, i.e. boundaries, of significant structural changes in the music. Finally, a time series prediction problem using a fuzzy rule-based system is established and evaluated. The antecedents of the rules are constructed by clustering the time series using proximity information in order to localize the behavior of the rule consequents in the architecture. Evaluation of these efforts on both synthetic and real-world data demonstrate that proximity fuzzy clustering is well suited for a variety of problems.

Table of Contents

1. Introduction & Motivation 1

2. Background 6

2.1. Unsupervised learning 6
2.2. Supervised learning 23
2.3. Evolutionary Optimization 27

3. Related Work 28

3.1. Forms of Domain Knowledge 28
3.2. Partially supervised learning 30
3.3. Active learning 35
3.4. Proximity Clustering Approaches 38
3.5. Time Series Clustering 38
3.6. Time Series Segmentation 42

4. Kernel Functions and Proximity Functions 47

4.1. Introduction 47
4.2. Kernel Functions: Theory 47
4.3. Types of Kernel Functions 48
4.4. Proximity Functions 51
4.5. Motivation: Kernel Clustering Case Study 53

5. Parametric Optimization of Kernels 57

5.1. Optimization Framework 57
5.2. Domain Knowledge: Proximity 57
5.3. Methodology: evolutionary optimization 58
5.4. Differential evolution 59

6. Proximity-based Fuzzy Clustering 60

6.1. Clustering Framework 60
6.2. Evaluation Criterion 62
6.3. Experimental Results 63

7. Active Learning 69

7.1. Introduction 69
7.2. FCM-based Approach 70
7.3. Experimental Evaluation 72
7.4. Discussion 79

8. Multi-proximity Fuzzy Clustering 80

8.1. Motivation 80
8.2. Multi-proximity clustering framework 82
8.3. Learning the weights 84
8.4. Experiments 86
8.5. Discussion 88

9. Graph Clustering 89

9.1. Introduction 89
9.2. Background – Edit Distance of Graphs 92
9.3. Relational Clustering of Graphs 93
9.4. Graph clustering applied to software requirements engineering 97
9.5. Software Requirement Analysis – Case Study 109
9.6. Conclusions 118

10. Time Series Prediction with Proximity-based Fuzzy Clustering 119

10.1. Model Architecture 120
10.2. Network Design 122
10.3. Experiments 126

11. Structural Musical Segmentation 133

11.1. Problem Formulation and Motivation 133
11.2. The Feature Space 134
11.3. Segmentation: a proximity approach 135
11.4. Experiments 136
11.5. Conclusions 139

12. Conclusions 140

References 142

Appendices 155

Appendix A: Incorporating Semantic Proximity of Nodes in the Proximity Function 155
Appendix B: Variability models 156
Appendix C: Musical segmentation labels 158

List of Figures

Figure 1. Unsupervised learning and supervised learning 1
Figure 2. Partially supervised personalized music recommendation system 2
Figure 3. Partially supervised learning 2
Figure 4. Cluster distance measures 7
Figure 5. Dendrogram example 8
Figure 6. Kernel Example 14
Figure 7. KFCM-F feature space and kernel space 15
Figure 8. KFCM-K feature space and kernel space 17
Figure 9. Example graph 19
Figure 10. Ensemble clustering framework 20
Figure 11. Feed-forward neural network architecture 24
Figure 12. Radial basis function (RBF) neural network architecture 25
Figure 13. Taxonomy of domain knowledge 28
Figure 14. Example of must-link and cannot link constraints 29
Figure 15. Committee-based active learning 37
Figure 16: Subsequences of a time series 40
Figure 17: Clustering of many time series 41
Figure 18. Synthetic data sets 52
Figure 19. Boundaries produced by FCM, KFCM-F and KFCM-K on the Fuzzy "X" data set 54
Figure 20. Classification rate (a) versus σ^2 for Gaussian KFCM-K, and classification rate (b) versus polynomial kernel parameters for polynomial KFCM-K on the ring data set (c=2) 55
Figure 21. Classification rate (a) versus σ^2 for Gaussian KFCM-K (c=2) and classification rate (b) versus polynomial kernel parameters KFCM-K (c=2) on the ionosphere data set 56
Figure 22. Kernel Function Optimization Framework 57
Figure 23. Proximity fuzzy clustering architecture 60
Figure 24. Example cluster boundaries 63
Figure 25. Prototypes of synthetic data sets for cosine, Gaussian and polynomial kernel functions 66
Figure 26. Convergence of proximity fuzzy clustering on the zig-zag data set with the Gaussian kernel ($m=2.5, \sigma^2=0.5$) 66
Figure 27. Classification error and reconstruction error on the zig-zag data set for (a) cosine, (b) Gaussian, and (c) polynomial kernel functions 67
Figure 28. Active learning 69
Figure 29. Proposed active learning framework 70
Figure 30. Labeled patterns in synthetic data sets 74
Figure 31. Views of data 80
Figure 32. Demonstration data set of multi-proximity fuzzy clustering 81
Figure 33. Multi-proximity fuzzy clustering on demonstration data set (m=1.5, c=3) 81
Figure 34. Ensemble clustering on demonstration data set (m=1.5, c=3) 82
Figure 35. Multi-proximity fuzzy clustering framework 82
Figure 36. Multi-proximity fuzzy clustering architecture 83
Figure 37. Learning the weights for each view/source 85
Figure 38. A collection of trees 91
Figure 39. Example trees 93
Figure 40. Synthetic example: number of clusters 96
Figure 41. Partition for synthetic example (c=3) 96
Figure 42. Objective function value at each iteration 97
Figure 43. Variability Models of Automotive Software Product Lines 98
Figure 44. Process of identifying best practices in software requirements engineering 100
Figure 45. Variability Models for Sibling Similarity 101
Figure 46. A Collection of Variability Models 106

Figure 47. Synthetic example: number of clusters 108
Figure 48. Partition for synthetic example (c=3) 108
Figure 49. Variability Model Fragments of the Crisis management System 110
Figure 50. Partitioning of Data Set 1 113
Figure 51. Partitioning of Data Set 2 114
Figure 52. Fragments of Variability Models of Cluster 1 116
Figure 53. Example of Fuzzy Rule-based Architecture 120
Figure 54: An architecture of the TSK model 121
Figure 55: Processing realized by the neuron 122
Figure 56. Example Time Series 123
Figure 57. Prototypes for clustering with FCM (a) & (b) and proximity fuzzy clustering (c) and (d) for *c*=2 and *c*=4 respectively 124
Figure 58. Rule antecedents of the proximity architecture for c=2 130
Figure 59. Error time series for Mackey-Glass (t=1749–2999) (a); Lorenz (t=14384–16383) (b); Wolf Sunspot (t=2249–3080) (c); oil prices (t=64–127) (d); star brightness (t=450–599) (e); IBM stock prices (t=2250–3332) (f) 131
Figure 60. Structural musical segmentation 133
Figure 61. Schematic Overview of Musical Segmentation 134
Figure 62. Cluster contiguity example 137

List of Tables

Table 1. Common nomenclature 5
Table 2. Common hierarchical clustering distance functions 7
Table 3. Common kernel functions 15
Table 4. Expressions for determining explicit prototypes 18
Table 5. Positive semi-definite kernel functions 50
Table 6. Conditionally positive semi-definite kernel functions 50
Table 7. Common Proximity functions 51
Table 8. Results of Synthetic data sets 53
Table 9. UCI Machine Learning Results 55
Table 10. Example of proximity hints of songs 58
Table 11. Synthetic data without kernel parameter learning 64
Table 12. Synthetic data with FCM and Gustafson-Kessel FCM 64
Table 13. Machine learning data without kernel parameter learning 68
Table 14. Machine learning data with FCM and Gustafson-Kessel FCM 68
Table 15. Synthetic data – Gaussian kernel parameter learning (no active learning) 73
Table 16. Synthetic data – Gaussian kernel parameter learning (active learning) 73
Table 17. Synthetic data – polynomial kernel parameter learning (no active learning) 75
Table 18. Synthetic data – polynomial kernel parameter learning (active learning) 75
Table 19. Machine Learning data – Gaussian kernel parameter learning (no active learning) 76
Table 20. Machine Learning data – Gaussian kernel parameter learning (active learning) 77
Table 21. Machine learning data – polynomial kernel parameter learning (no active learning) 78
Table 22. Machine learning data – polynomial kernel parameter learning (active learning) 79
Table 23. Construction of the views on synthetic data 86
Table 24. Performance on synthetic data 86
Table 25. Construction of the views on real-world data 86
Table 26. Performance on real-world data 86
Table 27. Construction of the views on synthetic data 87
Table 28. Performance on synthetic data 87
Table 29. Weighting of the views 87
Table 30. Construction of the views on real-world data 88
Table 31. Performance on synthetic data 88
Table 32. Weighting of the views 88
Table 33. Proximity values of the synthetic graph data set given in Figure 38 95
Table 34. Set-based Representations of Variability Models 102
Table 35. Proximity values of the synthetic graph data set given in Figure 46 107
Table 36. Clustering results for data set 1 112
Table 37. Clustering results for data set 2 112
Table 38. Cardinalities of the fuzzy clusters 112
Table 39. One-step prediction results 128
Table 40. Proximity-based time series prediction architecture parameters 128
Table 41. p-step prediction results 132
Table 42. Parameters for p-step prediction 132
Table 43. Musical segmentation performance 138
Table 44. Musical segmentation parameters 138
Table 45. Variability Models of Data Set 1 156
Table 46. Variability Models of Data Set 2 157
Table 47. Structure of the song "Yellow" (S=10, hopSize=0.690s) 158
Table 48. Structure of the song "A Message" (S=8, hopSize=0.627s) 158
Table 49. Structure of the song "Fix You" (S=10, hopSize=0.572s) 158
Table 50. Structure of the song "Swallowed in the Sea" (S=8, hopSize=0.423s) 158

Table 51. Structure of the song "Talk" (*S*=9, hopSize=0.560s) 158
Table 52. Structure of the song "A Rush of Blood to the Head" (*S*=11, hopSize=0.566) 158
Table 53. Structure of the song "In My Place" (*S*=10, hopSize=0.622s) 158
Table 54. Structure of the song "Politik" (*S*=9, hopSize=0.708s) 158
Table 55. Structure of the song "God Put a Smile Upon Your Face" (*S*=11, hopSize=0.716s) 158
Table 56. Structure of the song "The Scientist" (*S*=9, hopSize=0.822s) 159

1. INTRODUCTION & MOTIVATION

An important and growing area of research is the field of computational intelligence (CI). It is a collection of paradigms and algorithms used by scientists and engineers to accomplish tasks such as classification, clustering, approximation learning and prediction. CI is a hybrid of fuzzy sets, neural networks and evolutionary optimization that form an integral part of the design of an intelligent system, c.f. [102][158]. The algorithms and paradigms stemming from CI have the important property of being able to learn from data. A vast majority of these algorithms fall under the traditional dichotomy of unsupervised and supervised learning [102]. These frameworks are illustrated in Figure 1.

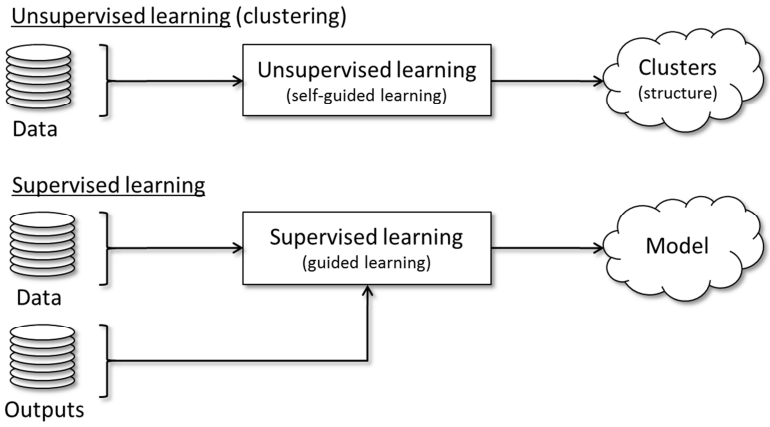


Figure 1. Unsupervised learning and supervised learning

The distinctive feature of supervised learning is that there are outputs which are needed to build an input-output mapping. Unsupervised learning on the other hand does not use outputs in the learning process but rather employs self-directed learning that discerns structure in data.

A third paradigm of learning that is gaining increased interest enhances traditional self-directed learning, i.e. unsupervised learning, by making use of some hints that guide the learning process, cf.
[157][26][161][162][16][34][4][22][208][122][43][46][25][179][13][217][135][218][33][80][117][52][74][165][166]. Undoubtedly, this is a highly promising and practically feasible direction in clustering that has the potential to enrich the learning process by accommodating domain knowledge [161]. As an example, we can consider a music recommendation system where the objective is to personalize the recommendation of music for each user based on their preferences. The problem is depicted in Figure 2.

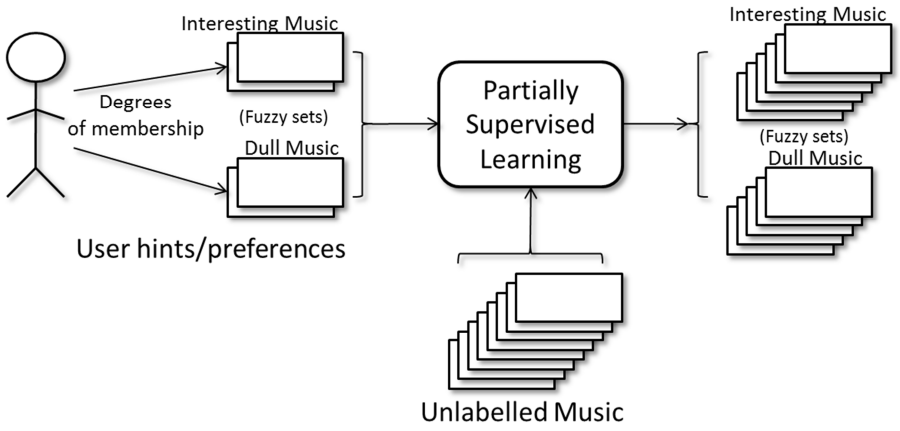


Figure 2. Partially supervised personalized music recommendation system

The idea is to request a small number of hints from the user in order to construct a model of the data that describes the user's preferences. Due to the enormous amount of music available, it is unrealistic and expensive to request the user to form a large training data set required of a conventional classifier to build the model. Hence, a partially supervised architecture is better suited for this task. Figure 3 illustrates the general framework of a partially supervised learning system.

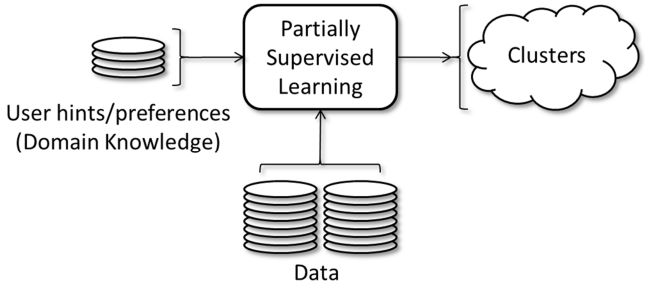


Figure 3. Partially supervised learning

Several outstanding research questions remain that need to be addressed in this area of partially supervised learning. One of the prominent questions remaining is one of a philosophical point of view: what is the best way to exploit this domain knowledge in a partially supervised framework? Despite the ubiquity of domain knowledge in common real-world data analysis problems, knowledge hints are rarely considered due to a lack of meaningful and algorithmically sound mechanisms available to incorporate them. Part of the issue lies with the many different forms of domain knowledge available. For example, there are constraint labels of many different forms, class labels, proximity labels, output labels, time stamps, etc, all depending on the problem at hand. Therefore, it is imperative that algorithmic frameworks for partially supervised learning be both well-supported and inspired by practical applications. It is with that reasoning that novel developments presented in this research will be two-fold, involving both theory and the application of partially supervised learning.

Second, the selection of the distance function and its parameters is a considerable challenge in clustering and has important implications on the shapes of the clusters. Partially supervised learning can be effective in adapting the parameters of the distance function and thereby play an integral role in any unsupervised learning framework. This leads to an important research

question regarding how it is best to exploit knowledge hints in order to adapt the distance function to a given problem. It is common to see the Euclidean distance in many clustering approaches; however, this metric tends to form hyper-spherical cluster geometries. While various distance functions have been introduced to address this restriction by providing increased flexibility, there often lacks mechanisms to determine the optimal parameters of these functions. The kernelization of clustering algorithms is one of the latest trends in producing flexibility distance functions, cf. [77][76]. Partial supervision is of prime interest in this application since the hints can be used to determine the optimal parameters for a particular kernel function.

A third important question deals with the selection of the user hints. Specifically the problem is in finding a small amount of data that need to be labeled. This area of research is called active learning and has received significant attention in problems where the objective is to select a small training data set from a large repository of data to discover a model for classification or regression purposes. The goal is to obtain high accuracy and strong generalizability on a concise training data set. Active partially supervised learning is a related problem with a slightly different objective: to determine what hints will be most beneficial in the learning. Often it will be the case where we have the ability to request some hints either from a human expert or automatically from a machine. However, since labelling is a taxing process, it is important to include an active component in the learning that is able to choose a small yet highly informative set of patterns to be labelled.

The aim of this study is to address many of these challenges presented in partially supervised learning by developing a partially supervised framework capable of exploiting proximity domain knowledge to benefit the clustering. Specifically, these objectives are to

- develop a partially supervised learning framework for clustering with proximity hints (domain knowledge),
- present an approach to learning the parameters of the proximity function using proximity hints,
- introduce an active learning framework for selecting the labeled patterns that is applicable to partially supervised learning, and
- outline a framework for clustering with multiple sources of proximity information.

The main points of the theoretic developments are discussed. Firstly, a proximity fuzzy clustering framework forms the basis of this research. It clusters data according to a proximity function that returns the degree of similarity between patterns – a concept highly related to kernel functions and kernel methods. The parametric optimization of the proximity function is the second accomplishment. A third novelty is an active learning framework that selects the optimal set of patterns to be labeled according to a measure of fuzziness. Finally, multi-proximity fuzzy clustering that reconciles proximity information coming from multiple sources is formulated.

Several real-world applications demonstrate how the new framework improves upon the current state-of-the-art. Several important applications of partially supervised clustering which will be expounded upon in the thesis are the

- adaption of the distance function and kernel function,
- prediction and clustering of time series,
- clustering of graphs in software development, and
- segmentation of music into meaningful structural sequences.

When learning the distance function (and kernel function), active learning is employed to select the patterns for which we require proximity hints. Most existing approaches in semi-supervised metric learning do not develop an active learning mechanism in their approach. Additionally,

most assume the available domain knowledge consists of pair-wise constraints (must link and cannot link) that dictate whether or not patterns should be in the same cluster. However, these constraints are Boolean and very limiting. A more general approach is to make use of proximity hints that indicate the degree of similarity between patterns.

The next problem of time series prediction is well studied and boasts numerous prediction architectures, each with advantages and disadvantages. A popular architecture is the fuzzy-rule based system since it provides a way to combine the predictive power of several simple local prediction models. The construction of the antecedents in these rules is important for constructing effective local models in the consequents. A system is presented and evaluated that constructs the antecedents with proximity fuzzy clustering. The benefits of the approach are a significant reduction in the number of rules in the architecture.

The third application is the clustering of a collection of graphs and is an important research problem in computing science. One of the main novelties in this application is the measure of proximity used between graphs since it is based on both their structure and semantic similarity. The semantics of the graphs come from feature trees in computing science which describe the properties or features of a software product based on a requirements specification.

Finally, the segmentation of music into a structural partition, namely intro, verse, chorus, is an area of research that could eventually facilitate more effective indexing and searching of a large database of music. The idea is to determine the structural information in a song using proximity fuzzy clustering aided by some temporal domain knowledge hints.

With each of these applications, how the partially supervised framework improves upon traditional clustering frameworks is investigated. Furthermore, the benefits procured by incorporating knowledge hints in the clustering are demonstrated.

The common nomenclature used throughout this document is provided in Table 1. We use standard matrix notation and set notation. We will use non-italicized capital letters to denote matrices, bold-face non-italicized lower-case letters to denote vectors and bold-face, italicized capital letters to denote vector spaces.

Table 1. Common nomenclature

Symbol	Description
$\boldsymbol{F}$	Feature space, e.g. $\mathbb{R}^d$
N	Number of data
d	Dimensionality of data
$\mathbf{X}=\{\mathbf{x}_k \in \boldsymbol{F} \mid k=1\ldots N\}$	Data set
c	Number of clusters
K	Number of classes
$\mathbf{C}_i \subset \mathbf{X}$ for $i=1\ldots c$	Clusters
$\mathbf{U}=[u_{ik}]$ for $k=1\ldots N, i=1\ldots c$	Partition matrix
$\mathbf{V}=\{\mathbf{v}_i \in \boldsymbol{F} \mid i=1\ldots c\}$	Set of prototypes
$\boldsymbol{H}$	Kernel space (higher dimensional space)
$\Phi(\mathbf{x}) \in \boldsymbol{H}$	Non-linear mapping to kernel space
$K(\cdot,\cdot)$	Kernel function
$p(\cdot,\cdot)$	Proximity
$d(\cdot,\cdot)$	Distance
$\mathbf{Q}=[q_{ij}]$ for $i=1\ldots c, j=1\ldots N$ where $q_{ij} \equiv p(\mathbf{x}_j, \mathbf{v}_i)$	Proximity matrix of patterns to clusters
$\mathbf{P}=[p_{ij}]$ for $i=1\ldots N, j=1\ldots N$	Proximity matrix of patterns
M	Number of labeled patterns
$\hat{\mathbf{X}}=\{\hat{\mathbf{x}}_k \mid k=1\ldots M\},\ \hat{\mathbf{X}} \subset \mathbf{X}$	Data set of labeled patterns
$a \wedge b$	Minimum
$a \vee b$	Maximum
$\|\bullet\|$	Euclidean norm
$\|\bullet\|_{\mathbf{A}}$	Mahanabolis norm for positive semi-definite matrix A
$\|\bullet\|_F$	Frobenius matrix norm
$\ell(\bullet)$	Labeling function of a single pattern
$\mathcal{L}(\bullet,\bullet)$	Relational labeling function of two patterns

2. BACKGROUND

In this chapter, a background in the traditional learning dichotomy of unsupervised and supervised learning is given. The most common frameworks, architectures and algorithms in unsupervised and supervised learning are outlined to provide a basis on which to discuss a partially supervised approach in subsequent chapters. There are many algorithms available in unsupervised learning including

- k-means clustering,
- Fuzzy C-Means (FCM) clustering,
- agglomerative & divisive hierarchical clustering,
- minimum entropy clustering (k-nearest neighbor),
- relational fuzzy clustering,
- kernel clustering, and
- spectral clustering.

There are numerous well-known supervised architectures that will also be briefly discussed including

- feed-forward neural networks,
- recurrent neural networks,
- radial basis function neural networks, and
- support vector machines (SVMs).

2.1. UNSUPERVISED LEARNING

The basic clustering (i.e. unsupervised) problem aims to partition data into clusters, i.e. learn the structure. The various algorithms are designed around minimizing a particular criterion (objective function). Each algorithm will be discussed in terms of the objective function minimized and the benefits and issues surrounding its design and implementation.

The data used in most existing clustering frameworks is a set of real-valued *d*-dimensional vectors, i.e. $\boldsymbol{F} = \mathbb{R}^d$. The goal of a clustering algorithm is to produce "*c*" clusters C_i, *i*=1...*c*, that form a partition of the data X. The definition of a partition from set theory imposes the following conditions on the clusters, i.e.

$$\begin{aligned} &\mathrm{C}_i \cap \mathrm{C}_j = \varnothing, \forall i, j = 1...c, j \neq i \\ &\bigcup_{i=1}^{c} \mathrm{C}_i = \mathrm{X} \end{aligned} \qquad \textbf{(2-1)}$$

In [19], the author discusses clusters where degrees of membership are allowed in the partition.

$$\mathrm{U} = [u_{ik}],\ u_{ik} \in [0,1],\ i = 1...c, k = 1...N \qquad \textbf{(2-2)}$$

Here, u_{ik} designates the membership of the *k*th pattern to the *i*th cluster. We will use the term *partition* (denoted by the matrix U) throughout the document to describe a partition with degrees of membership (fuzzy partition).

2.1.1. AGGLOMERATIVE HIERARCHICAL CLUSTERING

One of the most common and oldest unsupervised learning approaches is hierarchical clustering, cf. [53][141][95][46][17][81]. While there are two forms of hierarchical clustering –

agglomerative and divisive – agglomerative hierarchical clustering is the predominant method and is the focus of this section. The objective of hierarchical clustering is to obtain clusters C_i, i=1...c, with minimal cluster variance.

The algorithm is initialized by assigning each pattern to its own cluster, i.e. $C_i=\{\mathbf{x}_i\}$ for all i=1…N. It proceeds to merge the two closest clusters together and continues to merge the two closest clusters until the desired number of clusters is reached. The closeness of clusters is measured using a distance function. Some of the common distance functions are illustrated in Figure 4.

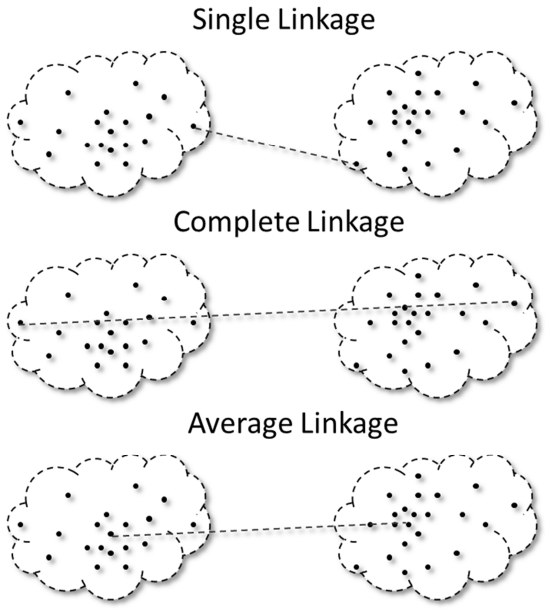


Figure 4. Cluster distance measures

The single linkage distance function returns the distance between the two closest patterns in each cluster. The complete linkage is the opposite and returns the distance between the two furthest patterns in each cluster. The average linkage distance function returns the average of the distances between the patterns in each cluster. The expressions for common distance functions are given in Table 2, cf. [53].

Table 2. Common hierarchical clustering distance functions

Single linkage	$d_{\min}(C_1,C_2)=\min_{\forall \mathbf{x}_1\in C_1,\mathbf{x}_2\in C_2}\left\|\mathbf{x}_1-\mathbf{x}_2\right\|$	The distance between clusters C_1 and C_2 is the distance between their nearest patterns
Complete linkage	$d_{\max}(C_1,C_2)=\max_{\forall \mathbf{x}_1\in C_1,\mathbf{x}_2\in C_2}\left\|\mathbf{x}_1-\mathbf{x}_2\right\|$	The distance between clusters C_1 and C_2 is the distance between their furthest patterns
Average linkage	$d_{\text{avg}}(C_1,C_2)=\frac{1}{\left\|C_1\right\|\left\|C_2\right\|}\sum_{\forall \mathbf{x}_1\in C_1}\sum_{\forall \mathbf{x}_2\in C_2}\left\|\mathbf{x}_1-\mathbf{x}_2\right\|$	The distance between clusters C_1 and C_2 is the average distance between each of their patterns

The merging can be described by a hierarchy, viz. a dendogram. An example of a dendrogram is given in Figure 5.

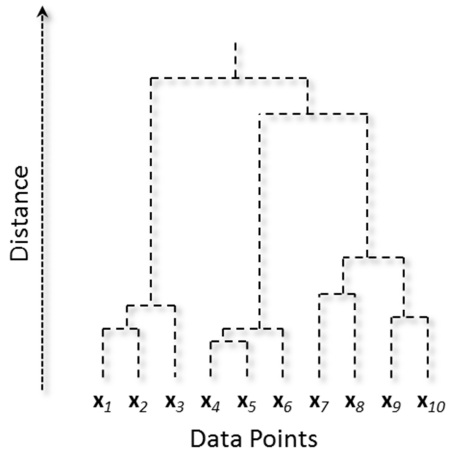


Figure 5. Dendrogram example

The agglomerative hierarchical clustering algorithm is described in pseudo-code, cf. [53].

Agglomerative Hierarchical Clustering

Input: X, $c_{desired}$

Output: C_i for i=1…$c_{desired}$

1. Form initial set of clusters for k=1…N
   a. $C_k = \{\mathbf{x}_k \in X\}$
2. c = N
3. Do
   a. Find the closest pair of clusters, C_i and C_j
   b. Merge clusters C_i and C_j
   c. Delete cluster C_j
   d. c = c - 1
4. Until $c = c_{desired}$

One of the advantages of the hierarchical clustering algorithm is its simplicity. A variety of different distance functions can be used providing many useful variations. However, the computational complexity of the algorithm is fairly high, $O(N^3)$, and thus scales poorly to large data sets.

2.1.2. K-MEANS CLUSTERING

Like hierarchical clustering, K-Means clustering is one of the first clustering algorithms to be developed, cf. [17][53]. The problem partitions data using a set of mean vectors, aka prototypes, belonging to each cluster and aims to minimize the variance with each cluster. The objective function minimized by K-Means is given by the expression

$$Obj = \sum_{i=1}^{c} \sum_{\mathbf{x} \in C_i} \left\| \mathbf{x} - \mathbf{v}_i \right\|^2 \qquad \textbf{(2-3)}$$

which is optimized with respect to the prototypes $\mathbf{v}_i$ (or the mean of the clusters) and pattern-to-cluster assignments C_i, i=1…c.

This objective is minimized by an iterative calculation of prototypes and cluster assignments. The expression for prototypes $\mathbf{v}_i$ is given by

$$\mathbf{v}_i = \frac{1}{|C_i|}\sum_{\mathbf{x}\in C_i}\mathbf{x} \qquad \textbf{(2-4)}$$

which is the mean of the patterns that belong to the *i*th cluster, *i*=1…*c*. The cluster assignments are then determined by selecting the prototype that is closest. In summary the algorithm determines the clusters in the following steps [53].

K-Means

Input: X, *c*

Output: V, C_i for *i*=1…*c*

1. Initialize the set of prototypes $\mathbf{v}_i$ for *i*=1…*c*
2. Do
    a. Assign each pattern in X to the cluster with the closest prototype (i.e. mean)
    b. Recompute the prototypes using expression (2-4)
3. Until convergence in prototypes, i.e. $\sum_{i=1}^{c}\left\|\mathbf{v}_i(iter)-\mathbf{v}_i(iter-1)\right\| < \varepsilon$

Note that ε is a small constant that determines the tolerance level in the change of prototypes allowed before halting the algorithm. A significant advantage of K-Means is that it is computationally efficient, O(*Ncd*). However, it suffers from imposing clusters on the data that are roughly equal in size and have hyper-spherical geometries. These assumptions are rarely true in real-world data.

2.1.3. Fuzzy C-Means (FCM) Clustering

K-Means was extended to clusters with grades of membership by Bezdek with the introduction of the Fuzzy C-Means (FCM) algorithm [19][162][163]. Like K-Means, the aim of FCM is to determine structure in the data represented by a set of "*c*" prototypes. Patterns are assigned membership values, i.e. a partition matrix U, to the clusters based on their distance from the prototypes. With FCM, one is concerned with discovering the prototypes and partition matrix U that minimizes the objective

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m}\left\|\mathbf{x}_k-\mathbf{v}_i\right\|^2 \qquad \textbf{(2-5)}$$

where m>1 is the fuzzification coefficient that controls the fuzziness of the partition. The minimization of (2-5) is subject to the constraint

$$\sum_{i=1}^{c} u_{ik} = 1 \qquad \textbf{(2-6)}$$

k=1…*N*. The constraint is handled by introducing a Lagrange multiplier λ thereby resulting in the objective

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m}\left\|\mathbf{x}_k-\mathbf{v}_i\right\|^2 + \lambda\sum_{k=1}^{N}\sum_{i=1}^{c}\left(u_{ik}-1\right) \qquad \textbf{(2-7)}$$

The minimum of (2-7) with respect to the prototypes is found by computing the gradient $\nabla_{\mathbf{v}_i} Obj = \mathbf{0}$ and solving for $\mathbf{v}_i$

$$\nabla_{\mathbf{v}_i} Obj = 2\sum_{k=1}^{N} u_{ik}^{m}\left(\mathbf{x}_k-\mathbf{v}_i\right) = \mathbf{0} \qquad \textbf{(2-8)}$$

The expression for the prototypes is thereby given as

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik}^m \mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}^m} \quad \textbf{(2-9)}$$

for *i*=1…*c*. In a similar manner, the partial derivative of the objective with respect to membership is given by

$$\frac{\partial Obj}{\partial u_{ik}} = m u_{ik}^{m-1} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 - \lambda = 0 \quad \textbf{(2-10)}$$

Solving for u_{ik} produces the expression

$$u_{ik} = \left(\frac{\lambda}{m \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2} \right)^{\frac{1}{m-1}} \quad \textbf{(2-11)}$$

By making use of the constraint in expression (2-6), the Lagrange multiplier is eliminated from the expression resulting in the formulation of the membership grades

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2}{\left\| \mathbf{x}_k - \mathbf{v}_j \right\|^2} \right)^{\frac{1}{m-1}}} \quad \textbf{(2-12)}$$

The FCM algorithm is an execution of the expressions (2-9) and (2-12) iteratively until the partition does not change significantly, cf. [19]. In summary the algorithm determines fuzzy clusters in the following steps.

FCM

Input: X, *c, m*
Output: V, U

1. Initialize membership U satisfying expression (2-6).
2. Do
    a. Calculate the cluster prototypes V using expression (2-9)
    b. Calculate the membership U using expression (2-12)
3. Until convergence in the membership U, i.e. $\left\| \mathrm{U}(iter) - \mathrm{U}(iter-1) \right\|_{\mathrm{F}} < \varepsilon$

Note that ε is a small constant that determines the tolerance level in the change of the membership matrix allowed before halting the algorithm. The benefit of FCM is that the partitioning allows for grades of membership. FCM also has a significant advantage in terms of computational efficiency, i.e. O(*Ncd*). However, the Euclidean metric often used with FCM is somewhat limiting as in many problems this metric may not be an appropriate choice.

2.1.4. Gustafson-Kessel Fuzzy Clustering

In order to alleviate the limitations of the Euclidean distance, a weighted Euclidean distance can be used to permit greater flexibility in the clusters. Gustafson-Kessel FCM, cf. [114][116][6][84], provides this increased flexibility by making use of the Mahanalobis distance. The aim of Gustafson-Kessel FCM is to determine a membership matrix U and "*c*" prototypes that partitions the data. The Gustafson-Kessel FCM algorithm minimizes the following objective function

$$Obj = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^m \left\| \mathbf{x}_k - \mathbf{v}_i \right\|_{\mathrm{A}_i}^2 \quad \textbf{(2-13)}$$

where A_i is a positive semi-definite matrix in the Mahanalobis distance. The minimization of (2-13) is subject to the standard FCM constraint in expression (2-6). The matrix A_i is computed from the inverse of the fuzzy covariance matrix Cov_i for i=1…c.

$$\mathrm{A}_i = \left(\rho_i \left|\mathrm{Cov}_i\right|\right)^{\frac{1}{d}} \mathrm{Cov}_i^{-1} \qquad \textbf{(2-14)}$$

The value ρ_i is a positive scaling parameter. It is common to encounter $\rho_i = 1$. The calculations of the fuzzy covariance matrix are governed by the expression

$$\mathrm{Cov}_i = \frac{\sum_{k=1}^{N} u_{ik}^m \left(\mathbf{x}_k - \mathbf{v}_i\right)\left(\mathbf{x}_k - \mathbf{v}_i\right)^{\mathrm{T}}}{\sum_{k=1}^{N} u_{ik}^m} \qquad \textbf{(2-15)}$$

Following the same derivation as with FCM, the expression for the prototypes is formulated by finding the gradient of (2-13) with respect to $\mathbf{v}_i$ equal to zero, i.e. $\nabla_{\mathbf{v}_i} Obj = \mathbf{0}$. The expression for computing the prototypes is identical to FCM, see (2-9). The membership matrix is found by determining the partial derivative $\frac{\partial Obj}{\partial u_{ik}} = 0$ with respect to $u_{i,}$, i.e.

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\left\|\mathbf{x}_k - \mathbf{v}_i\right\|_{\mathrm{A}_i}^2}{\left\|\mathbf{x}_k - \mathbf{v}_j\right\|_{\mathrm{A}_j}^2} \right)^{\frac{1}{m-1}}} \qquad \textbf{(2-16)}$$

In the worst case scenario, there is no inverse for Cov_i in which case the matrix A_i reduces to the identity matrix and Gustafson-Kessel FCM becomes identical to FCM. The algorithm proceeds in the following steps

Gustafson-Kessel FCM

Input: X, *c, m,* $\boldsymbol{\rho}$

Output: V, U

1. Initialize membership U subject to expression (2-6)
2. Do
    a. Calculate the cluster prototypes V using expression (2-9)
    b. Calculate the fuzzy covariance matrix Cov_i using expression (2-15)
    c. Calculate the Mahanalobis matrix A_i using expression (2-14)
    d. Calculate the membership u using expression (2-16)
3. Until convergence in the membership U, i.e. $\left\|\mathrm{U}(iter) - \mathrm{U}(iter-1)\right\|_{\mathrm{F}} < \varepsilon$

Gustafson-Kessel has a significant advantage over FCM in that elongated cluster geometries are possible. However, Gustafson-Kessel only benefits when there exists an inverse for the fuzzy covariance matrices.

2.1.5. Fuzzy C-Varieties

Traditional FCM uses prototypes that are specified by a point in the feature space. Bezdek, et al extend the concept of a point prototype to a hyper-plane *variety*, cf. [20][21]. The idea is to allow for line-clusters as opposed to spherical cluster shapes discovered using standard FCM. The problem involves computing a membership matrix U and the variety of each cluster. A variety is an r-dimensional hyperplane where $r \le d$. The objective function minimized in Fuzzy C-Varieties (FCV) is

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m} d_{\mathrm{A}}^{2}\left(\mathbf{x}_k, Var_{ri}\right) \tag{2-17}$$

where *m* is the fuzzification coefficient, *c* is the number of clusters, *N* is the number data, U=$\{u_{ik}\}$ the membership, and Var_{ri} is the cluster variety for *i*=1…*c*. The objective function is subject to the same constraints as standard FCM, i.e. (2-6). The distance between data and varieties is given by

$$d_{\mathrm{A}}^{2}\left(\mathbf{x}_k, Var_{ri}\right) = \left\|\mathbf{x}_k - \mathbf{v}_i\right\|_{\mathrm{A}}^{2} - \alpha\sum_{j=1}^{r}\left(\left(\mathbf{x}_k - \mathbf{v}_i\right)^{\mathrm{T}}\mathrm{A}\mathbf{b}_{ij}\right)^{2} \tag{2-18}$$

using the Mahanabolis distance with positive semi-definite matrix A by projecting patterns onto $\mathbf{b}_{ij}$. The parameter $\alpha \in [0,1]$ controls the effect of the variety on the cluster shape, $\mathbf{v}_i$ is a prototype point on the variety (hyper-plane) and $\mathbf{b}_{ij}$ is a normal vector to the variety for *j*=1…*r*, *i*=1…*c*, and *k*=1…*N*. When $\alpha = 0$ the variety reduces to a single prototype and when $\alpha = 1$ the variety is a line of infinite length. For $0 < \alpha < 1$, the variety is an ellipsoid. The parameter *r* controls the dimensionality of the variety. FCV reduces to standard FCM when either *r*=0 or $\alpha = 0$ since the variety reduces to a point. When *r*=1, the variety is a line, and when *r*=2, the variety is a plane. For *r*>2, the variety is a hyper-plane.

The algorithm for Fuzzy C-Varieties is summarized by the following pseudo-code.

Fuzzy C-Varieties Algorithm

Input: X, *c, m, r,* α

Output: V, U

1. Initialize membership U subject to expression (2-6)
2. Do
   a. Calculate the fuzzy varieties $Var_{ri}\left(\mathbf{v}_i; \mathbf{b}_{i1}, \ldots, \mathbf{b}_{ir}\right)$ for *i*=1…*c*
      i. Determine the *i*th prototype

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik}^{m}\mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}^{m}} \tag{2-19}$$

      ii. Determine *i*th fuzzy within cluster scatter matrix S_i

$$\mathrm{S}_i = \mathrm{A}^{\frac{1}{2}}\left(\sum_{k=1}^{N} u_{ik}^{m}\left(\mathbf{x}_k - \mathbf{v}_i\right)\left(\mathbf{x}_k - \mathbf{v}_i\right)^{\mathrm{T}}\right)\mathrm{A}^{\frac{1}{2}} \tag{2-20}$$

      iii. Determine the *r* largest eigenvalues ψ_{ij} and associated eigenvectors $\boldsymbol{\varphi}_{ij}$ for *j*=1…*r* from the cluster scatter matrix S_i and *i*=1…*c*
      iv. Determine the normal vector $\mathbf{b}_{ij}$ for *j*=1…*r* of the fuzzy variety Var_i

$$\mathbf{b}_{ij} = A^{-\frac{1}{2}}\boldsymbol{\varphi}_{ij} \tag{2-21}$$

   b. Calculate the membership matrix using $d_{\mathrm{A}}^{2}\left(\mathbf{x}_k, Var_{ri}\right)$ from expression (2-18)

$$u_{ik} = \frac{1}{\sum_{h=1}^{c}\left(\frac{d_{\mathrm{A}}^{2}\left(\mathbf{x}_k, Var_{ri}\right)}{d_{\mathrm{A}}^{2}\left(\mathbf{x}_k, Var_{ri}\right)}\right)^{1/m-1}} \tag{2-22}$$

   c. Calculate
3. Until convergence in the membership U, i.e. $\left\|\mathrm{U}(iter) - \mathrm{U}(iter-1)\right\|_{\mathrm{F}} < \varepsilon$

Fuzzy C-Varieties (FCV) offers the ability to form many interesting cluster shapes that are not possible with FCM, unfortunately, at the cost of increased computational complexity. The choice of *r* is not trivial and requires careful consideration in its selection.

2.1.6. RELATIONAL FCM

Relational Fuzzy C-Means (FCM) produces a partition U when provided with relational data, cf. [1]. Relational FCM accepts a matrix of relationships between objects as its input. We denote this matrix by $\mathrm{R} = [r_{ij}]$ for *j,k*=1...*N*. Relational FCM minimizes the standard FCM objective function, i.e.

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 \qquad \textbf{(2-23)}$$

which is subject to the same constraints as FCM, i.e. (2-6). It was shown in [1] that we can rewrite the distance in terms of the relational matrix R such that the patterns $\mathbf{x}_k$, *k*=1...*N*, are not needed in its determination. By Theorem 2 in [1], the authors show that the distance can be rewritten as

$$\left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 = \left(\mathrm{R}\,\mathbf{w}_i \right) - \frac{1}{2}\mathbf{w}_i\,\mathrm{R}\,\mathbf{w}_i \qquad \textbf{(2-24)}$$

where R=[r_{ij}], $r_{ij} = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2$ for $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ and *i,j*=1…*N*, and $\mathbf{w}_i$ is determined by the expression

$$\mathbf{w}_i = \frac{\left[u_{i1}^{m}, u_{i2}^{m}, u_{i3}^{m}, \ldots u_{iN}^{m} \right]^{\mathrm{T}}}{\sum_{k=1}^{N} u_{ik}^{m}} \qquad \textbf{(2-25)}$$

i=1...*c*. Finally, the membership matrix is computed using the same expression from standard FCM, i.e.

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d_{ik}^2}{d_{jk}^2} \right)^{\frac{1}{m-1}}} \qquad \textbf{(2-26)}$$

i=1...*c* and *k*=1...*N*.

The algorithm is described using the following pseudo-code.

Relational FCM

Input: R, c

Output: V, U

1. Initialize a membership matrix U subject to expression (2-6)
2. Do
    a. Determine c-mean ("prototype") vectors for *i*=1…*c* using expression (2-25)
    b. Calculate distances for *i*=1…*c* and *k*=1…*N* using expression (2-24)
    c. Calculate membership matrix for *i*=1…*c* and *k*=1…*N* using expression (2-26)
3. Until convergence in the membership, i.e. $\left\| \mathrm{U}(iter) - \mathrm{U}(iter-1) \right\|_{\mathrm{F}} < \varepsilon$

The advantage of relational FCM is that it can be applied to a broad class of problems whose objects are ill-defined in a $\mathbb{R}^d$ feature space. Unfortunately, relational FCM suffers from a serious shortcoming since the relationships are Euclidean distances, i.e. $r_{ij} = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2$ for $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ and *i,j*=1…*N* when in fact this may not be the case in general. An extension to

relational FCM was made in [2] for the general case when relationships are not a metric, i.e. when expression (2-24) does not satisfy the requirements of distance, i.e.

$$\begin{aligned} r_{ij} &= r_{ji} \quad \text{(symmetry)} \\ r_{ii} &= 0 \quad \text{(identity)} \\ r_{ij} &\geq 0 \quad \text{(non-negative)} \\ r_{ik} + r_{kj} &\geq r_{ij} \quad \text{(triangular inequality)} \end{aligned} \tag{2-27}$$

for all *i,j,k*=1...*N*. The authors in [2] accomplish this by adding a constant β to all elements in R except for the diagonal to ensure R does not have negative values.

2.1.7. KERNEL-BASED FUZZY CLUSTERING

Kernel-based fuzzy clustering is a relatively new area of fuzzy clustering. The motivation is to overcome the limitations of the Euclidean distance by introducing a kernel function in the calculation of the distance. The kernel function performs an arbitrary non-linear mapping Φ from the original *d*-dimensional feature space ***F*** to a space of higher dimensionality ***H*** (i.e. the kernel space) - possibly of infinite dimensionality, cf. [88][148]. Kernel methods originally appeared in classification frameworks such as Support Vector Machines (SVMs). The objective is to map the data to higher dimensional kernel space where the data was more likely to be linearly separable [88][148]. When kernelizing clustering algorithms, such as FCM, the objective is to seek non-hyper-spherical cluster geometries via the non-linear mapping. In other words, the clusters become "warped" through the inverse non-linear mapping Φ^{-1}. An example motivating the kernel trick is illustrated in Figure 6.

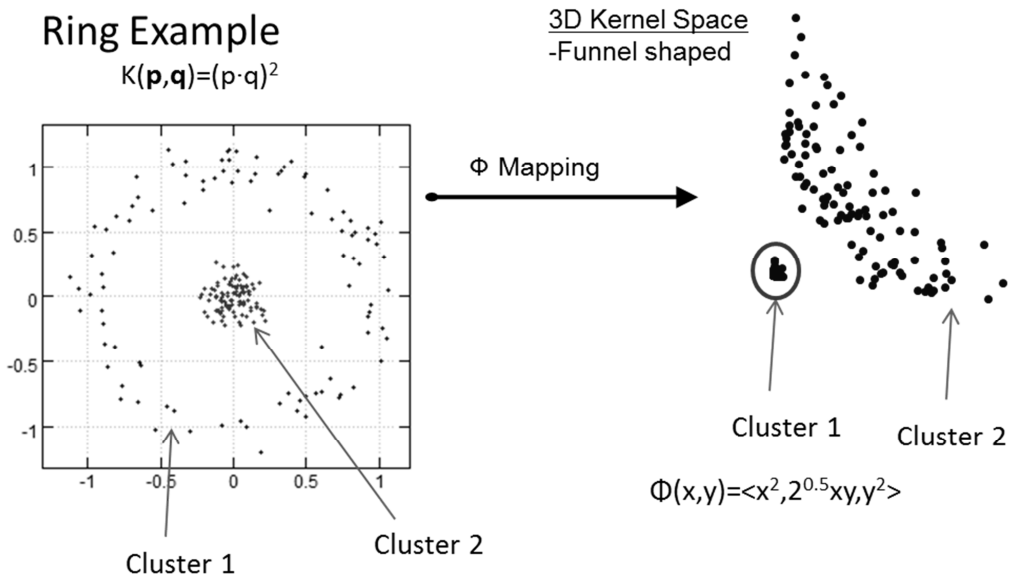


Figure 6. Kernel Example

By making use of a polynomial kernel and moving from a 2D feature space to a 3D kernel space, the ring data set become more suitable for classification algorithms since it is linearly separable as shown in Figure 6. The kernel space is also better suited for the problem of clustering compared with the original feature space.

Kernels takes advantage of the fact that dot products in the kernel space can be expressed by a kernel function *K* given by $K(\mathbf{x},\mathbf{y}) \equiv \Phi(\mathbf{x})^{\mathrm{T}}\Phi(\mathbf{y})$ where $\mathbf{x},\mathbf{y} \in \mathbb{R}^d$. Distances in the kernel space that can be expressed in terms of dot products (inner products) such as the popular Euclidean distance are easily calculated using the kernel function without explicitly knowing Φ. This is commonly known as the kernel trick. A positive semi-definite symmetric function is a kernel function [4]. Some common kernel functions are given in Table 3, cf. [148].

Table 3. Common kernel functions

Kernel Name	Kernel Function
Gaussian	$K(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2}}$, $\sigma^2 > 0$
Polynomial	$K(\mathbf{x},\mathbf{y}) = \left(\mathbf{x}^{\mathrm{T}}\mathbf{y} + \theta\right)^p$, $p \in \mathbb{N}$, $\theta \geq 0$
Sigmoidal	$K(\mathbf{x},\mathbf{y}) = \left(\kappa\left(\mathbf{x}^{\mathrm{T}}\mathbf{y}\right) + \theta\right)^p$, $\kappa,\theta \in \mathbb{R}$
Inverse Multiquadratic	$K(\mathbf{x},\mathbf{y}) = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{y}\|^2 + \theta}}$, $\theta \geq 0$

There are two major forms of kernel-based fuzzy clustering. The first one positions prototypes in the original feature space. These clustering methods will be referred to as KFCM-F (with F standing for the feature space). In the second category, abbreviated as KFCM-K, the prototypes are arranged in the kernel space where an approximation of the inverse mapping is used to determine their location in the original feature space.

KFCM-F minimizes an objective function subject to the same constraints as FCM (2-6), cf. [77][76].

$$Obj = \sum_{k=1}^{N}\sum_{i=1}^{c} u_{ik}^{m} \left\|\Phi\left(\mathbf{x}_k\right) - \Phi(\mathbf{v}_i)\right\|^2 \qquad \textbf{(2-28)}$$

The advantage of the KFCM-F clustering algorithm is that the prototypes reside in the feature space and are implicitly mapped to the kernel space through the use of the kernel function as depicted in Figure 7.

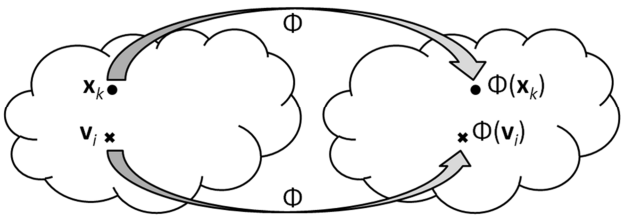


Figure 7. KFCM-F feature space and kernel space

By constraining ourselves to the Euclidean distance in $\left\|\Phi\left(\mathbf{x}_k\right) - \Phi(\mathbf{v}_i)\right\|$, the squared distance is computed in the kernel space using a kernel function such that

$$\begin{aligned}\left\|\Phi\left(\mathbf{x}_k\right) - \Phi(\mathbf{v}_i)\right\|^2 &= \Phi\left(\mathbf{x}_k\right)^{\mathrm{T}}\Phi\left(\mathbf{x}_k\right) - 2\Phi\left(\mathbf{x}_k\right)^{\mathrm{T}}\Phi\left(\mathbf{v}_i\right) + \Phi\left(\mathbf{v}_i\right)^{\mathrm{T}}\Phi\left(\mathbf{v}_i\right) \\ &= K\left(\mathbf{x}_k,\mathbf{x}_k\right) - 2K\left(\mathbf{x}_k,\mathbf{v}_i\right) + K(\mathbf{v}_i,\mathbf{v}_i)\end{aligned} \qquad \textbf{(2-29)}$$

If we confine ourselves further to the commonly used Gaussian kernel, then $K(\mathbf{x},\mathbf{x})=1$ and $\left\|\Phi(\mathbf{x}_k)-\Phi(\mathbf{v}_i)\right\|^2=2\left(1-K(\mathbf{x}_k,\mathbf{v}_i)\right)$. The optimization of the partition matrix U involves the use of the technique of Lagrange multipliers to handle the constraint given in (2-6). By solving the partial derivative of the objective with respect to membership, as with the derivation of the membership expression with FCM, one obtains

$$u_{ik}=\frac{1}{\sum_{j=1}^{c}\left(\frac{1-K(\mathbf{x}_k,\mathbf{v}_i)}{1-K(\mathbf{x}_k,\mathbf{v}_j)}\right)^{\frac{1}{m-1}}} \qquad \textbf{(2-30)}$$

for *i*=1, 2, … *c* and *k*=1, 2, … *N*. The calculation of the prototypes $\mathbf{v}_i$ for i=1, 2, … c is determined by solving the derivative of the objective equal to zero; however, the prototypes cannot be separated from the kernel hence we use an approximation by employing the prototypes from the previous iteration, producing the expression

$$\mathbf{v}_i(iter)=\frac{\sum_{k=1}^{N}u_{ik}^m K(\mathbf{x}_k,\mathbf{v}_i(iter-1))\mathbf{x}_k}{\sum_{k=1}^{N}u_{ik}^m K(\mathbf{x}_k,\mathbf{v}_i(iter-1))} \qquad \textbf{(2-31)}$$

The sequence of steps in the minimization of (2-28) is elaborated.

KFCM-F Algorithm

Input: X, *c, m,* σ^2 (Gaussian kernel width)
Output: V, U

1. Initialize membership U subject to expression (2-6)
2. Initialize prototypes $\mathbf{v}_i(0)$ for *i*=1…*c*
3. Do
    a. *iter*=*iter*+1
    b. Calculate the cluster prototypes V(*iter*) using expression (2-31)
    c. Calculate the membership U using expression (2-30)
4. Until convergence in the membership U, $\left\|\mathrm{U}(iter)-\mathrm{U}(iter-1)\right\|_{\mathrm{F}}<\varepsilon$

In [76] it is observed that the performance improvements offered by KFCM-F are not vast improvements compared with FCM and Gustafson-Kessel FCM. The reason for the lackluster performance of KFCM-F may be due to the restriction of the prototypes in the feature space.

A second approach to kernel-based fuzzy clustering, called KFCM-K, allows for greater flexibility with the prototypes since they are positioned in the higher dimensional kernel space. The objective function minimized by KFCM-K is subject to the constraint given by expression (2-6). KFCM-K minimizes the following objective function

$$Obj=\sum_{k=1}^{N}\sum_{i=1}^{c}u_{ik}^m\left\|\Phi(\mathbf{x}_k)-\mathbf{v}_i^{\Phi}\right\|^2 \qquad \textbf{(2-32)}$$

where $\mathbf{v}_i^{\Phi}$ for *i*=1…*c* are the implicit prototypes in the kernel space, cf. [77][76]. This approach provides increased freedom in the placement of the prototypes; however, the prototypes must be explicitly mapped back to the original feature space, see Figure 8.

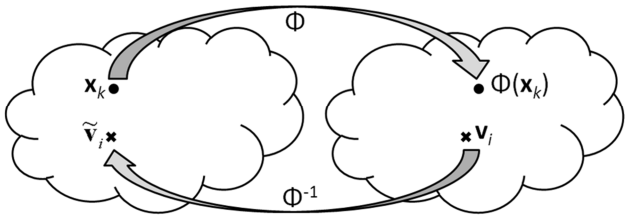


Figure 8. KFCM-K feature space and kernel space

The implicit prototypes are determined by solving the gradient of expression (2-32) equal to the zero vector, i.e. $\nabla_{\mathbf{v}_i^{\Phi}} Obj = \mathbf{0}$, producing the expression

$$\mathbf{v}_i^{\Phi} = \frac{\sum_{k=1}^{N} u_{ik}^m \Phi(\mathbf{x}_k)}{\sum_{k=1}^{N} u_{ik}^m} \tag{2-33}$$

for *i*=1…*c*. The membership can be found in the same manner by solving $\frac{\partial Obj}{\partial u_{ik}} = 0$. The membership is computed via the expression

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\left\| \Phi(\mathbf{x}_k) - \mathbf{v}_i^{\Phi} \right\|^2}{\left\| \Phi(\mathbf{x}_k) - \mathbf{v}_i^{\Phi} \right\|^2} \right)^{\frac{1}{m-1}}} \tag{2-34}$$

for *i*=1…*c* and *k*=1…*N*. However, the kernel mapping must be removed explicitly from the calculations. To do this we focus on the squared distance $\left\| \Phi(\mathbf{x}_k) - \mathbf{v}_i^{\Phi} \right\|^2$ in expression (2-34) and through substitution of expression (2-33), obtain a new expression with the kernel function.

$$\begin{aligned} \left\| \Phi(\mathbf{x}_k) - \mathbf{w}_i \right\|^2 &= \Phi(\mathbf{x}_k)^{\mathrm{T}} \Phi(\mathbf{x}_k) - 2\Phi(\mathbf{x}_k)^{\mathrm{T}} \mathbf{v}_i^{\Phi} + (\mathbf{v}_i^{\Phi})^{\mathrm{T}} \mathbf{v}_i^{\Phi} \\ &= K(\mathbf{x}_k, \mathbf{x}_k) - 2 \frac{\sum_{j=1}^{N} u_{ij}^m (iter) K(\mathbf{x}_k, \mathbf{x}_j)}{\sum_{j=1}^{N} u_{ij}^m (iter-1)} \\ &\quad + \frac{\sum_{j=1}^{N} \sum_{h=1}^{N} u_{ij}^m (iter-1) u_{ih}^m (iter-1) K(\mathbf{x}_j, \mathbf{x}_h)}{\left(\sum_{j=1}^{N} u_{ij}^m (iter-1) \right)^2} \end{aligned} \tag{2-35}$$

where $u_{ik}(iter-1)$ for *i*=1…*c* and *k*=1…*N* are the values of the membership in the previous iteration of the optimization.

KFCM-K Algorithm

Input: X, *c*, *m*, σ^2 (Gaussian kernel width)
Output: U

1. Initialize membership U(0) subject to expression (2-6)
2. Do
    a. *iter*=*iter*+1
    b. Calculate the membership U(*iter*) using expression (2-35) using the previous values U(*iter*-1)
3. Until convergence in the membership U, $\left\| \mathrm{U}(iter) - \mathrm{U}(iter-1) \right\|_{\mathrm{F}} < \varepsilon$

The inverse mapping of the implicit prototypes in the higher dimensional kernel space to explicit ones in the original feature space is an open problem. One method proposed in [227] minimizes

$$O = \sum_{i=1}^{c} \left\| \Phi(\mathbf{v}_i) - \mathbf{v}_i^{\Phi} \right\|^2 \quad \textbf{(2-36)}$$

with respect to $\mathbf{v}_i$ where $\mathbf{v}_i^{\Phi}$ for i=1…c (the implicit prototypes) is defined in expression (2-33) and $\mathbf{v}_i$ for i=1…c is the set of explicit prototypes. An iterative solution to the optimization is found by solving the gradient equal to zero, i.e. $\nabla_{\mathbf{v}_i} O = \mathbf{0}$. The expression for determining the explicit prototypes depends on the choice of kernel function [76], see Table 4.

Table 4. Expressions for determining explicit prototypes

Gaussian	Polynomial
$\mathbf{v}_i(iter) = \frac{\sum_{k=1}^{N} u_{ik}^m K(\mathbf{x}_k, \mathbf{v}_i(iter-1))\mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}^m K(\mathbf{x}_k, \mathbf{v}_i(iter-1))}$	$\mathbf{v}_i(iter) = \frac{\sum_{k=1}^{N} u_{ik}^m \left(\mathbf{x}_k{}^{\mathrm{T}} \mathbf{v}_i(iter-1) + \theta\right)^{p-1} \mathbf{x}_k}{\left(\mathbf{v}_i(iter-1)^{\mathrm{T}} \mathbf{v}_i(iter-1) + \theta\right)^{p-1} \sum_{k=1}^{N} u_{ik}^m}$

The algorithm for computing the prototypes is described in the following pseudo-code.

KFCM-K Prototypes Algorithm

Input: U, X, *c*, *m*, σ^2 (Gaussian kernel width)

Output: V

1. Initialize prototypes $\mathbf{v}_i(0)$
2. Do
   a. *iter*=*iter*+1
   b. Calculate the prototypes $\mathbf{v}_i$(*iter*) using the appropriate expression in Table 4and the previous prototypes $\mathbf{v}_i$(*iter*-1)
3. Until convergence in the prototypes V, $\sum_{i=1}^{c} \left\| \mathbf{v}_i(iter) - \mathbf{v}_i(iter-1) \right\| < \varepsilon$

The prototypes are calculated after the KFCM-K algorithm has determined a partition matrix U. The advantage of KFCM-K is that the prototypes are not restricted in the feature space since they are in the kernel space. However, both KFCM-F and KFCM-K require parametric tuning of the kernel function as investigated in [76].

2.1.8. Spectral Clustering

Spectral clustering is highly related to kernel clustering, c.f. [58][138]. It interprets the problem of clustering through a graph representation where data points are vertices and the similarities between vertices are weighted edges. The graph is complete, weighted and undirected. It is formalized by the tuple *Graph*(*Vertices*, A) where *Vertices*={$\mathbf{x}_1$, $\mathbf{x}_2$, …. $\mathbf{x}_N$} and A is an $N \times N$ symmetric matrix called an affinity or adjacency matrix. The adjacency matrix, sometimes called a proximity matrix, represents the weight of the edges in the graph. It is common to compute the entries of matrix A from a function $h(\mathbf{x}_i, \mathbf{x}_j)$ (such as the well-known Gaussian function) that measures the similarity between patterns [58].

$$a_{ij} = \begin{cases} h(\mathbf{x}_i, \mathbf{x}_j) \; for \; i \neq j \\ 0 \qquad otherwise \end{cases} \quad \textbf{(2-37)}$$

for i,j=1…N. An example of a simple graph is shown in Figure 9.

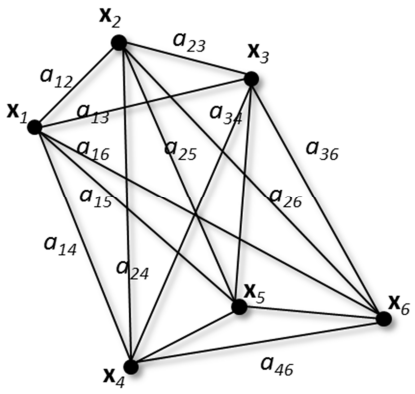


Figure 9. Example graph

It is possible to solve this graph clustering problem by constructing a Laplacian matrix L from the adjacency matrix A. There are several types of Laplacian matrices that can be fashioned; however, the traditional unnormalized Laplacian is given by

$$\mathrm{L} = Diag - \mathrm{A} \tag{2-38}$$

where *Diag* is the diagonal matrix of the degrees of the vertices, i.e.

$$Diag_{ii} = \sum_{j=1}^{N} a_{ij} \tag{2-39}$$

for *i*=1…*N*. Other types of Laplacian matrices are given in [58]. From the Laplacian matrix L, one can create the spectral decomposition which provides insight into the structure of the graph and can be used to form clusters of vertices. The most common approach [138] determines the "c" largest eigenvalues of L and their corresponding eigenvectors denoted by s_1, s_2, … s_c. The $N \times c$ matrix S is constructed by concatenating the eigenvectors forming the columns of S, i.e. S=[s_1;s_2;…s_c]. The K-Means clustering algorithm with the number of clusters equal to "c" is applied to S by treating the rows which are also of dimensionality "c" as separate data points.

Spectral Clustering Algorithm

Input: X, *c*

Output: C_i for *i*=1…*c*

1. Calculate the affinity matrix A using (2-37) where $h\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is commonly

$$h\left(\mathbf{x}_i, \mathbf{x}_j\right) = e^{-\frac{1}{\sigma^2}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} \tag{2-40}$$

2. Calculate the unnormalized Laplacian matrix L from (2-38)
3. Compute the "c" largest eigenvalues and their associated eigenvectors {s_1,s_2,…s_c}
4. Construct the $N \times c$ matrix S where the eigen-vectors form the columns, i.e. S=[s_1;s_2;…s_c]
5. C = K-Means(S,c)

Like relational FCM, spectral clustering discovers the structure in data from a set of relationships. However, with spectral clustering, the relationships are similarity values. Therefore, spectral clustering does not suffer from the same disadvantages as relational FCM. By restricting the similarity values to proximity values which are on the interval [0,1], spectral clustering becomes a method of clustering with proximity information. A serious disadvantage is that the prototypes

produced by the K-Means clustering lack useful meaning. In addition, the computational complexity is poor with large data sets due to the eigenvector computation.

2.1.9. MINIMUM ENTROPY CLUSTERING

Clustering can also be performed from an information-theoretic point of view, cf. [128][125]. Information theory was first introduced by Shannon in 1948 [182] where he showed that the amount of information contained in a variable could be quantified using entropy. The entropy of a random variable x_{rand} is mathematically given by the expression

$$H(x_{rand}) = -\sum_{\forall x} p(x_{rand}) \log_b p(x_{rand}) \quad \textbf{(2-41)}$$

where $H(x_{rand})$ is the entropy, x_{rand} is the random variable, $p(x_{rand})$ is a probability function, and b is the base of the logarithm, cf. [182]. Entropy measures the amount of disorder of a variable and is also termed self-information [125]. It was originally used in communication to measure the amount of information contained in a stream of data in order to maximize the amount of information sent per message with the fewest number of characters, i.e. data compression.

The concept of entropy can be applied to clustering by viewing the measure of entropy as a measure of uncertainty in data. Lower values of entropy indicate lower degrees of uncertainty (i.e. higher degrees of certainty). This is useful in clustering as we can maximize the entropy $H(\mathbf{x}|C)$ of data $\mathbf{x} \in X$ given a particular partition C. This type of entropy is referred to as conditional entropy. Clustering can be performed using posterior probabilities as done in [125] where the entropy maximized is denoted by $H(C|\mathbf{x})$.

2.1.10. ENSEMBLE CLUSTERING

Ensemble clustering, also known as consensus clustering, combines the results of several partitions by reconciling differences in their cluster structure to form a global structure. The concept is depicted in Figure 10.

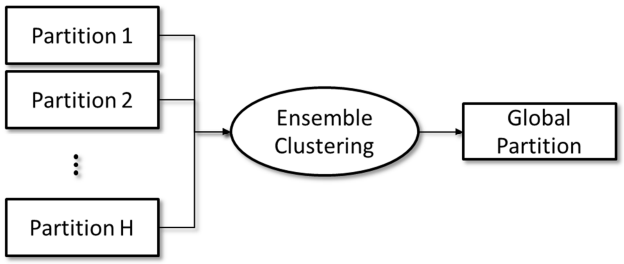


Figure 10. Ensemble clustering framework

Many of the ensemble-based clustering algorithms are based on the mechanism of voting where the principle is to select the cluster assignment with the most votes. There are several different approaches to ensemble clustering, including cluster-based similarity partitioning algorithm (CSPA) [191] and evidence accumulation clustering (EAC).

The CSPA approach builds a single similarity matrix by concatenating the partitions $U^{(1)}, U^{(2)}, \ldots U^{(H)}$ column-wise into a single partition block matrix $U=[U^{(1)} \mid U^{(2)} \mid \ldots U^{(H)}]$ where H is the number of partitions. It is typically assumed that the partitions are Boolean [191]. The co-association similarity matrix is constructed from the expression

$$\mathrm{S} = \frac{1}{H}\mathrm{U}\mathrm{U}^{\mathrm{T}} \tag{2-42}$$

where S is an $N \times N$ matrix denoting the similarity between each pattern. Any relational clustering approach can be used to cluster the similarity matrix S including spectral clustering, hierarchical clustering or relational FCM.

Cluster-based Similarity Partitioning Algorithm (CSPA)

Input: U, *H*, *c*

Output: C_i for *i*=1…*c*

1. Calculate the similarity matrix S using expression (2-42)
2. C = RelationalClustering(S, *c*)

A popular method of ensemble clustering based on voting is evidence accumulation clustering (EAC), cf. [200][59]. EAC constructs a co-association similarity matrix S using voting. The co-association similarity matrix is given by

$$\mathrm{S} = \left\{ s_{ij} = \frac{n_{ij}}{H} | i, j = 1 \ldots N \right\} \tag{2-43}$$

where $n_{ij} \in \mathbb{N}$ is the number of times $\mathbf{x}_i$ and $\mathbf{x}_j$ are the in the same cluster considering all *H* partitions. As with CSPA, any relational clustering algorithm can be used to cluster the similarity matrix S. The authors in [59] cluster the similarity matrix via single-link hierarchical clustering.

Evidence Accumulation Clustering (EAC) Algorithm

Input: U, *H*, *c*

Output: C_i for *i*=1…*c*

1. Calculate the similarity matrix S using expression (2-43)
2. C = RelationalClustering(S, *c*)

One of the primary disadvantages of these algorithms is that voting does not perform well with only a handful of partitions.

2.1.11. Evaluation of Clustering Performance

Evaluating the clustering result is important for measuring the quality of the partition. A common approach is to compare the partition to a ground truth partition such as the Normalized symmetric Mutual Information (NMI) criterion based on information theory and entropy. The performance index [51] is given by the expression

$$NMI = \frac{I(C,T)}{H(C) + H(T)} \tag{2-44}$$

where *C* is the partition produced by the clustering algorithm satisfying (2-1) and *T* is the ground truth partition (also satisfying (2-1)). The *I*(*C*,*T*) is a measure of information and measures the congruency of the two partitions *C* and *T*. *H*(*C*) and *H*(*T*) denote the entropy of the clustering and the ground truth, respectively. The computation of NMI commences by constructing a contingency matrix (sometimes called confusion matrix) denoted by G=[n_{ij}] for *i*=1…*c* and *j*=1…*K* where "*c*" is the number clusters produced by the clustering algorithm, *K* is the number classes in the ground truth partition and n_{ij} is the number of patterns that are labeled with class *j* and fall into cluster *i*. With the ground truth partition given by the sets T_j for *j*=1…*K* the contingency matrix is formalized as

$$G = [n_{ij}], n_{ij} = \left| C_i \cap T_j \right| \tag{2-45}$$

, i=1…c and j=1…K. The information content between the two partitions is computed from the contingency matrix G using

$$I(C,T) = \sum_{i=1}^{c} \sum_{j=1}^{K} \frac{n_{ij}}{N} \log \left(\frac{n_{ij} N}{\left(\sum_{i=1}^{c} n_{ij} \right) \left(\sum_{j=1}^{K} n_{ij} \right)} \right) \tag{2-46}$$

where N is the number of patterns in the data set. The entropies are calculated with the expressions

$$H(C) = -\sum_{i=1}^{c} \left(\sum_{j=1}^{K} \frac{n_{ij}}{N} \right) \log \left(\sum_{j=1}^{K} \frac{n_{ij}}{N} \right) \tag{2-47}$$

$$H(T) = -\sum_{j=1}^{K} \left(\sum_{i=1}^{c} \frac{n_{ij}}{N} \right) \log \left(\sum_{i=1}^{c} \frac{n_{ij}}{N} \right) \tag{2-48}$$

Large values of NMI indicate excellent performance while small values indicate poor performance. The NMI produces values on the interval [0,1] where 1 is a perfect match between the two partitions.

Another common performance index is the classification rate and is a measure of accuracy taken from classification problems. Unfortunately since clustering is unsupervised, there is no way to label the clusters with class labels. The most common tactic of assigning clusters with class labels is to select the class with the maximum count in the cluster. In this way, the classification rate can be computed directly from the contingency matrix using the following expression

$$CR = \frac{1}{N} \sum_{j=1}^{K} \max_{\forall i=1\ldots c} n_{ij} \tag{2-49}$$

The classification rate is on the unit interval [0,1] where .

There are a plethora of other performance measures that evaluate the quality of the clustering result without a ground truth partition, cf. [83][203]. One approach sometimes used for evaluating FCM clustering is the reconstruction error. It reconstructs the patterns using the prototypes V=[$\mathbf{v}_i$] and membership values U=[u_{ik}] for i=1…c and k=1…N, cf. [163]. The process involves reconstructing patterns based on the codebook (i.e., a collection of already constructed prototypes) and the computed membership values of the original patterns to the elements of the codebook [163]. The reconstructed patterns $\tilde{\mathbf{x}}_k$, k=1, 2, … N are calculated as follows:

$$\tilde{\mathbf{x}}_k = \frac{\sum_{i=1}^{c} u_{ik}^m \mathbf{v}_i}{\sum_{i=1}^{c} u_{ik}^m} \tag{2-50}$$

The error between the reconstructed pattern and the original one quantifies the quality of reconstruction provided by the fuzzy clustering (and a collection of the prototypes, in particular). The reconstruction error is expressed in the form of the following sum of distances

$$r = \frac{\sum_{k=1}^{N} \left\| \mathbf{x}_k - \tilde{\mathbf{x}}_k \right\|^2}{N} \quad \textbf{(2-51)}$$

Smaller values for the reconstruction error "*r*" indicate better clustering results (in terms of the reconstruction criterion introduced above). Clearly a lower number of prototypes implies higher values of the reconstruction error. As the number of prototypes gets closer to the number of patterns, the reconstruction error gets closer to zero. The reconstruction error is unlike the classification rate since it does not measure accuracy. Instead the reconstruction error measures the encoding and decoding performance of the patterns with respect to their prototypes and membership values. Visually, the reconstruction error is a measure for the spread of the prototypes across the feature space. In particular, the reconstruction error decreases as prototypes are situated centrally within dense regions of patterns in the feature space. Hence, reconstruction error measures the quality of the prototypes with regards to its representation of the data in terms of clusters.

There are a number of other unsupervised measures. Two of the most popular are described the Davies-Bouldin (DB) index, and the Dunn index. Both indices are based upon the dual objective of minimizing intra-cluster cohesiveness while maximizing inter-cluster separation. The DB index is calculated from the expression

$$\mathrm{DB} = \frac{1}{c} \sum_{i=1}^{c} \max_{j=1\ldots c, j \neq i} \left(\frac{\frac{1}{n_i} \sum_{\mathbf{x}_k \in C_i} d\left(\mathbf{x}_k, \mathbf{v}_i\right) + \frac{1}{n_j} \sum_{\mathbf{x}_k \in C_j} d\left(\mathbf{x}_k, \mathbf{v}_j\right)}{d\left(\mathbf{v}_i, \mathbf{v}_j\right)} \right) \quad \textbf{(2-52)}$$

where the bottom term $d\left(\mathbf{v}_i, \mathbf{v}_j\right)$ measures the distance between clusters (inter-cluster separation) and the top term measures the average distance within the cluster to the cluster prototype. The Dunn index is given by the expression

$$\mathrm{Dunn} = \frac{\min_{i=1\ldots c, \forall \mathbf{x}_k \in C_i} \left(\min_{j=1\ldots c, j \neq i, \forall \mathbf{x}_h \in C_j} d\left(\mathbf{x}_k, \mathbf{x}_h\right) \right)}{\max_{i=1\ldots c, \forall \mathbf{x}_k, \mathbf{x}_h \in C_i} d\left(\mathbf{x}_k, \mathbf{x}_h\right)} \quad \textbf{(2-53)}$$

2.2. SUPERVISED LEARNING

In this next section we discuss some common supervised learning algorithms including classic feed-forward neural networks, radial basis function neural networks, and support vector machines.

2.2.1. NEURAL NETWORKS

The neural network is a powerful architecture that is particularly effective at approximating continuous functions. Its popularity is due to a theorem proved by Kolomogorov [102]. It states that a feed-forward artificial neural network with a hidden layer consisting of non-linear activation functions is capable of approximating any continuous function to an arbitrary degree of accuracy – termed universal approximation. Interest waned when it became clear that a trained neural network was a black-box architecture and thus could not be interpreted easily. Still neural networks are popular today in many difficult applications such as time series prediction and speech recognition, to name a few.

A neural network is comprised of many interconnected neurons. Although these neurons are simple computation units by themselves, together they are capable of exhibiting more complicated behavior. Each neuron encapsulates a simple computational operator and

activation function which serves to "squash" the output feed to other neurons. Typically this activation function is a non-linear function such as the popular sigmoid function. The structure of the network dictates how the neurons are connected together and thus how they interact. The most common structure is a feed-forward neural network where neurons are organized into layers. There are also recurrent neural networks, fully-connected neural networks and many other types of structures. In Figure 11, we show the structure of the standard feed-forward neural network with one hidden layer.

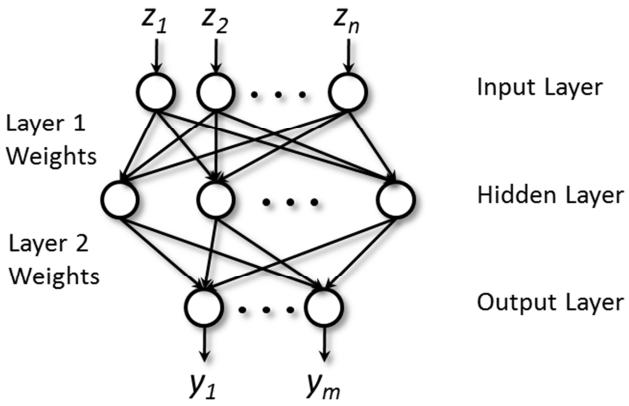


Figure 11. Feed-forward neural network architecture

The most common way to learn the weights of the feed-forward neural network is to use the back-propagation learning algorithm which uses gradient descent to minimize an error criterion (often mean squared error or MSE), cf. [102]. However, gradient descent is plagued by its susceptibility to initial conditions and is often gets caught in local minima. To alleviate this difficulty, evolutionary approaches have been fashioned using genetic algorithms (GA), differential evolution (DE), particle swarm optimization (PSO) and others to determine the optimal weights. Although the feed-forward network is a universal approximator, the network has its disadvantages. One of the most important disadvantages is that there is no meaningful interpretation of the weights. Also, the network is susceptible to over-learning, where the ability of the network to generalize to unseen data diminishes as it becomes over specialized on the training data.

2.2.2. RADIAL BASIS FUNCTION NEURAL NETWORKS

The radial basis function neural network is a modification of the standard feed-forward neural network where radial basis functions (RBF), sometimes called kernel functions, are the activation functions used in the first layer (hidden layer) of the network. The benefit of using an RBF in the hidden layer is that the neuron has a localizing effect on the input; that is, the neuron fires (or outputs high values) when inputs lie within its receptive field. The idea is to specialize parts of the network according to regions in the feature space. This allows the network to learn and generalize more effectively.

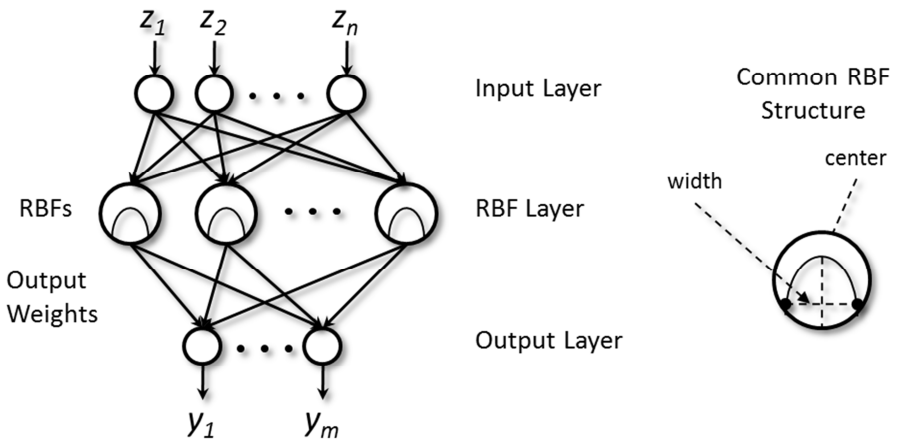


Figure 12. Radial basis function (RBF) neural network architecture

The overall network shown in Figure 12 is expressed by

$$\hat{y}_k = \sum_{i=1}^{n_{RBF}} w_i f_i(\mathbf{z}) \tag{2-54}$$

for k=1...m outputs where $\mathbf{z} = [z_1, z_2, \ldots z_n]$ is the input vector, and n_{RBF} is the number of radial basis functions, i.e. receptive fields. The most common RBF neural activation function is the Gaussian function given by the expression

$$f_i(\mathbf{z}) = e^{-\frac{1}{\sigma_i^2}\|\mathbf{z}-\mathbf{v}_i\|^2} \tag{2-55}$$

where $\mathbf{v}_i$ is the receptive field center (prototype) and σ_i^2 controls the width of each RBF receptive field i=1...n_{RBF}. Another common receptive field function is given by the FCM membership expression

$$f_i(\mathbf{z}) = \frac{1}{\sum_{j=1}^{n_{RBF}} \left(\frac{\|\mathbf{z}-\mathbf{v}_i\|^2}{\|\mathbf{z}-\mathbf{v}_j\|^2} \right)^{1/m-1}} \tag{2-56}$$

where m is the fuzzification coefficient.

Typically the learning is divided into two stages:

1. Determine the RBF centers and widths
2. Learn the weights from the RBF layer to the output layer

The learning of the RBF centers is usually done using unsupervised clustering such as k-means clustering or FCM. The prototypes become the RBF centers. The widths are usually fixed for all receptive fields [15], i.e.

$$\sigma^2 = \frac{\max_{\forall j,k=1\ldots n_{RBF}} \|\mathbf{v}_j - \mathbf{v}_k\|^2}{2n_{RBF}} \tag{2-57}$$

Another method of learning the RBF centers is through conditional FCM, cf. [160]. The main advantage of conditional FCM over traditional approaches is that the learning of the RBF centers includes the output information.

The second stage is accomplished via supervised learning where the weights are either learned via gradient descent or direct numerical optimization. If the activation function of the output neuron(s) is linear, the mean squared error (MSE) criterion can be easily solved directly by numerical means (i.e. matrix inversion).

2.2.3. SUPPORT VECTOR MACHINES (SVMS)

A non-linear supervised learning architecture that is becoming more popular in the literature is the support vector machine (SVM). SVMs are fairly new and have been used in a number of different fields especially pattern recognition problems (classification), cf. [30]. We start by describing the linear SVM and then the kernelized version. The SVM aims to find a decision boundary that divides data $\mathbf{x}_i \in \mathrm{X}$ into two classes labeled as $y_i \in \{-1,+1\}$ for *i*=1...*N* where *N* is the number of data points. The decision boundary is linear and must satisfy

$$y_i\left(\mathbf{x}_i^{\mathrm{T}}\mathbf{w}+b\right)-1\geq 0\ ,\ \forall i=1\ldots N \tag{2-58}$$

where **w** is the weight vector and forms the normal of the hyper-plane, and *b* is the bias. Extensions to SVM have been made to handle multi-classes that are not linearly separable. The objective is to maximize margin, i.e. the distance between the closest positive pattern to the hyper-plane plus the distance between the closest negative pattern to the hyper-plane, cf. cf. [30]. The margin is given by the expression

$$margin=\frac{1}{\|\mathbf{w}\|}+\frac{1}{\|\mathbf{w}\|}=\frac{2}{\|\mathbf{w}\|} \tag{2-59}$$

The SVM determines the hyper-plane that maximizes margin subject to the constraint given in expression (2-58) by minimizing

$$Obj_1=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{N}\alpha_i y_i\left(\mathbf{x}_i^{\mathrm{T}}\mathbf{w}+b\right)+\sum_{i=1}^{N}\alpha_i \tag{2-60}$$

where $\alpha_i \geq 0$ for *i*=1...*N* are the Lagrange multipliers [30]. The data points $\mathbf{x}_i$, where $\alpha_i > 0$, are called support vectors for *i*=1...*N*. As shown in [30], this problem has a dual formulation; that is, minimizing (2-60) is exactly equivalent to maximizing

$$Obj_2=\sum_{i=1}^{N}\alpha_i-\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j \tag{2-61}$$

subject to a different set of constraints

$$\mathbf{w}=\sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \tag{2-62}$$

$$\sum_{i=1}^{N}\alpha_i y_i=0 \tag{2-63}$$

This formulation is more useful as it can be kernelized by replacing the inner product with a kernel function that satisfies Mercer's condition [30]. In other words, the problem becomes the maximization of

$$Obj_3 = \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \quad \textbf{(2-64)}$$

where $K\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is the kernel function. Many different numerical optimization algorithms can be used to maximize (2-64) subject to constraints (2-62) and (2-63), cf. [30]. The advantage of the SVM is that non-linear decision boundaries are possible through the kernelization.

2.3. Evolutionary Optimization

In this section, the evolutionary minimization of an objective function through differential evolution is described.

2.3.1. Differential Evolution

Differential evolution is a simple population-based evolutionary optimization technique that minimizes an objective function much like the popular genetic algorithms, c.f. [190]. Differential evolution uses mutation and crossover operations; however, differential evolution is specialized for objective function Obj($\boldsymbol{\theta}$) whose population vectors are real, i.e. $\boldsymbol{\theta} \in \mathbb{R}^n$ where n is the is the size of population vector. The primary difference between genetic algorithms and differential evolution (DE) is that the mutation operation in DE is based on the difference between two population vectors. We denote the population of vectors by the set $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots \boldsymbol{\theta}_P\}$ where *P* is the size of the population. Differential evolution is expressed via this short pseudo code algorithm:

Differential Evolution
Input: *P*, *G*, p_{cr}
Output: $\boldsymbol{\theta}^{(\text{Best})}$

1) Initialize Population of size *P* randomly (uniform) and within a pre-described hyper-cube in $\mathbb{R}^n$
2) Loop
   a) For *i*=1…*P*
      i) Select three different parameter vectors at random with no preference: $\boldsymbol{\theta}^{(1)}$, $\boldsymbol{\theta}^{(2)}$, $\boldsymbol{\theta}^{(3)}$
      ii) Generate a mutant vector $\textbf{mutant}_i = \boldsymbol{\theta}^{(1)} + G[\boldsymbol{\theta}^{(2)} - \boldsymbol{\theta}^{(3)}]$ *[Mutation]*
      iii) Generate trial vector $\textbf{cross}_i$=crossover($\textbf{mutant}_i$, $\boldsymbol{\theta}_i$) *[Crossover]*
      iv) If Obj($\textbf{cross}_i$) < Obj($\boldsymbol{\theta}_i$) then replace $\boldsymbol{\theta}_i$ with $\textbf{cross}_i$
3) Until maximum number of iterations reached

The real-valued parameter *G* is the differential gain factor and controls the amount of mutation (or exploration of the parameter space). The values of *G* typically range from 0 (no mutation) to 4, c.f. [190]. The crossover operation is very similar to crossover available in genetic algorithms and is performed at random with a probability of p_{cr}. The crossover operation combines two vectors $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ by swapping the elements of the vectors with a probability of p_{cr}. Formally, crossover is accomplished by selecting a random vector **r** of 0's and 1's of length *n*, where the probability of occurrence of 1 is p_{cr}. In this array, 0 indicates "do not crossover" and 1 indicates "crossover." The crossover vector $\textbf{cross}_i$ is computed by

$$\textbf{cross} = \boldsymbol{\theta} \times \mathbf{r} + \textbf{mutant} \times \left(\mathbf{1} - \mathbf{r}\right) \quad \textbf{(2-65)}$$

where $\boldsymbol{\theta} \times \mathbf{r}$ is element-wise vector multiplication.

3. RELATED WORK

Partially supervised learning is introduced in this chapter as a viable method of exploiting domain knowledge. The focus is to outline the benefits and drawbacks of the current state-of-the-art algorithms in the field. A discussion on current clustering frameworks that make use of proximity information is provided followed by a thorough history of partially supervised learning. The literature review starts with the initial papers in this relatively new field and concludes with the most recent developments and trends in partially supervised learning. Following the literature review, a section is devoted to the rather limited yet pertinent developments in selecting the data samples to be labeled called active learning. Finally, in order to adequately cover the background required for the application of partially supervised clustering to time series clustering, prediction and segmentation, several sections are devoted to reviewing their recent advances.

3.1. FORMS OF DOMAIN KNOWLEDGE

Before reviewing partially supervised learning, one must consider the various forms of domain knowledge that can be used to aid clustering. The form of the domain knowledge depends on the problem. We divide the types of available domain knowledge into two parts to form a taxonomy as depicted in Figure 13.

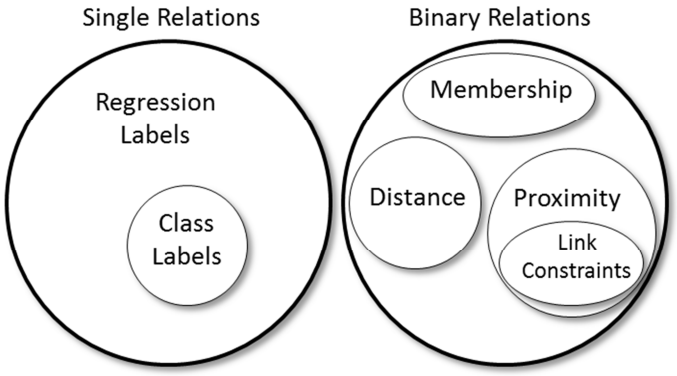


Figure 13. Taxonomy of domain knowledge

The first type of domain knowledge comes in the form of outputs where each labeled element in the data has exactly one output associated with it. The second kind of label is a binary relation where labels are relationships between two elements in the data. For example, the user rating of a song in a music database would be a regression label. Similarly, the degree of similarity between songs, in terms of their genre, would be a proximity label. The output labels are described by

$$\ell(\mathbf{x}) = y \tag{3-1}$$

where ℓ is a mapping between data point **x** and label *y*. The mapping ℓ may represent a process for automatically labeling data by a machine or may represent a human operator that manually assigns labels to the data. For regression labels, ℓ is defined as the mapping where the outputs belong to the space of real numbers, i.e. $\ell : \mathrm{X} \mapsto \mathbb{R}$. Similarly for class labels, ℓ maps data to nominal outputs often represented by integers, i.e. $\ell : \mathrm{X} \mapsto \{1, 2, \ldots K\}$ where *K* is the number of classes in the problem. Many supervised learning algorithms use this output domain knowledge in learning.

Labels that are binary relations are given by

$$\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_2\right) = r_{12} \tag{3-2}$$

where r_{12} is the label associated with the relationship between the data point $\mathbf{x}_1$ and point $\mathbf{x}_2$, and $\mathcal{L}(\bullet,\bullet)$ is the mapping between a pair of data points and the label.

There are several types of binary relations: proximity relations, distance relations, constraint relations and membership relations. When dealing with distance relations, the mapping given in (3-2) is called the distance function and therefore must satisfy the well-known properties of the distance function, i.e.

$$\begin{gathered}
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_2\right) \geq 0 \text{ (non-negative)} \\
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_1\right) = 0 \text{ (identity)} \\
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_2\right) = \mathcal{L}\left(\mathbf{x}_2, \mathbf{x}_1\right) \text{ (symmetry)} \\
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_2\right) \leq \mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_3\right) + \mathcal{L}\left(\mathbf{x}_2, \mathbf{x}_3\right) \text{ (triangular inequality)}
\end{gathered} \tag{3-3}$$

where $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathrm{X}$. The labels produced by the distance function are sometimes called distance labels or distance relationships.

Proximity relations on the other hand are produced by a proximity function and therefore the mapping in (3-2) must satisfy the properties of a proximity function, i.e.

$$\begin{gathered}
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_1\right) \in [0,1] \\
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_1\right) = 1 \text{ (identity)} \\
\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_2\right) = \mathcal{L}\left(\mathbf{x}_2, \mathbf{x}_1\right) \text{ (symmetry)}
\end{gathered} \tag{3-4}$$

where $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathrm{X}$. The corresponding labels are typically referred to as proximity labels. Link constraints are a special class of proximity labels where the proximity function is Boolean, i.e. $\mathcal{L}\left(\mathbf{x}_1, \mathbf{x}_1\right) \in \{0,1\}$. The Boolean 1 denotes the pair of patterns that are strongly associated and must be linked together. The Boolean 0 denotes the pair of patterns that are strongly dissociated and must not be linked together. An example of these constraints is depicted in Figure 14.

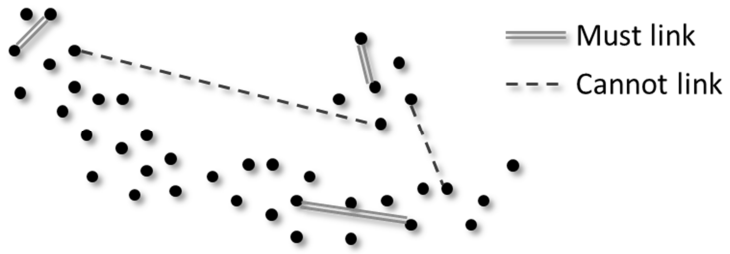


Figure 14. Example of must-link and cannot link constraints

From this example, it can be seen that the purpose of the constraints are to aid in the identification of the clusters. Must-link and cannot link constraints are very common in partially supervised learning, cf. [43][46][13][218][33][80][117][52]. The reason for its popularity is that these constraints are a natural form of domain knowledge for aiding clustering algorithms since they specify that patterns either must or cannot be placed in the same cluster. Unfortunately, this domain knowledge is highly restrictive since the constraints are Boolean.

The final type of domain knowledge described in this section is a membership relation designated by the mapping

$$\mathcal{L}(\mathbf{x}_1, \mathbf{v}_i) = \tilde{u}_{i1} \tag{3-5}$$

where $\tilde{u}_{i1}$ is the membership grade of pattern $\mathbf{x}_1$ to the prototype $\mathbf{v}_i$ from cluster i=1...c where c is the number of clusters in the data. Clearly, membership labels of this form, dictate the number of clusters. This type of data can be transformed into proximity hints eliminating the need to assume the number of clusters "c" equals the number of classes "K," i.e.

$$\tilde{p}_{kj} = \sum_{i=1}^{c} \left(\tilde{u}_{ik} \wedge \tilde{u}_{ij} \right) \tag{3-6}$$

where $\wedge$ is the minimum t-norm operator for j,k=1...N.

3.2. PARTIALLY SUPERVISED LEARNING

Numerous augmentations to a number of common clustering algorithms, including FCM [157][26][161][162][16][179][33][80][117][166], k-means [13][218][52], and agglomerative clustering [43][46][80][73], have appeared in the literature to exploit domain knowledge of various forms in clustering. Several partially supervised algorithms are described in this section, with special attention given to fuzzy and kernel-based clustering.

3.2.1. PARTIALLY SUPERVISED FCM

Membership hints have been exploited in several variations of the FCM algorithm, cf. [157][26][161][162][16][195]. These algorithms accomplish partial supervision by augmenting the standard FCM with a supervised term that penalizes the objective function when a labeled pattern is assigned to the wrong cluster. One variation is introduced in [157][162] where generic FCM is modified with a cost factor that minimizes the difference between the labeled membership and clustering membership. The objective function to be minimized with regards to the membership and prototypes is given by the expression

$$Obj = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^{m} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 + \alpha \sum_{i=1}^{c} \sum_{k=1}^{N} \left(u_{ik} - f_{ik} b_k \right)^m \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 \tag{3-7}$$

subject to the well-known constraint

$$\sum_{i=1}^{c} u_{ik} = 1 \tag{3-8}$$

where the matrix F=[f_{ik}] is the desired membership levels (i.e. hints), and $b_k \in \{0,1\}$ indicates whether the kth pattern is labeled or not, i=1…c, k=1…N. The parameter $\alpha \geq 0$ controls the weight of the partially supervised information in the clustering. The fuzzification coefficient is assumed to be m=2 in order to find expressions for membership and prototypes that minimize (3-7). The expression for updating the membership values is found by solving the partial derivative of (3-7) equal to zero, i.e. $\partial Obj / \partial u_{ik} = 0$, using a Lagrange multiplier to deal with the constraint. The membership is thus computed by

$$u_{ik} = \frac{\alpha}{1+\alpha} f_{ik} b_k + \frac{1 - \frac{\alpha}{1+\alpha} \sum_{j=1}^{c} f_{jk} b_k}{\left(\sum_{j=1}^{c} \frac{\left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2}{\left\| \mathbf{x}_k - \mathbf{v}_j \right\|^2} \right)} \quad \textbf{(3-9)}$$

Similarly, by solving the gradient of (3-7) with respect to prototypes equal to the zero vector, i.e. $\nabla_{\mathbf{v}_i} Obj = \mathbf{0}$, the prototypes are computed using the expression

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} \left(u_{ik}^2 + \alpha \left(u_{ik} - f_{ik} b_k \right)^2 \right) \mathbf{x}_k}{\sum_{k=1}^{N} \left(u_{ik}^2 + \alpha \left(u_{ik} - f_{ik} b_k \right)^2 \right)} \quad \textbf{(3-10)}$$

The algorithm for minimizing (3-7) is as follows.

Partially supervised FCM

Input: F, **b**, X, *c*, α

Output: U, V

1. Initialize membership satisfying constraint (3-8)
2. Do
    a. Compute prototypes V with expression (3-10)
    b. Compute membership U with expression (3-9)
3. Until convergence in the membership U, i.e. $\left\| \mathrm{U}(iter) - \mathrm{U}(iter-1) \right\|_{\mathrm{F}} < \varepsilon$

Note that ε is a small constant that determines the tolerance level in the change of the membership matrix allowed before halting the algorithm. The Euclidean distance in (3-7) can be replaced by the Mahanabolis distance as done in [166] where the fuzzy covariance matrix is calculated based on the labeled data. The Gustafson-Kessel partially supervised FCM was tested on Gustafson's cross and EKG data and performs well compared with standard FCM.

A second partially supervised FCM algorithm is given in [195] where the distance function is modified by adding a positive penalty factor when labeled patterns are assigned to the wrong cluster. The objective function minimized by this partially supervised FCM algorithm is given by

$$Obj = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^m \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 + \alpha \sum_{i=1}^{c} \sum_{k=1}^{N} \left(1 - f_{ik} b_k \right) u_{ik}^m \quad \textbf{(3-11)}$$

The minimization of (3-11) with respect to membership U and prototypes V is subject to the same constraint in (3-8). The parameter $\alpha \geq 0$ controls the amount of repulsion when labeled patterns are assigned incorrectly. An expression for membership is determined by solving the partial derivative of (3-11) with respect to membership equal to zero. The membership is computed with

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 + \alpha \left(1 - f_{ik} b_k \right)}{\left\| \mathbf{x}_k - \mathbf{v}_j \right\|^2 + \alpha \left(1 - f_{jk} b_k \right)} \right)^{\frac{1}{m-1}}} \quad \textbf{(3-12)}$$

The algorithm proceeds in the same manner as FCM by alternating evaluation of the expression for membership (3-12) and the expression for prototypes that is taken from standard FCM.

Partially supervised FCM (#2)

Input: F, **b**, X, *c*, α
Output: U, V

1. Initialize membership satisfying constraint (3-8)
2. Do
    a. Compute prototypes V with the standard FCM expression (2-9)
    b. Compute membership U with expression (3-12)
3. Until convergence in the membership U, i.e. $\left\|\mathrm{U}(iter)-\mathrm{U}(iter-1)\right\|_{\mathrm{F}}<\varepsilon$

Unfortunately, the clustering performance is highly sensitive to the choice of α. The authors report improved performance on a single data set when compared with standard FCM.

Another variation of the FCM algorithm sets the membership equal to the labeled partition for the labeled patterns and otherwise calculates the membership using standard FCM for the unlabeled data, cf. [16]. The objective function minimized is the nearly the same as standard FCM,

$$Obj=\sum_{i=1}^{c}\sum_{k=1}^{N}\left(1-b_k+\alpha f_{ik}b_k\right)u_{ik}^{m}\left\|\mathbf{x}_k-\mathbf{v}_i\right\|^2 \qquad \textbf{(3-13)}$$

subject to the standard FCM constraint (3-8). The partition is therefore obtained using the standard FCM membership expression for the unlabeled patterns (b_k=0). For labeled data (b_k=1), the membership is directly substituted by the known membership, i.e.

$$u_{ik}=\begin{cases}\dfrac{1}{\displaystyle\sum_{j=1}^{c}\left(\dfrac{\left\|\mathbf{x}_k-\mathbf{v}_i\right\|}{\left\|\mathbf{x}_k-\mathbf{v}_j\right\|}\right)^{\frac{2}{m-1}}} & \text{for } b_k=0\\ f_{ik} & \text{for } b_k=1\end{cases} \qquad \textbf{(3-14)}$$

for *i*=1…*c*, *k*=1…*N*. Prototypes are computed identically with standard FCM except where both the labeled and unlabeled data is used to complete them, i.e.

$$\mathbf{v}_i=\frac{\displaystyle\sum_{k=1}^{N}(1-b_k+\alpha b_k f_{ik})u_{ik}^{m}\mathbf{x}_k}{\displaystyle\sum_{k=1}^{N}(1-b_k+\alpha b_k f_{ik})u_{ik}^{m}} \qquad \textbf{(3-15)}$$

In this approach, the labeled data is used to initialize the prototypes, i.e.

$$\mathbf{v}_i=\frac{\displaystyle\sum_{k=1}^{N}f_{ik}^{m}\mathbf{x}_k}{\displaystyle\sum_{k=1}^{N}f_{ik}^{m}} \qquad \textbf{(3-16)}$$

The algorithm is described as follows.

Partially supervised FCM (#3)

Input: F, **b**, X, *c*, α

Output: U, V

1. Initialize prototypes with expression (3-16)
2. Do
    a. Compute prototypes V with expression (3-15)
    b. Compute membership U with expression (3-14)
3. Until convergence in the membership U, i.e. $\left\|\mathrm{U}(iter)-\mathrm{U}(iter-1)\right\|_{\mathrm{F}}<\varepsilon$

The authors test this algorithm on MRI (Magnetic Resonance Images). In [52], the authors apply this same algorithm to an image data set to identify textures in images.

Often some known class information is used to generate the membership hints F. As a result, a serious disadvantage of using the partition matrix for domain knowledge in each of these partially supervised algorithms is that the number of clusters and the number of classes are assumed to be equal. An inherent problem with this is that more complicated class structure, i.e. where there are several clusters in each class, cannot be encapsulated by this clustering paradigm.

Other works in partially supervised fuzzy clustering have been produced that are briefly mentioned. In [161], a unified framework for partially supervised fuzzy clustering is developed for knowledge-based guidance of clustering using proximity hints, labels, or uncertainty (entropy). The author also discusses the suitability of partially supervised clustering for real-world applications such as website clustering, where data is heterogeneous or the feature space often incomplete. Authors in [26] explore different distance measures for partially supervised clustering including Euclidean, Mahanalobis distance, weighted Euclidean, Gustafson-Kessel's distance based on fuzzy covariance, and a kernelized Euclidean distance. They introduce a modification to typical partially supervised FCM by adding a second membership matrix which is minimized according to the provided label information using gradient descent. The difference between membership matrices is minimized by adding a penalty term to the standard FCM objective function, cf. [8]. A partially supervised fuzzy clustering algorithm that uses genetic algorithms (GA) is developed in [135], where FCM is augmented to reduce the variance between the membership matrix and desired membership matrix. They test the algorithm on a data set consisting of web pages from computing science departments at four universities by comparing the results to support vector machines (SVM) and partially supervised FCM.

3.2.2. Kernel Learning: Partially Supervised Learning

The parametric optimization of the parameters for the kernel function has received recent attention, cf. [180][88][148][179][74][183]. Kernel-based clustering, a relatively new clustering paradigm, maps data from the original feature space through a non-linear mapping to a higher dimensional space (referred to as the kernel space in this study to avoid confusion in terminology). Traditionally, the purpose of going to higher dimensions is that the data is more likely to be linearly separable [88][148]. However, the objective of clustering is slightly different than that of classification where the aim in going to higher dimensions is to facilitate the discovery of structure in the data by allowing more flexible cluster geometries through the non-linear mapping. Unfortunately, as discussed by Graves and Pedrycz in [74][77], kernel-based fuzzy clustering requires parameter tuning for optimal clustering performance. Hence a few approaches to learning the kernel parameters using partial supervision have been developed in recent literature.

Optimization of the kernel is a well discussed topic in supervised machine learning algorithms, e.g. Support Vector Machines (SVMs) [42][213][120]. A common technique with SVMs is the minimization of the kernel alignment criterion

$$Align = \frac{\langle K_1, K_2 \rangle_F}{\sqrt{\langle K_1, K_1 \rangle_F \langle K_2, K_2 \rangle_F}} \quad \textbf{(3-17)}$$

where K_1 and K_2 are kernel (Gram) matrices, K_2 is the target kernel (ground truth) and $\langle K_1, K_2 \rangle_F$ is the Frobenius inner product, i.e.

$$\langle K_1, K_2 \rangle_F = trace(K_1 K_2) \quad \textbf{(3-18)}$$

A very important question with optimizing the parameters of the kernel function that produces the Gram matrix K_1 is how to select the target kernel K_2. The choice of K_2 is not obvious. The most common approach with two class problems is to compute K_2 via

$$K_2 = \mathbf{y}\mathbf{y}^T \quad \textbf{(3-19)}$$

where **y** is the vector of all the class labels, i.e. $\mathbf{y}=[y_k]$, $y_k \in \{-1,1\}$ for $k=1\ldots N$. Various implementations of kernel target minimization for the optimization of the kernel parameters have been implemented including semi-definite programming, cf. [120]. They report good performance on the several synthetic and real-world data.

Parametric optimization of the kernel function (in the context of clustering) however have been largely overlooked in the literature [35][202][7][220][226]. Gradient descent has also been used to optimize the kernel parameters, c.f. [35][202], since the parameters are often intractable for direct methods. The authors in [35] develop a unified optimization framework for kernel functions based on pair-wise within-class and between-class scatter matrices. Their approach uses gradient descent to optimize the parameters of the kernel and evaluate their method using a kernel-based k-nearest neighbor classifier. Their results show significant improvements on several of their data sets. Methods of learning the kernel are given in [202] using kernel-based possibilistic fuzzy c-means clustering where the parameters are learned via iterations of gradient descent. Some improvements in prototypes are shown as well as small improvements in classification rate on the Machine Learning wine data set. Authors in [7] perform kernel-based learning in a semi-supervised kernel-based clustering environment where must-link (ML) and cannot-link (CL) constraints are used to optimize the kernel parameters using Semi-Definite Programming (SDP). The disadvantage of SDP is that computational performance scales poorly with large data sets. Their experimental results report the Rand performance index and show their method is better (statistically) than other kernel-learning methods. The width parameter of the Gaussian kernel for a semi-supervised kernel-based k-means algorithm is learned in [220] using gradient descent and exploiting CL constraints. A semi-supervised kernel-based Fuzzy C-Means (FCM) clustering algorithm is developed by minimizing classification error on the labeled data in [226]. The width parameter for the Gaussian kernel function is learned using gradient descent.

The idea of adapting parameters of the distance function to the data is not new. The problem of metric learning has been a popular research topic in clustering that aims to accomplish a very similar task to kernel learning. Metric learning of the Mahanalabolis distance falls under the umbrella of kernel learning since the matrix in the Mahanalabolis distance is positive semi-definite (and thus a kernel). Metric learning with biological data is presented in [33] where the hints are provided in the form of "must-link" and "cannot-link" constraints. The Mahanalabolis distance with a non-zero diagonal is employed (i.e. weighted Euclidean distance). The algorithm minimizes the variance of must-link patterns and maximizes the separation of cannot-link

patterns with simulated annealing (SA). Authors employ gradient descent in [212] for parametric optimization of the Mahanalabolis distance. The problem they solve is identical to the one presented in [33]. Pair-wise constraints are again exploited in partially supervised learning in [80] where the authors extend standard FCM with competitive agglomeration and partial supervision. The competitive agglomeration eliminates the need to pre-select the number of clusters. The authors use the Mahanalabolis distance calculated using the fuzzy covariance matrix and report improved performance on the Iris data set and an image data set.

3.2.3. SUPERVISED CLUSTERING

There are a number of clustering algorithms where all data is labeled with hints, cf. [73][165][188][159][160]. One algorithm based on FCM is called Conditional FCM [159] where auxiliary data (conditional variable) is provided to help guide clustering. Conditional FCM reveals structure by considering the output space (conditional variable). Its aim is to minimize the standard FCM objective function with respect to membership U and prototypes V

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 \qquad \textbf{(3-20)}$$

subject to the constraint

$$\sum_{i=1}^{c} u_{ik} = f_k \qquad \textbf{(3-21)}$$

where $f_k \epsilon [0,1]$ is the amount of involvement of the kth pattern in the clustering for k=1…N. Thus, the constraint forms a conditioning on the clustering. Conditional FCM is applied to the design of RBF neural networks in [160], where fuzzy sets (linguistic contexts) defined in the output space form the conditional variable. A relevant issue of conditional FCM clustering is that the algorithm does not provide any way to generate the fuzzy sets in the output space; hence, in [160] the fuzzy sets are constructed manually by an expert.

Supervised fuzzy clustering for the training of radial basis function (RBF) neural networks is developed in [188] based on FCM clustering where output labels are used to enforce homogeneity in the output space when determining the prototypes of the RBF neurons. The algorithm extends the FCM by adding a penalty term to the distance function for the difference between the actual output and predicted output. The output is predicted based on a sum of several local linear models. The authors report improved performance compared with standard radial basis function neural network training techniques and conditional FCM. Pedrycz developed a supervised clustering algorithm using knowledge hints via a proximity matrix of patterns, c.f. [165]. Graves and Pedrycz supplied knowledge hints in the form of discrete (class) and continuous output labels for supervised agglomerative hierarchical clustering [73]. Supervised hierarchical clustering discovers class structure explicated as the number of clusters per class.

3.3. ACTIVE LEARNING

In partially supervised learning, not all the data is labeled. How to select the data to be labeled is an important research question. Active learning is a field of machine learning that aims to automatically query the labels of particular patterns that would be useful to the problem. Often active learning is used to determine a concise training data set for a supervised learning algorithm while maintaining high accuracy. Active learning, sometimes called query learning, generates a set of queries for advising what data should be labeled either by a human expert or the machine. The motivation is that the process of labeling can be prohibitively expensive for large data sets; thereby active learning aims to curb the cost of labeling. The benefit of active

learning is that it provides a means to justify the training data set used in the learning by allowing the algorithm to decide the patterns to include in training.

A survey of active learning methods is given in [181]. The most common method of active learning is based on uncertainty sampling [201][197][127][181][124]. The active learner uses a probabilistic model to annotate each pattern with a degree of uncertainty. This is often a trivial extension for probabilistic methods of classification. The uncertainty measure is then used to select the data to label based on the principle of least confidence [181]. Under the least confidence strategy, the pattern **x** with the smallest probability under the model is selected for labeling, cf. [201][197][181]. One method seeks to find the pattern with the minimum confidence expressed by

$$Conf\left(\mathbf{x}\right) = 1 - \max_{i=1\ldots K} p(y_i \mid \mathbf{x}) \tag{3-22}$$

where *Conf*(**x**) is the confidence of data point **x** in the data set. Here we use the standard notation to denote the conditional probability, $p(y_i \mid \mathbf{x})$, to be the probability of having the class label y_i for i=1…K (where K is the number of classes) assigned to the pattern **x**. The probabilities are subject to

$$\sum_{i=1}^{K} p(y_i \mid \mathbf{x}) = 1 \tag{3-23}$$

Therefore, when the label of a pattern **x** is uncertain, the conditional probabilities will be evenly distributed, e.g. $p(y_1 \mid \mathbf{x}) = p(y_2 \mid \mathbf{x}) = 0.5$ for a problem with K=2. The least confidence strategy of active learning will favor selecting patterns where all probabilities are distributed equally across all the classes. The principle of least confidence is also employed in [127]. In general, the algorithm for active learning with confidence, provided in [127], starts with a small initial labeled set $\hat{\mathrm{X}}_{init}$, often selected at random from the data with equal preference (uniform random selection). The algorithm is given by the following sequence of steps where L is the target number of labeled patterns.

Confidence-based Active Learning

Input: $\hat{\mathrm{X}}_{init}$, X, L

Output: $\hat{\mathrm{X}}$

1. $\hat{\mathrm{X}} = \hat{\mathrm{X}}_{init}$
2. $M = \left|\hat{\mathrm{X}}\right|$
3. Construct a model with training data set $\hat{\mathrm{X}}$
4. Do
    a. Use the model to determine labels for the unlabeled data set X
    b. Compute the confidence level for each unlabeled element
    c. Label the pattern with the least confidence
    d. Add the pattern with the least confidence to the training data set $\hat{\mathrm{X}}$
    e. M=M+1
    f. Retrain the model with the updated training data set $\hat{\mathrm{X}}$
5. Until M=L

This general algorithm is a commonly used framework for active learning with Support Vector Machines (SVMs) and other classifiers. The strategy of finding the pattern with the least confidence can be easily replaced with other approaches including finding the pattern with the

minimum of some other measure of uncertainty or finding the patterns with the maximum entropy.

A second approach that will be discussed is active learning based on the measure of entropy, i.e.

$$H(\mathbf{x}) = -\sum_{i=1}^{\kappa} p(y_i \mid \mathbf{x}) \log p(y_i \mid \mathbf{x}) \quad \textbf{(3-24)}$$

where $H(\mathbf{x})$ is the entropy of data point **x** in the data set. The idea is to request a label for the pattern with the maximum entropy, i.e. the most unexpected pattern. The maximum entropy active learner follows the same procedure as with confidence-based learning except where step 4(b) is replaced by the computation of the entropy. Steps 4(c) and 4(d) are also replaced with the labeling of the pattern with maximum entropy and adding it to the training data set.

A third type of uncertainty-based active learning is based on the classification margin where a probabilistic decision-making model is not available [216][173]. Another method of active learning queries a pattern based on the expected model change and is highly applicable to optimization problems involving gradient descent. The idea is to label the pattern which will result in the largest magnitude change in the gradient [181]. For these types of active learning, the algorithm must query one or more patterns at a time and repeat the procedure until the desired number of labeled patterns has been reached. However, this is not necessary as many patterns can be selected for labeling simultaneously in a process called batch active learning aimed to reduce redundancy [91].

Another strategy for active learning is called committee-based active learning [145], where multiple models are allowed to determine a structure in the data, e.g. via clustering, on the same data set as depicted Figure 15.

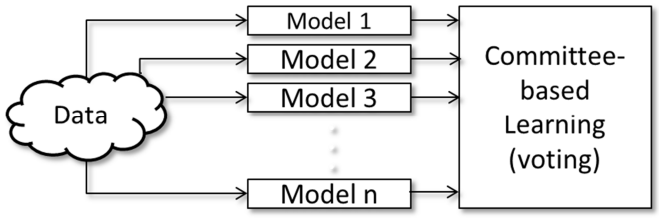


Figure 15. Committee-based active learning

The aim of committee-based active learning is to select the patterns where the structures produced by the models have the most disagreement.

Active learning in clustering is successfully developed and applied to the selection of data from a large data set [89]. Their objective is to select a set of data from a large data set in order to form a partition quickly since many clustering algorithms do not scale well. They select data based on the principle of expected value of information. With this principle, the data that is expected to provide the more information is selected for clustering. They compare their method of active learning to selecting data at random (with uniform probability) using the problem of texture segmentation and report better performance with active learning.

An interesting active learning approach is developed in [79] for selecting the pairs of data must be labeled by must-link (ML) and cannot-link (CL) constraints. They use the active learning methodology to select which constraints are needed in the pair-wise constrained competitive agglomeration (PCCA) semi-supervised clustering algorithm. Their approach selects patterns from the least well-defined cluster, more precisely, the cluster that is not compact and well

separated from its neighbors. The criterion they use is called the fuzzy hyper-volume computed using the determinant of the cluster's fuzzy covariance matrix, cf. [79], i.e.

$$FHV_i = \left|Cov_i\right| \tag{3-25}$$

where $\left|Cov_i\right|$ is the determinant of the fuzzy covariance matrix found by expression (2-15) for *i*=1…*c*, and *c* is the number of clusters. The authors choose to label the pattern belonging to the cluster with the lowest FHV and with membership closest to a threshold value. They report successful results on an image classification data set. A method of active learning based on confidence scores is presented in [201] for the problem of speech classification. Their approach requests labels for the samples for which there is least confidence in their classification. They conclude that active learning with speech classification allows for faster learning on a smaller training data set for similar accuracy as other methods

3.4. PROXIMITY CLUSTERING APPROACHES

Currently there is a growing number of clustering algorithms that cluster with proximity information [165][24][169][164][143]. Pedrycz, et. al. [165] developed a proximity-based fuzzy clustering algorithm that makes use of auxiliary knowledge in the form of proximity hints in fuzzy clustering. The approach uses the membership grades to determine a proximity matrix $\tilde{\mathrm{P}}$ with expression (3-6). The membership values that imply the proximity matrix are optimized via gradient descent to minimize the squared difference from the proximity hints. This optimization is nested within each iteration of FCM clustering. The approach is evaluated by clustering web pages where proximity information is available as additional domain knowledge. The conclusion is that the algorithm exploits the proximity information fairly well.

Another approach to proximity clustering is taken in [29] where objects are mapped to points in a space of specified dimensionality that preserves the proximity information. Fuzzy C-Means (FCM) clustering is then applied. The author evaluates their method on several synthetic and machine learning data using several measures of cluster validity indices.

Authors in [169] formulate a method of clustering proximity data and solve the optimization problem using a procedure which they term "deterministic annealing" which combines elements of a deterministic method with simulated annealing as their optimization vehicle. Their method is compared to Gibbs sampling, where the algorithm is applied to two problems: texture image segmentation and document clustering. They conclude that partitioning cluster algorithms can be applied to problems where only proximity data is available, i.e. where data is not necessarily presented in vector form. A fuzzy proximity clustering algorithm is introduced in [24], where movie critic opinions are clustered together based on a similarity or proximity index. However, limited experimentation is conducted. A clustering algorithm introduced in [143] makes use of belief functions. The authors successfully apply their algorithm to cortex data from the brain of a cat and protein data. A fuzzy clustering algorithm is developed in [164] that uses additional knowledge available from other clustering results. A proximity matrix is determined from the membership grades via expression (3-6) on an initial data set. The proximity information is then used to guide the process of clustering a new related data set.

3.5. TIME SERIES CLUSTERING

Time series clustering is a relatively hot topic given the quantity of time series information available from wireless sensor networks, environmental and weather instruments, seismic instruments, video cameras, audio microphones, and many other data sources [105][94][104]. Thus time series analysis tools for clustering are becoming increasingly important. Clustering is an important time series analysis tool for example in temporal rule discovery [177][87][36], finding temporal patterns [155][194][152][41], and clustering time series [131][153][133][8][176][100][67][50][204][146][129][23][12][130][66][147][214][72][140][187][225][3][97][39][38][9][37][10][199][142][119][82][186][18][57].

A number of temporal clustering algorithms have been developed [153][133][8][176][23][66][214][140][187][225][3][38][10][199][142][119][186][18][99]. Many of these algorithms incorporate different distances to account for time information such as Pearson's correlation coefficient [130], cross-correlation [176][130][66][10], Euclidean distance [130], dynamic time warping [153][31][199] and principle components analysis (PCA) [140][186]. Many clustering algorithms that have been explored for clustering time series including hierarchical clustering [176][38][199], fuzzy c-mean clustering [66][140], self-organizing maps (SOMs) [142][119], k-means clustering [133][8] and expectation-maximization (EM) [57][170].

There are two major approaches to time series clustering:

- Single time series (often streaming or real-time) often used for discovering motifs
- Several time series (often short and of equal length)

Time series clustering can also be broken down into three different approaches:

- Model-based clustering (HMM, ARMA, polynomial, etc)
- Distance-based clustering
- Global feature-based clustering

A survey in [130] shows many different clustering algorithms have been used in the literature including relocation clustering, agglomerative clustering, divisive clustering, k-means clustering, FCM clustering, and self-organizing maps (SOMs). They also report that many different distance measures have been used including Mikowski (generalization of Euclidean) distance, Pearson's correlation coefficient, short-time series distance, DTW distance, Kullback-Liebler distance (based on Markov chain transition probabilities) and its symmetric version, cross-correlation and few others. They also provide a few common cluster evaluation methods. One of mention is the cluster similarity which is a measure based on a provided ground truth.

The most common streaming time series clustering available is to generate several small overlapping subsequences from the original time series and cluster using the second approach as though each subsequence was its own time series as depicted in Figure 16.

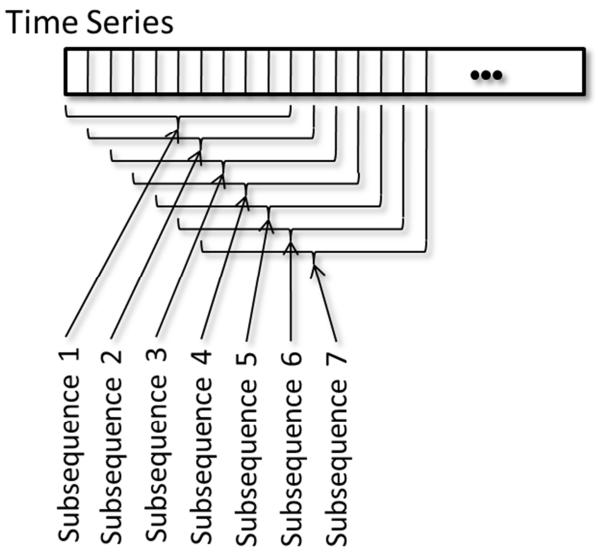


Figure 16: Subsequences of a time series

Unfortunately, as discussed in [131] the temporal dependency of overlapping subsequences produces problems when clustering. If using a partition clustering algorithm such as k-means or Fuzzy C-Means (FCM) then the prototypes will be sinusoidal with little information about the motifs in the time series. Some attempts have been made to correct this problem [67][50][37][82].

The work done in [67] indicates that the prototypes produced by streaming time series clustering, although sinusoidal, are not entirely meaningless by using a different measure of distance between clustering results called cluster shape distance. The authors in [50] apply kernel density estimation clustering to time series. They report that their algorithm is more meaningful than standard k-means based on a meaningfulness measure of internal cluster dispersion divided by between cluster dispersion. Performance of kernel density estimation compared with other methods in time series clustering is not reported. The research conducted in [37] indicates that other distance metrics than Euclidean distance can produce meaningful results. They introduce an interesting way of clustering STS by clustering in the delay space with specified lag factor. The results show promise although very little experimental evidence is given and the accuracy is not reported. The results of the research show however that this type of clustering is meaningful.

Clustering of several time series is a very common approach to time series clustering for example clustering several ECG readings, several stock prices, etc. The idea of clustering several time series is depicted in Figure 17.

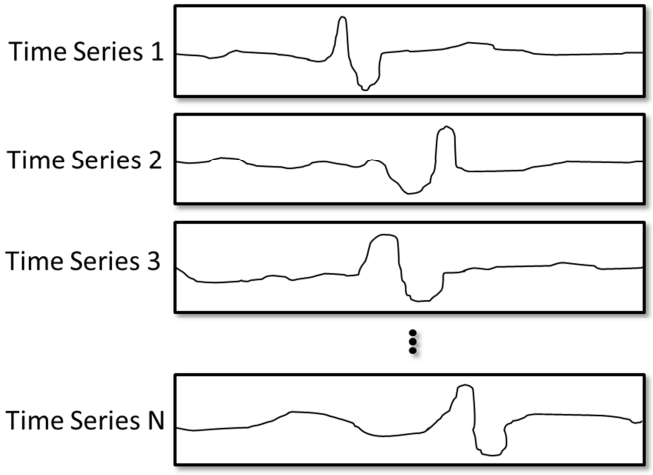


Figure 17: Clustering of many time series

The work done in [176] uses online hierarchical clustering of many streaming time series. The online hierarchical clustering is a divisive-agglomerative algorithm that first divides the set of streams into many streaming clusters and then allows for rejoining to handle non-stationary time series. The distance measure used is based on the Pearson's correlation coefficient. They provide several experimental data sets and show that their real-time is fairly effective compared with a non-real-time k-means clustering approach. Although the performance is slightly improved as measured by the Dunn's validity index and the modified Hubert's statistic; however, the main advantage is real-time clustering of time series. Work done in [204], approaches the time series clustering problem by first discretizing the time series via clustering and then FCM is used with the Kullback–Liebler distance. The results are compared with HMM clustering and show comparable performance via classification rate. They also compare the Kullback-Liebler distance with Euclidean distance and show small improvements in clustering performance. HMM and DTW are common methods of clustering time series [153]. In [153], the authors use both to cluster time series by using DTW hierarchical clustering to form an initial set of clusters followed by HMM clustering to refine the clusters. The experimental evidence in [153] is limited to a single artificial data set. They do not quantify the results of their clustering algorithm with a performance measure. They do however provide the resulting clusters for the synthetic data set. Authors in [12] apply hierarchical clustering with the Hausdorff distance to financial stock data. They conclude that the resulting hierarchical partition provides economically meaningful information in the form of evolution hints for stock prices. Clustering of time series using auto-regressive moving average (ARMA) mixture models is developed in [214]. The authors provide extensive experimental results using a cluster impurity performance measure and a cluster similarity performance measure. The results show comparable performance to other time series clustering methods. They used the Bayesian Information Criterion (BIC) to select the number of clusters. An approach given in [18] aims to find motifs in a set of time series by clustering with a scale-invariant version of FCM. The algorithm relies on the time series being pre-aligned. They experimental show through a few real-world examples, such as heart beat anomaly detection (ECG), that the clustering algorithm capable of detecting clusters of different ECG patterns. They plot the prototypes of each cluster for comparison. In [98], a new distance measure is used where upper and lower boundaries from one time series are defined for each sample in time. If a second time series being compared crosses those boundaries, the distance is the squared difference from the closest boundary. The authors introduce a symmetric distance that is the distance measure is based on the distance between the boundaries of the two time series. They introduce a validity index to measure the performance of their clustering against DTW clustering and Euclidean distance clustering. They also look at the silhouette width of each cluster. They

use only two data sets for experimental evidence. They conclude with little experiment that their method is better.

Thus far, clustering has been discussed in the context of static clustering where the number of clusters does not change with respect to time. However with temporal information, especially streaming or real-time sequences, the number of clusters may in fact change with time, hence dynamic clustering is needed, cf. [156]. The reconstruction error can be used to evaluate the quality of each cluster in FCM clustering and using a threshold cut-off error, clusters can be split as needed. The objective function for FCM is monitored as well and when the objective function falls below a threshold cutoff value, clusters are merged as needed. The result is dynamic clustering [156] where the number of clusters change with time as the structure of the time series evolves. Some experiments are conducted on a synthetic data set and some stock data set where the evolving number of clusters and their prototypes are depicted.

Clustering long time series is very difficult due to high dimensionality of data. Hence several attempts to find ways of representing the key features of time series also called granulation [11] or characterization of time series [84]. In [204], the authors develop a clustering scheme based on global features of a time series. The global features capture the following characteristics of time series: trend, seasonality, serial correlation, non-linearity, skewness, kurtosis, self-similarity, chaos, and periodicity for a total of 13 features. They evaluate clustering with several clustering algorithms including hierarchical clustering, k-means, FCM and self-organizing maps. They evaluate their results on many data sets including the popular UCR data set [105]. The results indicate similar performance to clustering with the entire time series. The main advantage is the dimensionality reduction of the feature.

3.6. TIME SERIES SEGMENTATION

Time series segmentation is a very important topic with temporal information processing as long time series need to be broken down into relevant segments. Another related problem to time series segmentation is time series labeling; once a time series has been segmented the problem of labeling the segments remains. Hence there are two problems in time series segmentation

- Segmentation
- Automatic segment labeling

It is important in any problem to both segment and label the resulting segments in order be complete. Automatic segment labeling too date depends highly upon the context of the problem. The problem of segmentation has not been solved with high accuracy in the literature hence most of research conducted in the literature focuses on the segmentation aspect and leaves the labeling to be done later.

Time series segmentation is accomplished in [60] using genetic algorithms to linearize an ECG time series. The fitness function minimizes the variability between the maximum and minimum slopes for each consecutive sample within the proposed segment. The number of segments is determined beforehand. The experiments are conducted on ECG signals which show a fairly good match on the data, i.e. little information appears to be lost during segmentation (also dimensionality reduction in this case).

There are three very common classical segmentation algorithms in the literature:

- Bottom-up
- Top-down
- Sliding-window

Unfortunately high segmentation accuracy is not always achieved on most data sets and there is no one method superior to the others. Generally speaking, experimental evidence has shown top-down to be the least performer [104]. The three algorithms are described very simply in [104]. All three have an error threshold where they stop segmenting once the error is exceeded. Bottom-up starts with each sample in its own segment, much like agglomerative hierarchical clustering. The difference between bottom-up and agglomerative hierarchical clustering is that segments are merged only if they are adjacent in terms of time such that each segment is contiguous. A merge cost is calculated for each adjacent segment and the segments with the smallest merge cost are joined together. This continues until all merge costs exceed the pre-determined error threshold.

Top-down is similar to divisive hierarchical clustering in that the entire time series is the only segment. Each segment is broken down into smaller segments until the error for each segment is less than the error threshold.

Sliding-window is a real-time segmentation algorithm that starts with the first sample and extends the segment with each new sample until an error threshold is exceeded, at which point the algorithm returns the boundary point and starts the procedure again from where it left off. The error threshold for all three methods is clearly important as it will determine the number of resulting segments.

3.6.1. Music Structural Segmentation

As described by Peeters, c.f. [154], there are approximately two general approaches to the problem of musical segmentation for discovering the structure of music: the state approach and the sequence approach. The author investigates musical segmentation of structural components using Mel Frequency Cepstral Coefficients (MFCC) and compares the sequence approach (chroma) of structural segmentation with the state approach (HMM) and concludes that the state approach is more robust and computationally efficient. The evaluation of the performance of musical segmentation in the literature is done by comparing the resulting segments with an expert labeled song, where a human listener has determined the boundary locations for the individual segments in each song.

Musical segmentation was performed by [122] using MPEG-7 features, c.f. [111], and constrained clustering based on K-Means roughly follows the state approach. Overall, the approach was fairly successful although their experiments demonstrated that there is still a significant amount of room for improvement. A disadvantage is that the number of clusters used in clustering is not determined automatically. Likewise, there is a one-to-one correspondence between a cluster and class (structural type) meaning that a class cannot have more than one cluster assigned too it, which prohibits determining the structure in complex clusters. They use the AudioSpectrumProjection features described in the MPEG-7 standard, which are log-spectra projected onto principle component basis functions, to build a hidden Markov model (HMM) with Gaussian state modeling, c.f. [170]. A single vector-valued Gaussian distribution was used for each state. The number of states was set to M=80 since the authors in [122] show that an HMM has great difficulty in capturing the high-level structural information of music. It is argued by the authors that many states in the HMM can allow the model to capture a sequence of states that corresponds to a structural segment in the music. Their results show that the statement is justified. The authors also employ beat detection using [44] in order to set the frame rate to the beat of the music such that each state in the HMM corresponds to a beat in the music. A local distribution of the states at each frame is clustered using constrained k-means clustering. The must link constraints introduced into k-means clustering are generated automatically by constraining neighboring local state distributions to belong to the same cluster. Cannot link constraints are not used in the research. The research achieve are reported to be better then several other similar clustering techniques on many popular Western-style music databases.

They use two performance indices in the evaluation of their results including the pair-wise f-measure and the boundary retrieval performance (also an f-measure).

Goto, c.f. [71], follows the sequence approach and develops a method called RefraiD that detects the chorus sections of music and can even detect key changes in choruses using a 12-dimensional chroma feature vector. The approach was successful in that 80 of 100 of their test songs were able to retrieve the chorus sections correctly. The chorus detection algorithm finds the similarity between two chroma feature vectors producing a two-dimensional sequence of similarity values with time being the first dimension and lag being the second. Using the similarity measure, the algorithm finds repeated sections in the song. Then using a set of assumptions, the algorithm determines which segments belong to the chorus. The performance is evaluated using the f-measure for boundary retrieval on the chorus.

Authors in [154] propose a method of musical segmentation by detecting boundaries first, followed some aggregation in a two-phase approach. The approach was relatively successful but still has room for significant improvements in performance. The first phase involves detecting possible boundaries using MFCC and sub-band energy features. The boundaries are found by first calculating frame similarity matrices with a cosine similarity index for each feature. The correlation of the similarity matrices is found and the 40 highest local maxima are chosen leading to a set of potential structural boundaries in the song. For phase two, the feature set consists of zero crossing rate, spectral centroid, spectral flatness, spectral roll-off, spectral flux, RMS, low bass energy, and high-medium energy. The mean of these features are calculated for each segment and the between segment similarity is calculated. The potential boundaries from phase one are then combined according to the segment similarity if the similarity is less than some pre-defined threshold. The performance measure is the f-measure for boundary retrieval and they allow 3 seconds (approximately 1 bar of music) in the boundary error before considering a boundary incorrect.

Abdallah et. al., c.f. [1], build a musical segmentation architecture based on a Bayesian framework. Overall, the experimental results for the small database of music used do not indicate how well the architecture performs against existing segmentation methods. The method trains an HMM on a dimensionality-reduced log-frequency log-spectra based on the MPEG-7 AudioSpectrumProjection, as done by authors in [122]. Dimensionality reduction is done by principle component analysis (PCA). The HMM consists of many states such as 40 or 80 states and pair-wise clustering is performed to cluster the local state histograms as in [122]. The difference between what was done in [122] and what was done in [1] is the clustering algorithm employed. In [122], the clustering algorithm is constrained k-means while in [1] the clustering algorithm is based on maximizing a log-likelihood using deterministic annealing. They evaluate on 14 popular songs from Sony's catalogue (audio down-sampled to 11 kHz) and determine that the segment boundary intersection rate is about 80%. The measure the performance via the directional Hamming distance compares the expert boundaries with the segmented boundaries by finding the boundary segments with maximal overlap.

The authors in [123] conduct musical segmentation in order to generate thumbnails for music. They suggest a possible method of selecting the chorus of the song for automatic thumbnail generation. The authors in [61] perform musical segmentation by training an HMM on three different feature spaces: linear predictor coefficients (LPC), LPC-derived cepstral coefficient (LPCC), and MFCC. The results show that LPCC and MFCC are fairly close in performance. After segmentation, identification of the segments is performed where the accuracy is about 90% for both LPCC and MFCC.

Segmentation is also widely used to separate speech from music in audio such as in [68][136][115][2][210]. The problem of separating speech and music in audio is simpler

compared with isolating musical structure since each musical component though different is usually not vastly different; hence, performance of music/speech discrimination is generally good. The authors in [68] outline a method of separating speech in an audio signal with music. Unfortunately, the test results shown are very limited. The results shown for one of the data sets shows that the segmentation finds clusters fairly effectively; however, the results are not quantified nor compared with any other existing methods. The algorithm outlined is a supervised method that involves performing linear discriminant analysis (LDA) on training data. A dynamic program is then applied to identify sequential clusters in the data.

The support vector machine (SVM) has also been used to separate speech from music, c.f. [136]. The results reported are quite good with average accuracies around the mid-90 percent. The technique involves doing some pre-processing by calculating mel-frequency cepstral coefficients (MFCCs) and some perceptual features. The support vector machine with a Gaussian kernel is trained on this combined feature space.

Segmentation of speech and music is accomplished in [2] where a multilayer perceptron (MLP) neural network is used to compute the probabilities of phonemes for each input frame. From these probabilities, entropy and dynamism features are computed and an HMM is trained on the estimated probability density function (PDF). The HMM is a two-state model where one state is non-speech and the other state is speech. Hence non-speech and speech segments are determined. The performance is fairly good with classification rates close to 90%.

The authors in [210] describe an audio segmentation scheme that does not require the number of segments to be specified. In particular, it used the minimum descriptor length (i.e. code word length from coding theory) to determine when to stop the segmentation process. Segmentation is performed by splitting the audio into multiple change points that define boundaries for segments where segments have differing feature means and co-variances. The segmentation process is a top-down hierarchical segmentation process that iteratively breaks segments into sub-segments by finding the largest abrupt change in the means and co-variances in the features. The feature space used is MFCC's. The performance of the method outlined in [210] is decent on the test data sets. The false alarm rate (FAR) and missed detection rate (MDR) are the performance measures used in the research, where the rates are decent but could still allow significant room for improvement.

An approach for speaker segmentation that differentiates different human speakers in an audio signal uses features derived from both wavelets and Fourier transforms, c.f. [198]. The approach uses a combination of a number of different features and report fairly good performance. A support vector machine (SVM) is then used to detect (classify) the speaker in each frame. The performance measure is the classification rate (frame-by-frame) as compared with a ground truth human-labeled signal. The reported accuracies range from 86% on an audio signal with 8 different speakers to up to 100% on an audio signal with only 2 different speakers.

An approach for segmenting audio that contains speech, music and background noise is outlined in [126]. The authors report very good performance on their test data. They use both temporal and spectral features. Frames are classified into speech and non-speech using a k-nearest neighbor (KNN) approach and linear spectral pairs (LSP) that are pre-trained and stored in a speech codebook. The LSPs from the test audio data are compared with those in the codebook and using a threshold value are determined to be speech or non-speech. A rule-based is used to distinguish the different sounds for non-speech. The performance is measured using classification rate and is very good on the speech but is not as good on the environmental sounds and music for their test data.

A review of speaker segmentation is discussed in [115]. A common method used in speaker segmentation is Bayesian information criterion (BIC). The most common feature spaces for

speaker segmentation are MFCC, and MPEG-7 features. Several different segmentation methods are employed in the literature according to the review in [115] including model-based segmentation (e.g. HMM) and metric-based segmentation.

Speaker segmentation is also addressed in [48] where they use the Bayesian Information Criterion (BIC) for segmentation of speech audio containing several different speakers. Their approach is a two-pass approach where they first use distance to select possible candidate boundary points and then the second pass involves using BIC to keep or discard the candidate boundary points. The performance of the algorithm is measured by false alarm rate (FAR) and missed detection rate (MDR). The authors report moderate performance with some significant room for improvement as the false alarm rate is still relatively high in their experiments.

A hidden Markov model (HMM) is used to estimate the key of a song by using 24-states where each state represents the 24 major and minor chords in music in [151]. They report good performance with their HMM key determination scheme. A system is developed in [172] for improving the accuracy of querying a music database using a sung or hummed phrase using relevance feedback in which queries are reformulated and feedback as new queries through the optimization framework of genetic algorithms. The optimization is a function of a measure of relevance of returned music.

4. KERNEL FUNCTIONS AND PROXIMITY FUNCTIONS

Kernel functions are well-known in mathematics and have been used in many different applications including non-linear regression, modeling, and learning. Some of the technical background materials in kernel functions are discussed in this chapter along with a thorough evaluation of kernel-based fuzzy clustering algorithms. It is demonstrated that kernel-based fuzzy clustering promises substantial improvements in the clustering. However, there is the need for parametric optimization of the kernels.

4.1. INTRODUCTION

One of the most popular areas of kernel research is in the field of Support Vector Machine (SVM) – a problem involving the determination of decision boundaries for classification purposes. With SVM, the data is non-linearly mapped to a higher dimensional, possibly an infinitely dimensional space, through the use of a kernel function in the distance where a standard and well-known linear discriminator is learned by maximizing margin. The obvious question arises: why map to a higher dimensional space? The reason is that it can simplify the structure of the data allowing a large category of simple linear machine learning algorithms, including a broad range of classifiers and clusterers, to function as non-linear learning algorithms. With the SVM example, a linear classifier is transformed into a more powerful non-linear one by kernelizing the algorithm.

An obvious question is why kernel functions are not more popular in machine learning. The reasons are as follows:

- Kernelization of algorithms are restricted to ones where the comparison function is expressed solely using inner products
- Very little work has been done in developing a diverse collection of kernel functions
- Little work has been done on optimal selection of the kernel structure and its associated parameters

Many algorithms have been kernelized but very little research has been conducted on providing a suitable framework for determining the best kernel function (structural optimization) and its associated parameters (parametric optimization). It is shown in this chapter that structural optimization and precise parametric optimization of kernels is required to obtain the boasted improvements with kernel clustering methods.

4.2. KERNEL FUNCTIONS: THEORY

As alluded to earlier, a kernel function is also an implicit mapping of data X to a higher dimensional inner product space which we call the *kernel space* $\mathcal{H}$ ($\mathcal{H}$ stands for high dimensional space). In typical kernel literature, the kernel space is also called the feature space [163]. However, we refrain from using this terminology since it is also used in the clustering literature with a different meaning. Instead we choose to apply the term *kernel space* to the higher dimensional space $\mathcal{H}$. We call the lower dimensional inner product space *F* which is called the original feature space. The mapping is given by the expression

$$\Phi(\mathbf{x}) : \mathrm{F} \mapsto \mathcal{H} \tag{4-1}$$

where Φ is the non-linear mapping of data $\mathbf{x} \in \mathrm{X} \subset \mathrm{F}$ to the higher dimensional *kernel space*. Note that F and $\mathcal{H}$ are both inner product spaces and thereby are Euclidean metric spaces (ie. spaces where the Euclidean metric is defined); thus $\mathrm{X} \equiv \mathbb{R}^d$ and $\mathcal{H} \equiv \mathbb{R}^{d_H}$ where *d* is the dimension of the original feature space and d_H is the dimension of the kernel space, cf. [30].

Since the higher dimensional space $\mathcal{H}$ is an inner product space, there exists a kernel function $K(\mathbf{x},\mathbf{y}): \mathrm{F}\times\mathrm{F}\mapsto\mathbb{R}$ such that

$$K(\mathbf{x},\mathbf{y}) = \Phi(\mathbf{x})^{\mathrm{T}}\,\Phi(\mathbf{y}) \tag{4-2}$$

where $\mathbf{x},\mathbf{y}\in\mathrm{X}$ and $\Phi(\mathbf{x})^{\mathrm{T}}\,\Phi(\mathbf{y})$ is the inner product. Thus the kernel function is in fact an inner product in $\mathcal{H}$. Thereby, one can compute inner products in $\mathcal{H}$ implicitly by simply computing $K(\mathbf{x},\mathbf{y})$ (even if $\mathcal{H}$ is infinitely dimensional). The well-known Mercer kernel condition provides the necessary conditions required of a function in order to exhibit the property in expression (4-2). Mercer's condition [30][148][179] states that there exists a mapping $\Phi(\mathbf{x})\in\mathcal{H}$ where expression (4-2) is satisfied if and only if the following condition is true

$$\int K(\mathbf{x},\mathbf{y})\,g(\mathbf{x})\,g(\mathbf{y})\,d\mathbf{x}d\mathbf{y} \geq 0 \tag{4-3}$$

for all the possible functions $g(\mathbf{x}): \mathrm{F}\mapsto\mathbb{R}$ where the L_2-norm of $g(\mathbf{x})$ is finite [30], i.e.

$$L_2 - norm = \int g(\mathbf{x})^2\,d\mathbf{x} \quad \text{is finite} \tag{4-4}$$

Note that in some cases, it may be difficult to assess whether or not a function *K*(**x**,**y**) is a kernel function since condition (4-3) must hold true for every possible function *g*(**x**) that satisfies (4-4). Often kernel functions are instead constructed from known kernel functions using a number of known rules, cf. [30][64].

In addition to kernel functions, there are also kernel matrices which are more commonly referred to as Gram matrices [180]. A kernel matrix represents an inner product in a higher dimensional space $\mathcal{H}$ but for a finite number of data points $\mathbf{x}\in\mathrm{X}\subset\mathrm{F}$. Any symmetric positive semi-definite matrix G of size *N*-by-*N* is also a kernel (Gram) matrix [148], i.e.

$$\mathbf{z}^{\mathrm{T}}\mathrm{G}\mathbf{z} \geq 0\,, \quad \forall\mathbf{z}\neq 0, \mathbf{z}\in\mathbb{R}^N \tag{4-5}$$

Note that this condition is equivalent to saying that all the eigenvalues λ_i of G are non-negative [180], i.e. $\lambda_i \geq 0$, for *i*=1…*N*. A kernel matrix can also be constructed from a kernel function by evaluating the function on a finite number of data points in $\mathbf{x}\in\mathrm{X}$.

4.3. Types of Kernel Functions

The popular Mercer kernel function was described in the previous section. However, there are a number of different types of kernel functions including positive definite kernels and conditionally positive definite kernels.

A very general definition for kernel satisfies the condition of positive definiteness [64], i.e.

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j K(\mathbf{x}_i,\mathbf{x}_j) \geq 0 \tag{4-6}$$

for all $a_i, a_j \in \mathbb{R}$ and all patterns $\mathbf{x}_i, \mathbf{x}_j \in X$ where $K(\mathbf{x}_i,\mathbf{x}_j)$ is a symmetric function. Note that all Mercer kernels satisfy (4-6) and thus are positive definite. However, the converse is not necessarily true. In this way, one may choose a symmetric function $K(\mathbf{x}_i,\mathbf{x}_j)$ that is positive semi-definite (and thus a kernel) for a given data set X. It is a kernel despite the fact that the function does not need to satisfy Mercer's condition (4-3). A broader class of kernels is the conditionally positive semi-definite kernel, cf. [90]. These kernels must satisfy the condition of (4-6) for only those values of a_i and a_j that gratify the constraint

$$\sum_{i=1}^{N} a_i = 0 \qquad \textbf{(4-7)}$$

There are several categories of positive definite kernel functions [64]:

- Stationary
  - Isotropic stationary
    - Compactly supported isotropic stationary
- Non-stationary
  - Locally stationary
  - Separable non-stationary

A kernel is termed stationary if it only depends on the difference between the arguments. A stationary kernel must satisfy the following condition

$$K(\mathbf{x},\mathbf{y}) = K_{Stationary}(\mathbf{x}-\mathbf{y}) \qquad \textbf{(4-8)}$$

for all $\mathbf{x},\mathbf{y} \in \mathrm{X}$. The notation $K(\mathbf{x}-\mathbf{y})$ indicates that the kernel function depends only on **x**-**y** and is therefore stationary. A kernel is isotropic stationary if and only if the kernel is a function of the magnitude of the difference between its arguments, i.e.

$$K(\mathbf{x},\mathbf{y}) = K_{Isotropic}(\|\mathbf{x}-\mathbf{y}\|) \qquad \textbf{(4-9)}$$

for all $\mathbf{x},\mathbf{y} \in \mathrm{X}$ where $\|\mathbf{x}-\mathbf{y}\|$ is the distance between **x** and **y**. Thereby, the Gaussian kernel is an isotropic stationary kernel.

An isotropic stationary kernel function is said to be compactly supported if and only if the kernel function $K_{Isotropic}(\|\mathbf{x}-\mathbf{y}\|) = 0$ for some *threshold* distance, i.e. $\|\mathbf{x}-\mathbf{y}\| \geq threshold$, cf. [64]. Compact kernels are advantageous since a number of efficient algorithms are available that exploit the sparseness of the Gram matrix.

Not all kernels need to be stationary however. Locally stationary kernels satisfy the condition

$$K(\mathbf{x},\mathbf{y}) = K_1\left(\frac{\mathbf{x}+\mathbf{y}}{2}\right) K_{Stationary}(\mathbf{x}-\mathbf{y}) \qquad \textbf{(4-10)}$$

where $K_1((\mathbf{x}+\mathbf{y})/2)$ is a non-negative function, cf. [64]. All stationary kernels are locally stationary kernels (i.e. when K_1 is a positive constant).

Finally, there is a general class of kernels called non-stationary kernels. For example, the polynomial kernel is not stationary. Genton in [64] summarizes the general form of non-

stationary kernels which is beyond the scope of this work. Worth mentioning is a large group of non-stationary kernels called separable non-stationary kernels have the form

$$K(\mathbf{x},\mathbf{y}) = K_1(\mathbf{x})K_2(\mathbf{y}) \tag{4-11}$$

where K_1 and K_2 are stationary kernels that are calculated at **x** and **y** respectively.

A list of a number of the known positive semi-definite kernels are given in Table 5.

Table 5. Positive semi-definite kernel functions

Name	Type	Description
Gaussian	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2\right), \sigma \in \mathbb{R}$
Generalized Gaussian	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = \frac{\tau}{2r\Gamma(1/\tau)}\exp\left(-\frac{1}{r^\tau}\|\mathbf{x}-\mathbf{y}\|^\tau\right)$, $r>0,\tau>0$ where $\Gamma(\bullet)$ is the gamma function
Laplacian	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x}-\mathbf{y}\|\right), \sigma \in \mathbb{R}$
Cauchy	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = \frac{1}{1+\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma}}, \sigma > 0$
Quadratic	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = 1 - \frac{\|\mathbf{x}-\mathbf{y}\|^2}{\|\mathbf{x}-\mathbf{y}\|^2+\theta}, \theta \geq 0$
Multi-quadratic	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = \sqrt{\|\mathbf{x}-\mathbf{y}\|^2+\theta}, \theta \geq 0$
Inverse multi-quadratic	Isotropic Stationary	$K(\mathbf{x},\mathbf{y}) = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{y}\|^2+\theta}}, \theta \geq 0$
Generalized histogram intersection	Non-stationary	$K(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n}\min\left(\|x_i\|^a, \|y_i\|^b\right)$, $a>0,b>0$ where $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$
Polynomial	Non-stationary	$K(\mathbf{x},\mathbf{y}) = \left(\mathbf{x}^{\mathrm{T}}\mathbf{y}+\theta\right)^p$, $\theta \geq 0, p \in \mathbb{N}$
ANOVA [189]	Non-stationary	$K(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{n}\exp\left(-\frac{1}{\sigma^2}\|\mathbf{x}^k-\mathbf{y}^k\|^{2d}\right)$, $\sigma \in \mathbb{R}, d>0, n \in \mathbb{N}$

The function exp(u) denotes the exponential function e^u. Some conditional positive semi-definite kernels are shown in Table 6.

Table 6. Conditionally positive semi-definite kernel functions

Name	Type	Description
Log [90]	Conditionally positive semi-definite	$K(\mathbf{x},\mathbf{y}) = -\log\left(\|\mathbf{x}-\mathbf{y}\|^d+1\right), d>0$
Sigmoid [30]	Conditionally positive semi-definite	$K(\mathbf{x},\mathbf{y}) = \tanh\left(\mathbf{x}^{\mathrm{T}}\mathbf{y}+\theta\right), \theta \in \mathbb{R}$

4.4. PROXIMITY FUNCTIONS

Another useful function closely related to kernels is the proximity function and is a mapping $p(\mathbf{x},\mathbf{y}): X \times X \mapsto [0,1]$ that satisfies the following requirements [165]:

$$\text{Symmetry:} \quad p(\mathbf{x},\mathbf{y}) = p(\mathbf{y},\mathbf{x}) \tag{4-12}$$

$$\text{Identity:} \quad p(\mathbf{x},\mathbf{x}) = 1 \tag{4-13}$$

An example of a proximity function is the commonly encountered cosine-law function. Recall that $\frac{\mathbf{x}^{\mathrm{T}}\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \cos\theta$ where $\cos\theta$ is restricted to the interval [-1,+1]. By scaling the output to the interval [0,1] through a linear transformation, the cosine-based proximity function becomes

$$p(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left(1 + \frac{\mathbf{x}^{\mathrm{T}}\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right) \tag{4-14}$$

where $\|\mathbf{x}\| = \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}}$ is the Euclidean distance. A summary of common proximity functions is given in Table 7.

Table 7. Common Proximity functions

Name	Proximity Function
Gaussian	$p(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2}}$, $\sigma^2 > 0$
Cosine	$p(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left(1 + \frac{\mathbf{x}^{\mathrm{T}}\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}\right)$
Correlation	$p(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left(1 + \frac{E\{\mathbf{x}-\mu_{\mathbf{x}}\}E\{\mathbf{y}-\mu_{\mathbf{y}}\}}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}}\right)$ where $\mu_{\mathbf{x}} = E\{\mathbf{x}\}$, and $\sigma_{\mathbf{x}} = \sqrt{E\{(\mathbf{x}-\mu_{\mathbf{x}})^2\}}$

It is easy to see that some kernel functions also satisfy the properties for proximity functions including the Gaussian kernel function. However, not all kernel functions are proximity functions. Despite this, we can transform any kernel function into a proximity function using the cosine kernel by mapping all data points $\mathbf{x} \in X$ to a higher dimensional kernel space $\Phi(\mathbf{x}) \in \mathcal{H}$.

$$p(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left(1 + \frac{\Phi(\mathbf{x})^{\mathrm{T}}\Phi(\mathbf{y})}{\sqrt{\Phi(\mathbf{x})^{\mathrm{T}}\Phi(\mathbf{x})}\sqrt{\Phi(\mathbf{y})^{\mathrm{T}}\Phi(\mathbf{y})}}\right) \tag{4-15}$$

Since these are inner products in $\mathcal{H}$, they can be replaced with a kernel function. The proximity function then becomes

$$p(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left(1 + \frac{K(\mathbf{x},\mathbf{y})}{\sqrt{K(\mathbf{x},\mathbf{x})}\sqrt{K(\mathbf{y},\mathbf{y})}}\right) \quad \textbf{(4-16)}$$

This measure of proximity is very general as it is a normalized kernel function providing a strong link between proximity and kernel functions. This is exploited in subsequent chapters.

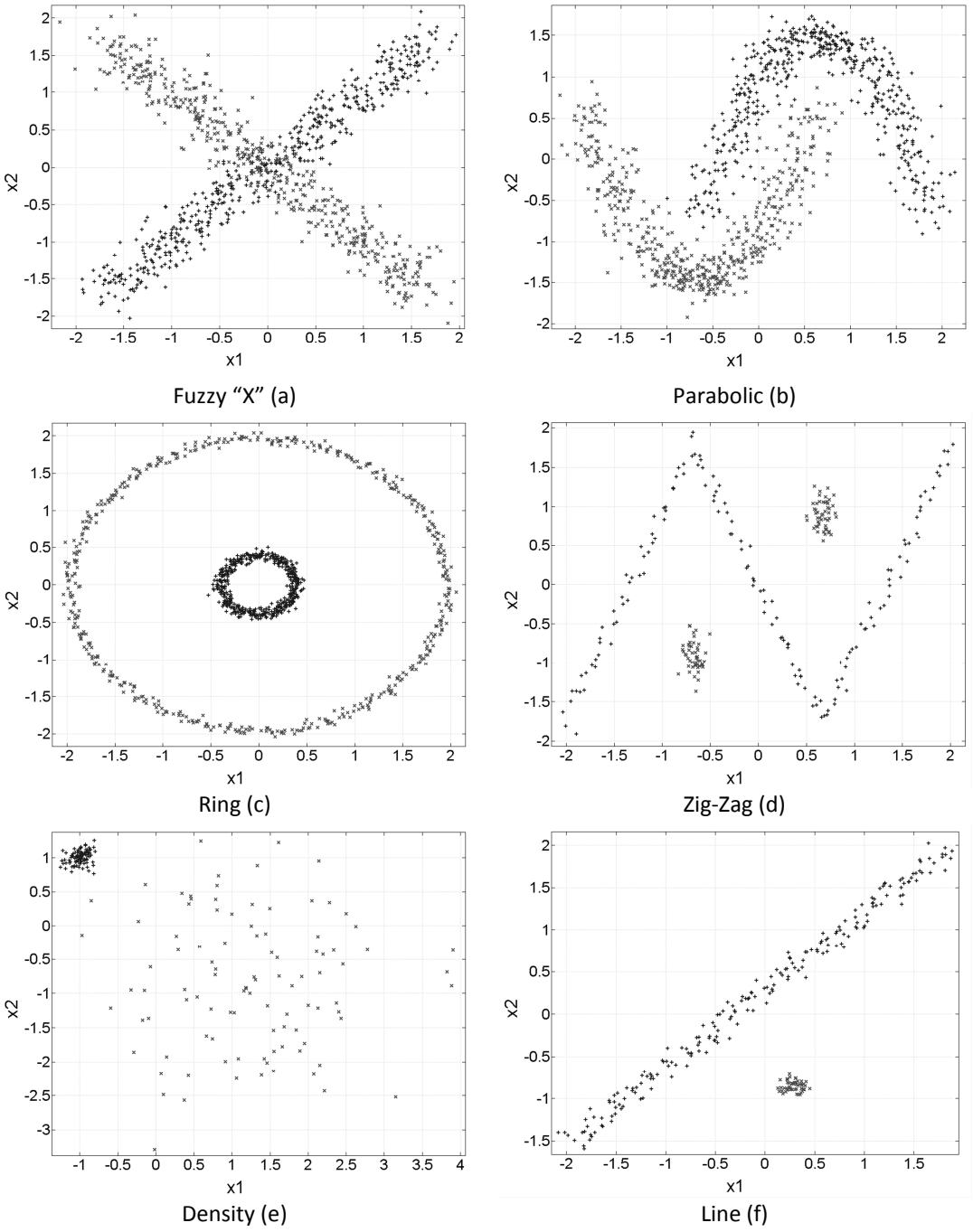


Figure 18. Synthetic data sets

4.5. MOTIVATION: KERNEL CLUSTERING CASE STUDY

In order to evaluate the potential benefits offered by kernel methods and motivate their use in unsupervised learning, two distinct kernelizations of the FCM objective function (KFCM-K and KFCM-F) are compared against standard FCM, see [74], [77], and [76] for further details. The objective is to determine the potential performance gains offered by kernelizing fuzzy clustering and to motivate the need for automated kernel selection. The performance is evaluated in terms of classification rate. First, a suite of synthetic experiments is conducted on the data sets shown in Figure 18.

The clustering algorithms were run over different values for the clustering parameters including the number of clusters c, the fuzzification coefficient m, and the corresponding kernel parameters. The number of clusters c was set to the number of known classes in the data set. The fuzzification coefficient m was varied over the pre-selected set of values {1.2, 1.4, 1.7, 2.0, 2.5, 3.0} for FCM and {1.4, 2.0, 2.5} for the kernel methods. The Gaussian kernel parameter σ^2 was selected manually and varied over the set {0.01, 0.05, 0.1, 0.25, 0.5, 1, 2, 4, 8, 12, 16, 20, 25, 30, 40, 50, 75, 100}. The polynomial kernel parameters (θ, d) were varied over the Cartesian product of the set of θ values {0, 2, 4, 7, 10, 15, 20, 30, 40, 50} and the set of power d values {2, 4, 8, 12, 16}. For each set of parameters the algorithms were repeated 20 times and the means and standard deviations of the classification rate were recorded. The initial membership matrix is generated randomly; hence, the standard deviation of the results quantifies the sensitivity of the algorithms to their initialization. The experimental results are provided in Table 8.

Table 8. Results of Synthetic data sets

Data Set Name	Algorithm	Parameters	Classification Rate (%)	Reconstruction Error
Fuzzy "X"	FCM	c=2, m=2.5	50.8±0.4	1.50±0.00
	KFCM-F (Gaussian)	c=2, m=2.5, σ^2=2	69.5±0.8	1.78±0.02
	KFCM-K (Gaussian)	c=2, m=1.4, σ^2=2	52.0±0.7	1.92±0.07
	KFCM-K (Polynomial)	c=2, m=2.5, θ=2, d=2	75.6±0.7	2.59±0.64
Parabolic	FCM	c=2, m=2.5	87.4±0.0	0.68±0.00
	KFCM-F (Gaussian)	c=2, m=1.4, σ^2=1	88.2±0.0	0.75±0.00
	KFCM-K (Gaussian)	c=2, m=2, σ^2=1	89.0±0.0	0.91±0.00
	KFCM-K (Polynomial)	c=2, m=2.5, θ=10, d=12	87.8±0.0	0.96±0.07
Ring	FCM	c=2, m=1.2	51.5±0.6	1.35±0.00
	KFCM-F (Gaussian)	c=2, m=2, σ^2=0.05	76.3±16.9	2.81±0.75
	KFCM-K (Gaussian)	c=2, m=2.5, σ^2=8	100.0±0.0	2.00±0.00
	KFCM-K (Polynomial)	c=2, m=2, θ=15, d=8	100.0±0.0	2.00±0.00
Zig-Zag	FCM	c=3, m=1.4	66.0±0.0	0.17±0.07
	KFCM-F (Gaussian)	c=3, m=1.4, σ^2=0.01	76.2±12.6	0.20±0.08
	KFCM-K (Gaussian)	c=3, m=1.4, σ^2=0.5	100.0±0.0	2.78±0.77
	KFCM-K (Polynomial)	c=3, m=2.5, θ=0, d=2	80.4±0.0	1.99±0.00
Density	FCM	c=2, m=1.2	93.5±0.0	0.16±0.02
	KFCM-F (Gaussian)	c=2, m=2.0, σ^2=2	94.5±0.0	0.17±0.02
	KFCM-K (Gaussian)	c=2, m=2.0, σ^2=0.25	100.0±0.0	2.93±2.01
	KFCM-K (Polynomial)	c=2, m=2.0, θ=20, d=2	93.0±0.0	1.09±0.32
Line	FCM	c=2, m=2.5	80.0±0.0	0.15±0.00
	KFCM-F (Gaussian)	c=2, m=1.4, σ^2=0.01	82.5±6.4	0.18±0.00
	KFCM-K (Gaussian)	c=2, m=2.5, σ^2=0.25	100.0±0.0	3.00±0.94
	KFCM-K (Polynomial)	c=2, m=2.0, θ=10, d=2	80.0±0.0	0.15±0.00

In a number of cases, the kernel methods provide considerable improvement in the performance in some cases with the right selection of kernel parameters. Although the reconstruction errors

are higher for the kernel methods, the classification rate is in many cases better than standard FCM. The higher reconstruction error may be explained by differences in the placement of the prototypes since the reconstruction error is sensitive to the placement of the prototypes in the data. The boundaries of the clusterings as well as the prototypes are shown in Figure 19 for the fuzzy "X" data set.

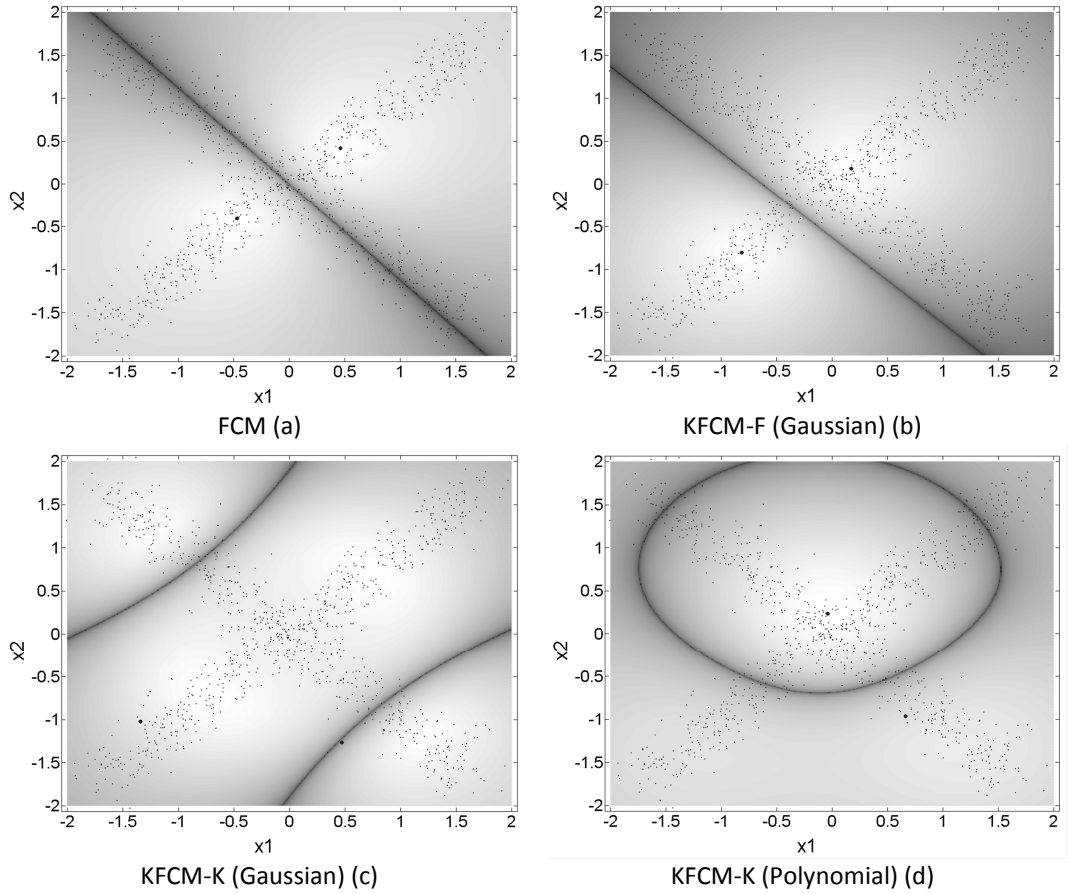
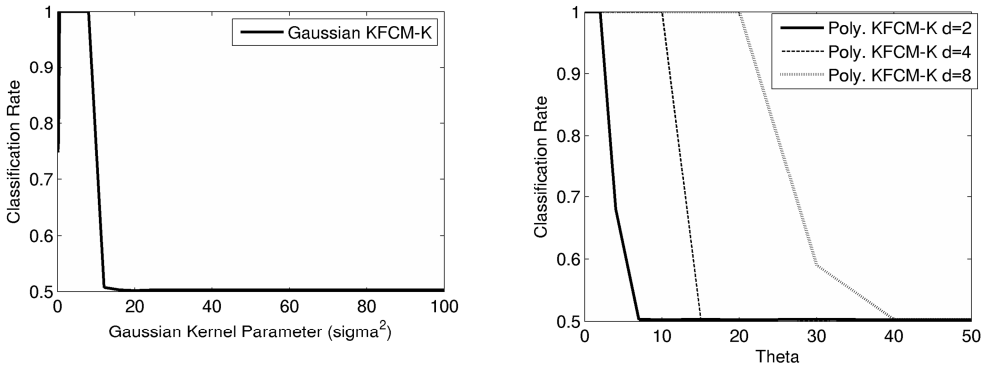


Figure 19. Boundaries produced by FCM, KFCM-F and KFCM-K on the Fuzzy "X" data set

Clearly, kernelized fuzzy clustering offers potentially significant benefits compared to standard FCM. However, an important question is how much does this performance depend on the selection of the kernel parameter. We demonstrate in Figure 20 how the classification rate changes drastically with the kernel parameters on the ring data set.



KFCM-K (Gaussian) (a) KFCM-K (Polynomial) (b)

Figure 20. Classification rate (a) versus σ^2 for Gaussian KFCM-K, and classification rate (b) versus polynomial kernel parameters for polynomial KFCM-K on the ring data set (c=2)

We conclude that an accurate procedure for optimizing the parameters of the kernel parameters is needed on many of these synthetic data sets.

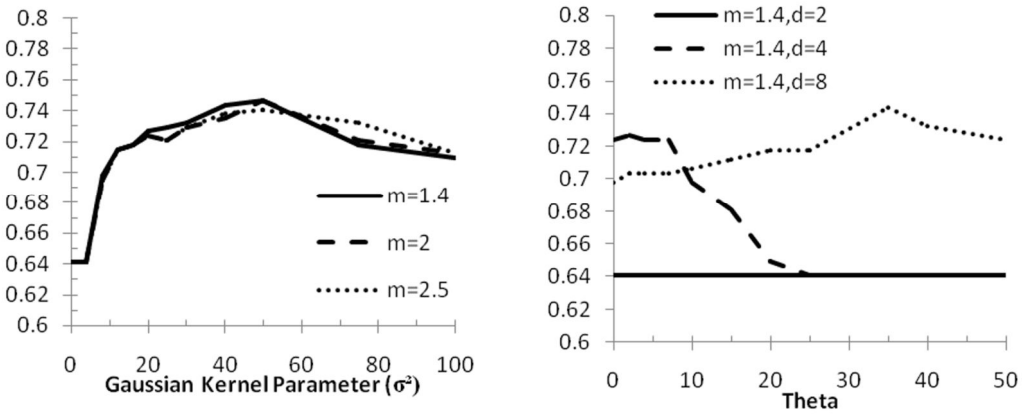
The next suite of experiments involves the performance on a handful of UCI Machine Learning data sets [5] to demonstrate the need for kernel optimization on real-world data. The data sets include iris, wine, ionosphere, image segmentation (testing data set only) and SPECT hear data (training and testing data sets combined).

Table 9. UCI Machine Learning Results

Data Set Name	Algorithm	Parameters	Classification Rate (%)	Reconstruction Error
Iris	FCM	c=3, m=2	84.0±0.0	0.89±0.00
	KFCM-F (Gaussian)	c=3, m=2.5, σ^2=100	84.0±0.0	0.90±0.00
	KFCM-K (Gaussian)	c=3, m=2.5, σ^2=4	85.3±0.0	1.52±0.03
	KFCM-K (Polynomial)	c=3, m=2.5, θ=15, d=8	88.7±0.0	1.94±0.40
Wine	FCM	c=3, m=1.4,	96.6±0.0	6.81±0.00
	KFCM-F (Gaussian)	c=3, m=1.2, σ^2=2	96.1±0.0	8.74±0.00
	KFCM-K (Gaussian)	c=3, m=1.4, σ^2=12	97.8±0.0	8.18±0.00
	KFCM-K (Polynomial)	c=3, m=1.4, θ=20, d=2	97.8±0.0	6.90±0.00
Ionosphere	FCM	c=2, m=1.2	64.1±0.0	32.10±0.02
	KFCM-F (Gaussian)	c=2, m=1.4, σ^2=50	70.7±0.0	26.25±0.00
	KFCM-K (Gaussian)	c=2, m=1.4, σ^2=40	74.6±0.0	26.85±0.00
	KFCM-K (Polynomial)	c=2, m=1.4, θ=30, d=8	72.7±0.0	34.62±1.44
Segment.	FCM	c=7, m=1.7	69.8±1.3	8.22±0.00
	KFCM-F (Gaussian)	c=7, m=2, σ^2=50	71.2±0.0	8.77±0.00
	KFCM-K (Gaussian)	c=7, m=1.4, σ^2=40	66.8±2.8	8.00±0.12
	KFCM-K (Polynomial)	c=7, m=1.4, θ=10, d=2	66.5±1.6	7.88±0.08
SPECT	FCM	c=2, m=1.2	79.4±0.0	17.87±0.00
	KFCM-F (Gaussian)	c=2, m=2.5, σ^2=0.1	80.4±2.0	32.14±5.11
	KFCM-K (Gaussian)	c=2, m=2.5, σ^2=0.05	84.3±0.0	30.67±2.00
	KFCM-K (Polynomial)	c=2, m=1.4, θ=50, d=2	79.4±0.0	18.89±0.00

There is a definite performance increase on the iris, wine, ionosphere, and SPECT data sets. KFCM-F does slightly better than FCM on the segmentation data set; however, KFCM-K does poorly compared to FCM. Looking more closely at the ionosphere data set, which is of relatively

high-dimensionality, there is a statistically significant increase in the classification rate for all the kernel clustering algorithms. Plots of the classification rate and reconstruction error with respect to kernel parameters and the fuzzification coefficient m are given in Figure 21.



KFCM-K (Gaussian) (a)

KFCM-K (Polynomial) (b)

Figure 21. Classification rate (a) versus σ^2 for Gaussian KFCM-K (c=2) and classification rate (b) versus polynomial kernel parameters KFCM-K (c=2) on the ionosphere data set

It is observed that the classification rate varies more than 10% versus the kernel parameters of the Gaussian and polynomial kernels on the ionosphere data set. Thus it is concluded that the parameters of the kernel function in kernelized clustering require an accurate selection procedure in order to obtain optimal performance (optimal in the sense of classification rate). An automated optimization procedure is introduced in the next chapter to address this issue.

5. PARAMETRIC OPTIMIZATION OF KERNELS

The kernelization of fuzzy clustering poses many unanswered questions, particularly the structural and parametric optimization of kernel functions in the presence of some available domain knowledge. As demonstrated in the previous chapter, there is important motivation to exploit hints in order to optimize the parameters of the kernel function. There is still a question of how these hints are attained and, in particular, how the labeled patterns are selected (i.e. via active learning). The specifics of the active learning process are deferred to Chapter 7. For now it is assumed these hints are provided. A promising form of domain knowledge that can be used to optimize the kernel function parameters is a set of proximity relationships. In this chapter a novel approach for optimizing the parameters of a kernel function using proximity hints is introduced.

5.1. OPTIMIZATION FRAMEWORK

The evaluation of kernel-based fuzzy clustering provided in the previous chapter prompts the development of a framework under which the parameters of the kernel function can be optimized. The overall framework is described in Figure 22, where the objective is to acquire the optimal kernel mapping for the data.

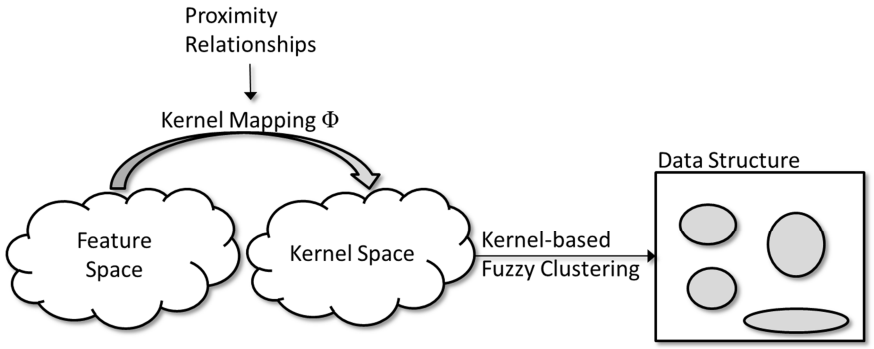


Figure 22. Kernel Function Optimization Framework

Using the semantic link established between kernel functions and proximity functions with expression (4-16), the parameters of the kernel function can be learned through partial supervision where the domain knowledge consists of a number of proximity hints. Given a labeled data set $\hat{\mathrm{X}}$ of size M with dimensionality *d*, where *M<N* where $\hat{\mathrm{X}} \subset \mathrm{X}$, the hints are represented as a matrix of proximity values, $\hat{\mathrm{P}}$ satisfying expressions (4-12) and (4-13). The parameters of the kernel function are then determined by minimizing the objective

$$Obj = \sum_{i=1}^{M} w_i \sum_{j=1}^{M} w_j \left(p\left(\mathbf{x}_i, \mathbf{x}_j\right) - \hat{p}_{ij} \right)^2 \qquad \textbf{(5-1)}$$

where w_i is a positive weight for each sample *i*=1…*N*. This optimization does not depend on the fuzzy clustering and can therefore be accomplished before executing kernel-based fuzzy clustering. Thus, in the proposed approach, the kernel-based fuzzy clustering is performed post-kernel learning.

5.2. DOMAIN KNOWLEDGE: PROXIMITY

An important question is where do the proximity hints $\hat{\mathrm{P}}$ come from? For the moment it is assumed that the data set to be labeled has already been selected (i.e. via active learning).

Obviously, one may obtain proximity hints directly from the available domain knowledge. For example, a human expert might assign a small number of proximity hints to pairs of patterns, see Table 10.

Table 10. Example of proximity hints of songs

	Song 1	Song 2	Song 3	Song 4
Song 1	1.0	0.7	0.3	0.1
Song 2	0.7	1.0	0.2	0.4
Song 3	0.3	0.2	1.0	0.8
Song 4	0.1	0.4	0.8	1.0

These proximity hints can be subjective as in this example. However, it may not be easy to obtain these hints for many kinds of problems. Therefore, it is necessary to determine a method to transform other forms of domain knowledge into proximity hints. Particularly an approach for transforming fuzzy sets in the output space to proximity knowledge is presented. Three types of knowledge are considered:

(a) Membership
(b) Class labels
(c) Fuzzy sets in the output space

For the first type of domain knowledge, the proximity matrix $\hat{\mathrm{P}}$ can be computed from a partial membership matrix $\hat{u}_{ik}$, i=1…c, k=1…M, via expression (3-6). For the second and third type of domain knowledge, the treatment is the same: fuzzy sets $f_i(\mathbf{x}_k)$ are constructed in the output space, i=1…n (n is the number of fuzzy sets) and k=1…M. In the case of class labels, the fuzzy sets are given by the characterization function $f_i(\mathbf{x}_k) \in \{0,1\}$ where

$$f_i(\mathbf{x}_k) = \begin{cases} 1 & if\ \mathbf{x}_k\ belongs\ to\ the\ ith\ class \\ 0 & otherwise \end{cases} \quad \textbf{(5-2)}$$

for i=1…K (K is the number of classes) and k=1…M. In the general case, $f_k(\mathbf{x}_i) \in [0,1]$, i=1…n (n is the number of fuzzy sets) and k=1…M. Using an expression similar to (3-6), the proximity values are computed via the output fuzzy sets

$$\hat{p}_{kj} = \max_{i=1\ldots n}\left(\min\left(f_i(\mathbf{x}_k), f_i(\mathbf{x}_j)\right)\right) \quad \textbf{(5-3)}$$

for k,j=1…M. Expression (5-3) can be generalized by replacing {min, max} with any t-norm and co-t-norm combination.

5.3. METHODOLOGY: EVOLUTIONARY OPTIMIZATION

A second important consideration is how to minimize expression (5-1). Unfortunately, when solving for the parameters of a specific kernel function, such as Gaussian or polynomial, an analytical solution when setting the derivative with respect to the parameters equal to zero is intractable. Therefore, one must employ other optimization vehicles to solve the problem such as differential evolution (DE) [190] or particle swarm optimization [103]. To demonstrate, let us consider a proximity function given by the Gaussian kernel; thus expression (5-1) becomes

$$Obj = \sum_{i=1}^{M} w_i \sum_{j=1}^{M} w_j \left(e^{-\frac{1}{\sigma^2}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} - \hat{p}_{ij} \right)^2 \quad \textbf{(5-4)}$$

where the problem involves finding the value for σ that minimizes *L* all other values constant. One approach is to analytically find a solution by solving $\frac{\partial Obj}{\partial \sigma} = 0$. After taking the partial derivative, the expression obtained is

$$\sum_{i=1}^{M}\sum_{j=1}^{M}\left(e^{-\frac{1}{\sigma^2}\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2} - \hat{p}_{ij} \right) = 0 \qquad \textbf{(5-5)}$$

which further simplifies to

$$\sum_{i=1}^{M}\sum_{j=1}^{M} e^{-\frac{1}{\sigma^2}\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2} = \sum_{i=1}^{M}\sum_{j=1}^{M}\hat{p}_{ij} \qquad \textbf{(5-6)}$$

Unfortunately, there is no known way to separate σ from the left hand side of expression (5-6) since it is part of the exponent; thus, an analytical solution for the Gaussian kernel is intractable. This is true of many kernels including the polynomial kernel. Thus, the use of an optimization vehicle, such as differential evolution or particle swarm optimization, is well motivated. The differential evolution (DE) is used in our approach.

5.4. DIFFERENTIAL EVOLUTION

Differential evolution (DE) is a population-based evolutionary global optimization vehicle for determining the minimum of an objective function with real-valued parameters. Differential evolution is particularly suited for complex objective functions that have many local minimum, cf. [190]. In this case, DE is useful when no known analytical solution exists to determine optimal kernel parameters with expression (5-1).

The idea of DE is to find the best solution in a population of *candidate* solutions. The objective function evaluates the fitness of a *candidate* solution, a.k.a. population vector. DE mutates a population vector by subtracting two vectors from each other. Crossover is performed on the mutated vectors, see section 2.3.1 or [190] for more details.

The representation of the kernel parameters as a real-valued population vector is an important concern. With the Gaussian kernel, the real-valued population vector consists of a single parameter [σ] where σ is the Gaussian kernel width parameter. With the polynomial kernel, the real-valued population vector consists of two parameters [α_1,α_2], where the polynomial shift parameter θ=| α_1| and the polynomial power parameter *p*=round(|α_2|+1). The round function returns the integer nearest to its argument. The reason for requiring the power to be a positive integer is that polynomial functions with powers of a fractional degree are not necessary positive semi-definite, and hence not kernels, cf. [134].

6. PROXIMITY-BASED FUZZY CLUSTERING

The new proximity fuzzy clustering framework is established in this chapter with the goal of clustering data described by proximity relationships. The overall approach of the partially supervised learning framework follows three basic steps:

1. Select a small number of patterns to be labeled with proximity hints (active learning)
2. Determine the optimal kernel mapping using the available proximity hints
3. Perform proximity-based fuzzy clustering

The specifics of step 1 are deferred until Chapter 7. The last chapter presented step 2. In this chapter, the details of clustering with proximity information (step 3) are discussed.

6.1. CLUSTERING FRAMEWORK

Traditionally, methods such as FCM minimize the Euclidean distance between data $X=\{\mathbf{x}_k | k=1\ldots N\}$ and prototypes $V=\{\mathbf{v}_i | i=1\ldots c\}$ where N is the number of data points and c is the number of clusters; however, when clustering a set of objects, we are provided only with a measure of proximity between pairs of objects. In many cases the concept of prototypes may be ill-defined. A more appealing approach is to embed the prototypes in an inner product space (i.e. kernel space) and thus never explicitly computed. The benefits of doing so are that

(a) each cluster's geometry can be non-ellipsoidal, depending on the proximity function, and
(b) the prototypes are not explicitly required in the clustering.

The latter is important for problems where the prototypes are ill-defined, e.g. cluster of graphs. Therefore, we introduce a quantity $q_{ij} = \mathbf{x}_j^{\mathrm{T}}\mathbf{v}_i$ that implies the prototypes within an inner product space, i.e. kernel space. In order to embed the prototypes in a higher dimensional space we replace the inner product with a proximity function, that is also a kernel function, and thus proximity fuzzy clustering aims to minimize the difference between proximity $p\left(\mathbf{x}_k, \mathbf{x}_j\right)$ and $q_{ij} \equiv p(\mathbf{x}_j, \mathbf{v}_i)$ for all $k,j=1\ldots N$, and $i=1\ldots c$. Since the prototypes may be ill-defined, the matrix $Q = \{q_{ij} \mid i = 1\ldots c, j = 1\ldots N\}$ is computed explicitly and is called cluster proximity values. The inputs and outputs of the clustering framework are depicted in Figure 23.

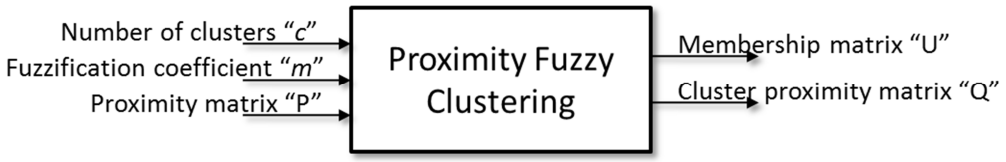


Figure 23. Proximity fuzzy clustering architecture

The generality of the framework allows for the clustering of objects that cannot be described in a $\mathbb{R}^d$ feature space. The outputs are the values to be optimized in the minimization of the proximity fuzzy clustering objective function.

Formally the problem calls for the determination of the optimal membership values u_{ik} and optimal proximity to cluster values q_{ij} for $i=1\ldots c$ and $j,k=1\ldots N$ so that the following performance index is minimized.

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m}\left(\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right)^2\right) \quad \textbf{(6-1)}$$

subject to the constraint

$$\sum_{i=1}^{c} u_{ik} = 1 \quad \textbf{(6-2)}$$

for all k=1…N. The constraint is handled by introducing a Lagrange multiplier λ. An expression for the cluster proximities q_{ij} can be found by solving the equation $\partial Obj/\partial q_{ij} = 0$.

$$\frac{\partial Obj}{\partial q_{ij}} = -2\sum_{k=1}^{N} u_{ik}^{m}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right) = 0 \quad \textbf{(6-3)}$$

$$\sum_{k=1}^{N} u_{ik}^{m} q_{ij} = \sum_{k=1}^{N} u_{ik}^{m} p\left(\mathbf{x}_k,\mathbf{x}_j\right)$$

$$q_{ij} = \frac{\sum_{k=1}^{N} u_{ik}^{m} p\left(\mathbf{x}_k,\mathbf{x}_j\right)}{\sum_{k=1}^{N} u_{ik}^{m}}$$

Similarly, membership u_{ik} is found by solving $\partial Obj/\partial u_{ik} = 0$, i.e.

$$\frac{\partial Obj}{\partial u_{ik}} = m u_{ik}^{m-1}\left(\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right)^2\right) - \lambda = 0 \quad \textbf{(6-4)}$$

$$u_{ik} = \left(\frac{\lambda}{m\left(\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right)^2\right)}\right)^{\frac{1}{m-1}}$$

We use expression (6-2) to determine the value of the Lagrange multiplier.

$$\sum_{h=1}^{c} u_{ik} = \sum_{h=1}^{c}\left(\frac{\lambda}{m\left(\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right)^2\right)}\right)^{\frac{1}{m-1}} \quad \textbf{(6-5)}$$

$$\left(\frac{m}{\lambda}\right)^{\frac{1}{m-1}} = \sum_{h=1}^{c}\left(\frac{1}{\left(\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right)^2\right)}\right)^{\frac{1}{m-1}}$$

Substituting expression (6-5) in expression (6-4) yields the result

$$u_{ik} = \frac{1}{\sum_{h=1}^{c}\left(\frac{\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{ij}\right)^2}{\sum_{j=1}^{N}\left(p\left(\mathbf{x}_k,\mathbf{x}_j\right)-q_{hj}\right)^2}\right)^{\frac{1}{m-1}}} \quad \textbf{(6-6)}$$

The clustering algorithm proceeds by iteratively evaluating expression (6-3) and expression (6-6).

In some problems, one may still require prototypes to be produced by the clustering algorithm. These prototypes can be estimated most-mortem clustering by returning the pattern with the highest proximity to the implied prototypes via the following expression

$$\tilde{\mathbf{v}}_i = \mathbf{x}_{k_i} \; where \, k_i = \arg\max_{\forall j=1\ldots N} q_{ij} \tag{6-7}$$

Note that, unlike FCM, the prototypes are patterns that exist in the data. This method of computing prototypes is beneficial in problems where it is not possible to calculate centroids in the data, e.g. graph clustering problem. The proximity-based fuzzy clustering algorithm is summarized as a sequence of the following steps:

Proximity Fuzzy Clustering Algorithm

Input: X, *c*, *m*

Output: V, U, Q

1. Generate a random initial membership matrix u_{ik} subject to the constraint (6-2)
2. Do
    a. Update q_{ij} from expression (6-3)
    b. Update u_{ik} from expression (6-6)
3. Until convergence in membership values has been observed, i.e. $\left\|\mathrm{U}(iter)-\mathrm{U}(iter-1)\right\|_{\mathrm{F}} < \varepsilon$
4. Estimate prototypes using expression (6-7)

Since, the proximity function can be expressed as a kernel function via expression (4-16), proximity fuzzy clustering is a kernel-based clustering algorithm. We provide an example with the zig-zag data set to motivate proximity fuzzy clustering.

From Figure 24 we can see that the boundaries produced by FCM are very limiting for this problem. The large dots represent the prototypes formed by the clustering. The boundaries produced by proximity fuzzy clustering can be of various shapes, including the circles shown in Figure 24(b).

6.2. Evaluation Criterion

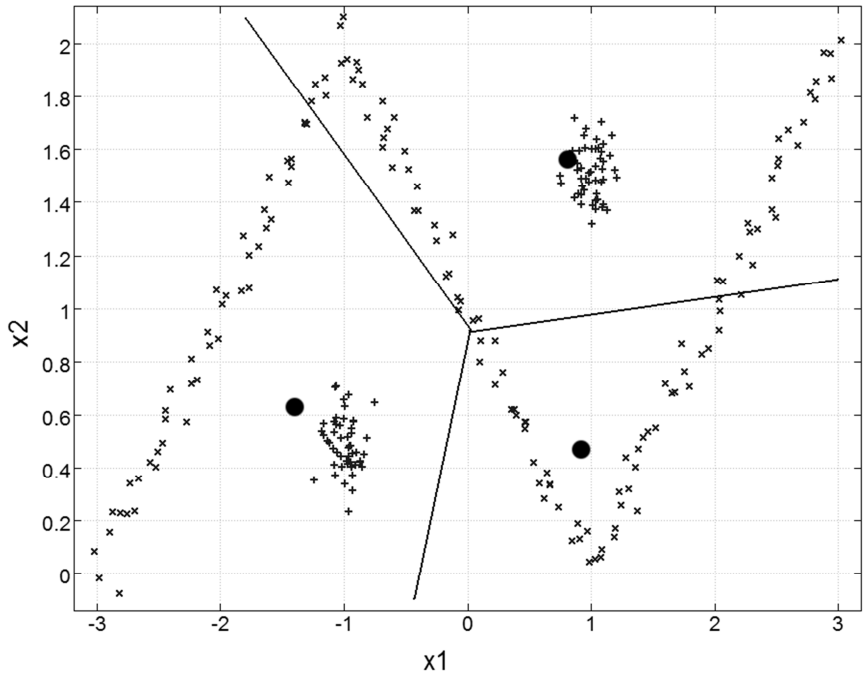
Proximity fuzzy clustering offers a simple and effective (unsupervised) performance index using the membership matrix U produced by the clustering. The index is called the reconstruction error given by

$$re = \sum_{k=1}^{N}\sum_{j=1}^{N}\left(p\left(\mathbf{x}_j,\mathbf{x}_k\right)-\tilde{p}_{jk}\right)^2 \tag{6-8}$$

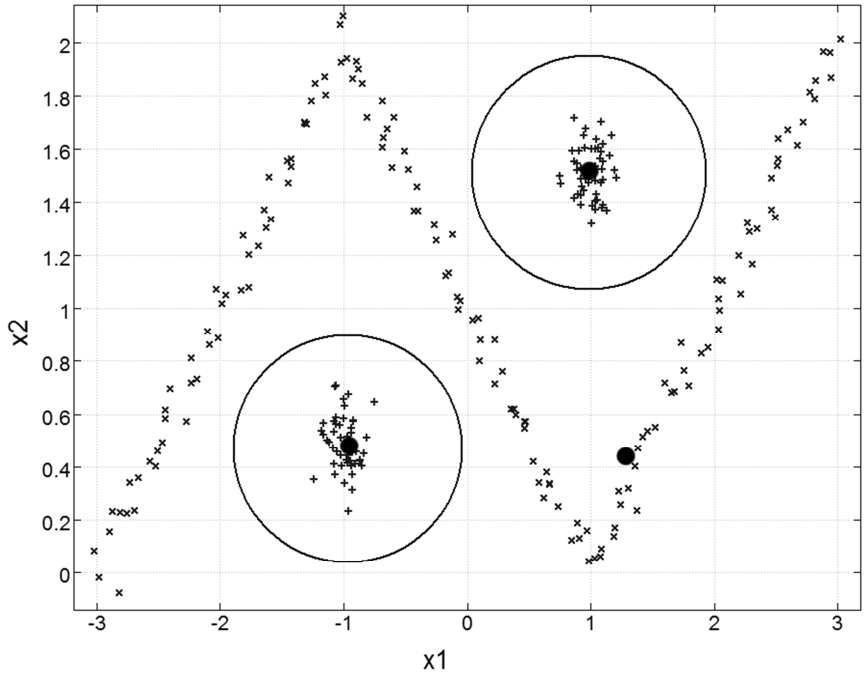
where $\tilde{\mathrm{P}}=[\tilde{p}_{jk}]$ is the proximity matrix reconstructed from the membership matrix U post-mortem clustering, *j*,*k*=1…*N*. The reconstruction error is calculated from the expression

$$\tilde{p}_{jk} = \sum_{i=1}^{c} min\left(u_{ij},u_{ik}\right) \tag{6-9}$$

, *j*,*k*=1…*N*.



(a) Boundaries produced by FCM



(b) Boundaries of Proximity Fuzzy Clustering

Figure 24. Example cluster boundaries

6.3. EXPERIMENTAL RESULTS

The following experiments were conducted by means of manual selection of the kernel parameters in order to determine the performance with respect to the parameters. The parameters evaluated for the Gaussian parameter σ^2 are taken from the set {0.01, 0.05, 0.1, 0.25, 0.5, 1, 2, 4, 7, 10, 25, 50}. The set of parameters for θ is {0, 1, 2, 4, 7, 10, 15, 20, 25, 50} and the

set of parameters for *p* is {1, 2, 3, 4, 5, 6}. The fuzzification parameter *m* is run for the set of parameters {1.2, 1.4, 1.7, 2.0, 2.5, 3.0}. The reason for choosing these sets of parameters is to see the performance across a wide range of the parameters where the performance tends to change the most. For instance, the fuzzification coefficient *m* can take on values greater than 1 but for values larger than 3, the performance was observed to be fairly consistent.

6.3.1. SYNTHETIC DATA

The synthetic data is the same synthetic data shown in Figure 18. The results on the synthetic data are given in Table 11 where "cr" is the classification rate expressed in terms of percentage, and "params" is the set of parameters. The experiments were run for several different values of the number of clusters. Intuitively, increasing the number of clusters improves performance; however, it is not an easy task given the plethora of performance indices available to determine how many clusters one must have for optimal performance on a given data set. We aim to show the performance where the number of clusters equals the number of classes to demonstrate that kernel methods outperform traditional clustering methods such as FCM by allowing non-spherical cluster shapes. The results in Table 11 are recorded using the mean classification rate and its standard deviation for 20 runs.

Table 11. Synthetic data without kernel parameter learning

Data Sets	c	Cosine		Gaussian		Polynomial	
		Params	cr	Params	cr	Params	cr
Parabolic	2	m=2.5	87.5±0	m=3,σ^2=1	88.8±0	m=2,p=6,θ=1	88.4±0
Line	2	m=2.5	80.0±0	m=3,σ^2=0.01	100.0±0	m=1.4,p=2,θ=0	93.2±0
Density	2	m=2.5	97.0±0	m=3,σ^2=1	100.0±0	m=2.5,p=6,θ=1	100.0±0
Ring	2	m=2.5	50.1±0	m=3,σ^2=1	100.0±0	m=2,p=6,θ=0.5	100.0±0
Zig-zag	3	m=1.4	65.8±0.2	m=2.5,σ^2=0.5	100.0±0	m=2.5,p=6,θ=0.5	83.6±0
Fuzzy "X"	2	m=3	51.2±0	m=2,σ^2=25	52.0±0	m=3,p=2,θ=0.5	94.1±0

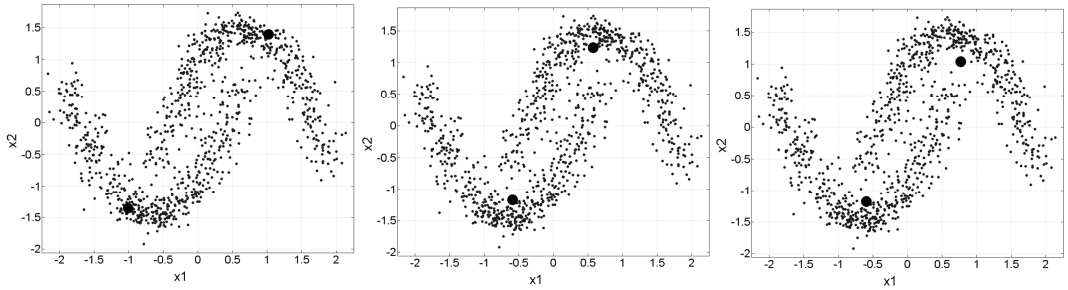
Results for standard FCM clustering and Gustafson-Kessel FCM are given in [77] for the same synthetic data sets. A summary of those results are given in Table 12.

Table 12. Synthetic data with FCM and Gustafson-Kessel FCM

Data Sets	c	FCM		Gustafson-Kessel FCM	
		Params	cr	Params	cr
Parabolic	2	m=2.5	87.4±0	m=3	88.5±0
Line	2	m=3	80.0±0	m=3	100.0±0
Density	2	m=1.2	93.5±0	m=3	90.5±0
Ring	2	m=1.2	51.5±0.6	m=1.2	52.5±1.9
Zig-zag	3	m=1.4	66.0±0	m=3	68.8±0
Fuzzy "X"	2	m=2.5	50.8±0.4	m=3	93.4±0

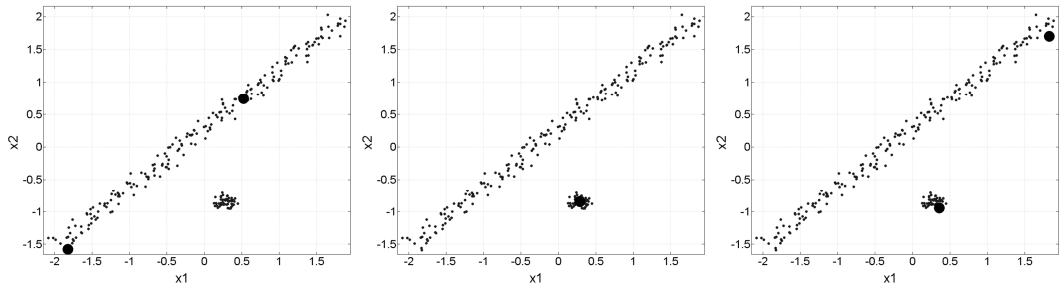
The proximity fuzzy clustering results show a significant improvement over the results from standard fuzzy clustering approaches. Interestingly the Gaussian kernel outperforms the Polynomial kernel on the line data set and the zig-zag data set. However, the Polynomial kernel beats the Gaussian kernel significantly on the fuzzy "X" data set.

The prototypes for the different kernel functions are given in Figure 25.



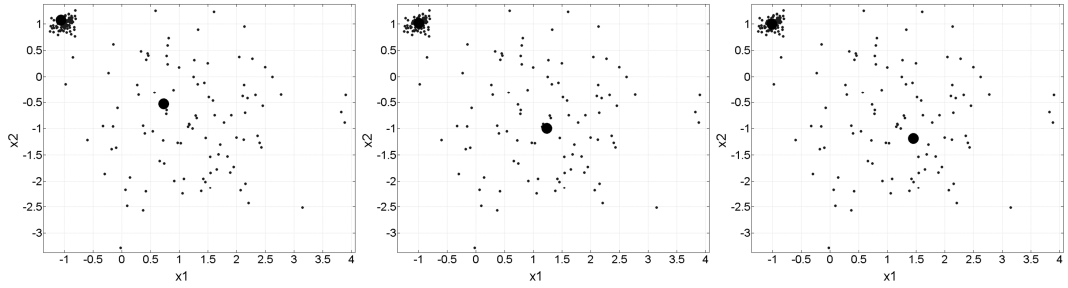
Cosine Gaussian Polynomial

(a) Parabolic



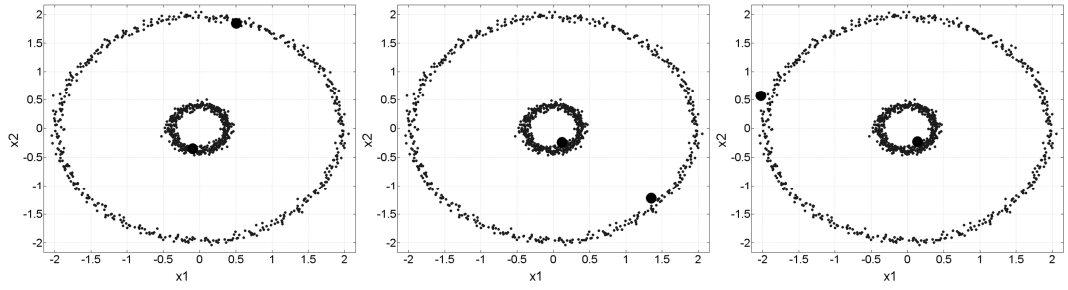
Cosine Gaussian Polynomial

(b) Line



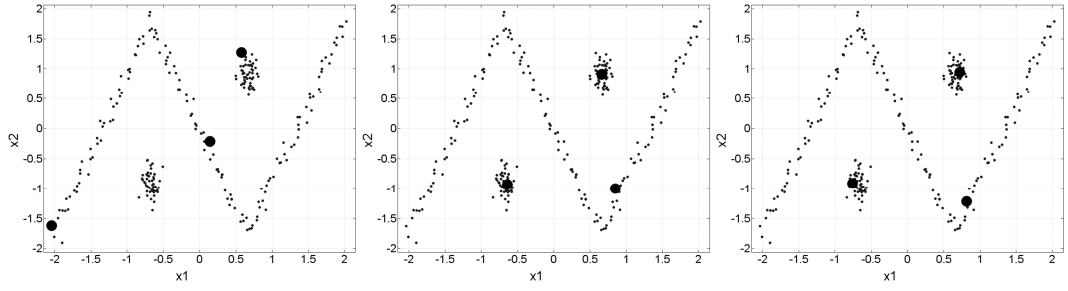
Cosine Gaussian Polynomial

(c) Density



Cosine Gaussian Polynomial

(d) Ring



Cosine Gaussian Polynomial

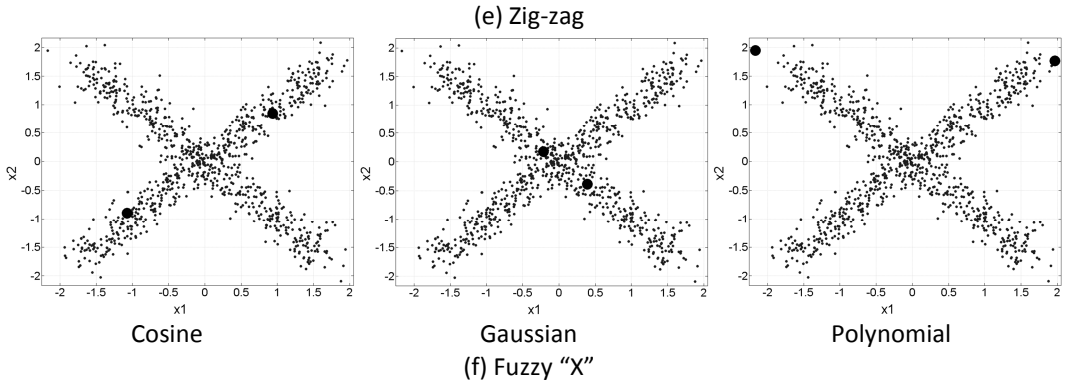


Figure 25. Prototypes of synthetic data sets for cosine, Gaussian and polynomial kernel functions

The locations of the prototypes for the line, zig-zag and fuzzy "X" data sets are very poorly selected when using the cosine kernel. However, when using more versatile kernels, such as the Gaussian or polynomial kernel, the prototypes are better situated in these data sets. A plot of the convergence of proximity fuzzy clustering on the zig-zag data set is provided in Figure 26.

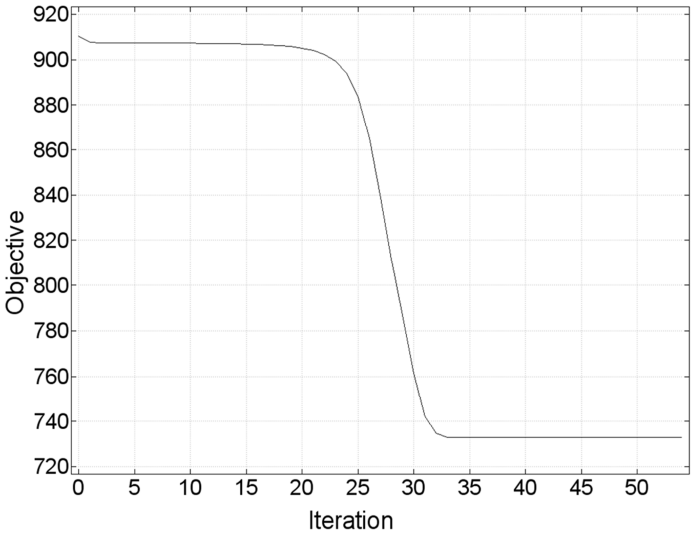
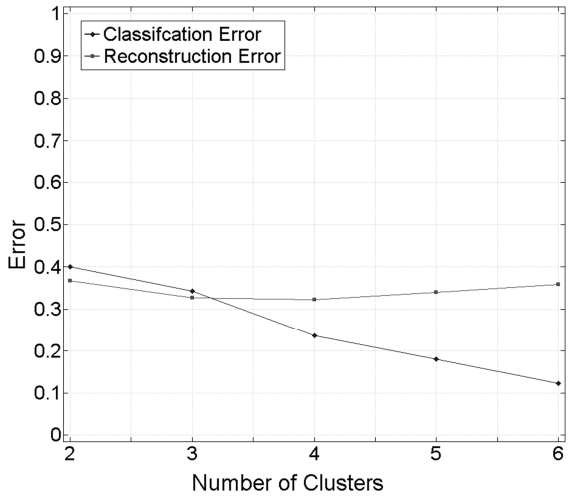
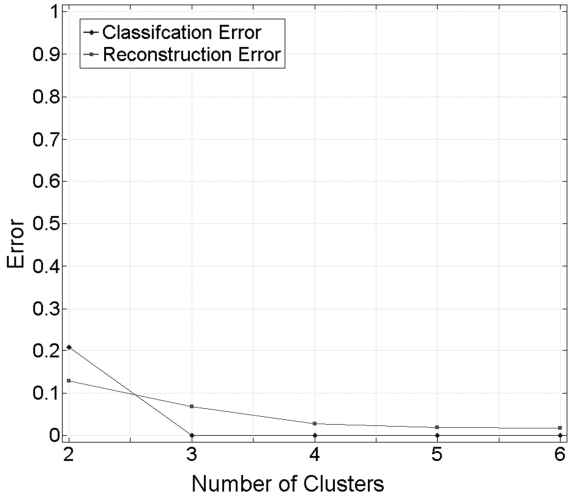


Figure 26. Convergence of proximity fuzzy clustering on the zig-zag data set with the Gaussian kernel (m=2.5,σ^2=0.5)

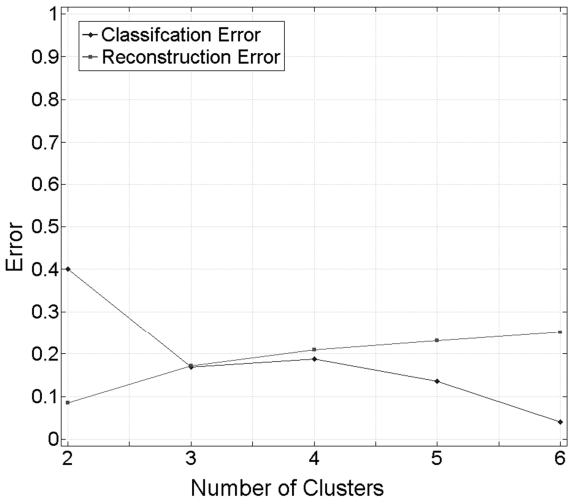
The performance in terms of classification error and reconstruction error are shown with respect to the number of clusters on the zig-zag data set in Figure 27.



(a) Cosine kernel function



(b) Gaussian kernel function



(c) Polynomial kernel function

Figure 27. Classification error and reconstruction error on the zig-zag data set for (a) cosine, (b) Gaussian, and (c) polynomial kernel functions

The anticipated number of clusters is c=3 in the zig-zag data set. The Gaussian kernel does an excellent job of achieving high accuracy (high classification rate) for c=3. However the other kernel functions do not perform as well. As expected, there is a general decreasing trend in classification error as the number of clusters increases.

6.3.2. MACHINE LEARNING DATA

Several Machine Learning data sets [5] were used to evaluate the performance of the clustering.

Table 13. Machine learning data without kernel parameter learning

Data Sets	c	Cosine		Gaussian		Polynomial	
		Params	cr	Params	cr	Params	cr
Breast	2	m=2	97.2±0	m=3,σ^2=10	96.6±0	m=2.5,p=1,θ=0	97.2±0
Ionosphere	2	m=2.5	68.9±0	m=2.5,σ^2=50	73.8±0	m=1.7,p=1,θ=50	75.5±0
Iris	3	m=2	83.3±0	m=2,σ^2=50	90.0±0	m=1.4,p=1,θ=20	90.7±0
Wine	3	m=2.5	93.8±0	m=3,σ^2=10	97.8±0	m=1.2,p=5,θ=20	97.8±0
Diabetes	2	m=1.7	69.7±0	m=1.4,σ^2=10	69.4±0	m=1.2,p=1,θ=10	72.0±0
SPECT-F	2	m=2	79.4±0	m=3,σ^2=25	79.0±0	m=2,p=6,θ=0.5	79.4±0

Experimental results with FCM and Gustafson-Kessel are provided in [77]. A summary is given in Table 14.

Table 14. Machine learning data with FCM and Gustafson-Kessel FCM

Data Sets	c	FCM		Gustafson-Kessel FCM	
		Params	ce	Params	ce
Breast	2	m=1.2	95.7±0	m=2.5	94.7±0
Ionosphere	2	m=1.2	64.1±0	m=1.2	64.1±0
Iris	3	m=2	84.0±0	m=1.7	95.3±0
Wine	3	m=1.4	96.6±0	m=1.4	71.4±5.3
Diabetes	2	m=2	71.3±0.1	m=1.2	65.2±0.3
SPECT-F	2	m=1.2	79.4±0	m=1.2	79.4±0

The classification errors of the kernel-based methods are lower than the classification errors for FCM and Gustafson-Kessel FCM clustering. The exception is Gustafson-Kessel clustering on the Iris data set, where the classification error is considerably lower. Also, the performance is consistently very similar among the different algorithms on the SPECT-F data set.

7. ACTIVE LEARNING

A novel framework for systematically selecting the most informative patterns to be labeled is developed and evaluated in this chapter. The goal is to produce a small yet effective set of labeled patterns since it is often expensive to label data. Recall our approach consists of the three basic steps

1. Select a small number of patterns to be labeled with proximity hints (active learning)
2. Determine the optimal kernel mapping using the available proximity hints
3. Perform proximity-based fuzzy clustering

The first step is the focus of this chapter. Active learning is often overlooked in partially supervised learning and thus this work is an important question to consider in the construction of any partially supervised learning framework.

7.1. INTRODUCTION

Partially supervised fuzzy clustering is still a relatively new paradigm in the realm of machine learning, where the aim is to address the often overlooked availability of domain knowledge that can be exploited to guide and improve the learning. An outstanding research question is how to determine which data should be labeled. A well-studied area of research is active learning. Unfortunately, most active learning approaches focus on answering a slightly different research question – how does one choose a small yet effective training data set? Typical active learning approaches follow the general scheme of determining the training data set of a (fully) supervised learning algorithm. However in this study, we are concerned with a slightly different research problem of selecting the data to be labeled for a partially supervised learning algorithm. The proposed active learning process is illustrated in Figure 28.

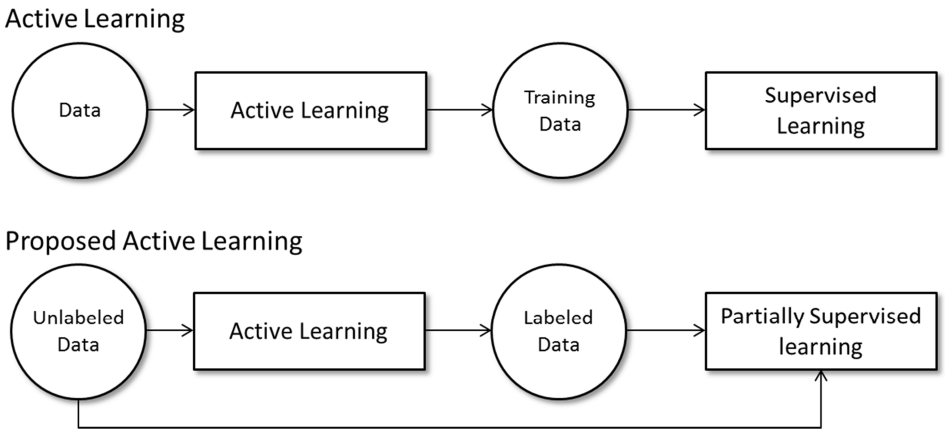


Figure 28. Active learning

The existing active learning literature uses a measure of uncertainty to select patterns to be labeled which is well-founded and motivated in the many statistical models available. The principle of least confident is then employed since the patterns with the most uncertainty will be more informative in the learning. Of particular interest in this study is active learning for partially supervised fuzzy clustering, where the partition is given in terms of grades of membership in the clusters. However, statistical models are not well suited for application to fuzzy clustering. We require an alternative that uses the available degrees of membership from fuzzy clustering to identify the most informative patterns that need to be labeled. The primary objective of our

study is to develop an active learning framework that can be used with partially supervised fuzzy clustering. The general framework of our approach is depicted in Figure 29.

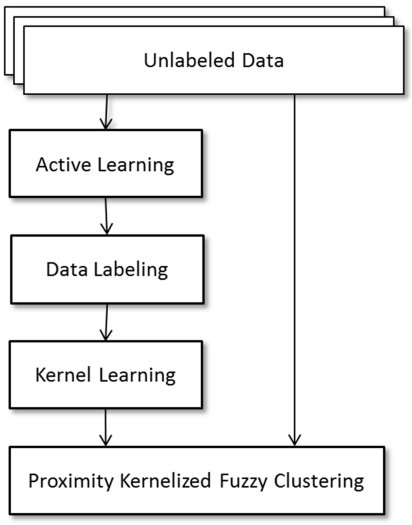


Figure 29. Proposed active learning framework

Although several methods of learning the parameters of the kernel function has been developed, very little work has been done on how to determine which patterns in the data should be labeled in order to conduct parametric optimization. Labeling all the data is generally expensive. Thereby, in this study we propose a new active learning framework in this study where the data to be labeled are automatically selected via active learning. It is the active learning component that is the focus of this study. The novelty in this approach is two-fold:

- we develop an active learning framework that is directly applicable to partially supervised clustering algorithms
- we base our active learning framework on fuzzy logic since the most prevalent clustering algorithms, e.g. FCM, produce partitions with degrees of membership

Our method of active learning will be applied to proximity fuzzy clustering which employs a special class of kernel functions called proximity functions. We choose to evaluate the proposed active learning framework with proximity fuzzy clustering since labeling the data with proximity hints (domain knowledge) is relatively simple to obtain. The proximity values are reflective of domain knowledge expressed in the form of congruency between objects (data) in the unit interval [0,1] where the lower and upper bounds of possible values indicate completely different and identical patterns, respectively. The proximity hints are well suited to form a reference kernel and hence the kernel function can be readily optimized using the approach outlined in the previous chapter.

7.2. FCM-BASED APPROACH

In this section, we introduce our own methodology of active learning that is readily applicable to fuzzy clustering algorithms. The goal is to provide advice to an expert on which data samples should be labeled in order to improve upon the quality of the partially supervised clustering. There could be patterns prepared (identified) by experts but that is a manual approach: we are interested here in some systematic way.

The approach starts with a very small initial set of patterns to be labeled of size *M*, which can be simply one pattern, and adds more patterns to be labeled based on a degree of fuzziness in the

clustering. Subsequently membership values are calculated for the unlabeled data set. The patterns that are very "poorly" explained by the data are selected for labeling and subsequently added to the pool of patterns to be labeled. Hence, the objective is to determine a small set of labeled patterns that best explains the data. The membership values are calculated using the standard FCM expression, that is

$$g_{ik} = \frac{1}{\sum_{j=1}^{M} \left(\frac{\left\| \mathbf{x}_k - \hat{\mathbf{x}}_i \right\|}{\left\| \mathbf{x}_k - \hat{\mathbf{x}}_j \right\|} \right)^{\frac{2}{\tilde{m}-1}}} \qquad \textbf{(7-1)}$$

where $\tilde{m}$ is the fuzzification coefficient, $\hat{\mathbf{x}}_k \in \hat{\mathrm{X}} \subset \mathrm{X}$ is the set of patterns that will be labeled, $\mathbf{x}_k \in \mathrm{D} \subset \mathrm{X}$ is the set of unlabeled data with *L*<*N* elements, and G=[g_{ik}] is the membership matrix for *i*=1…*M* and *k*=1…*L*. The value of *M* will increase from its initial value (the number of initial patterns with labels to be assigned) to a target number of labeled patterns M_{target}. Likewise, the value of L will decrease in exactly the same amount as unlabeled patterns are removed from D and added to Y. The degree of fuzziness of the assigned membership grades is calculated according to the expression

$$f_k = M^M \prod_{i=1}^{M} g_{ik} \qquad \textbf{(7-2)}$$

k=1…*L*. Other measures of fuzziness in membership values can be used as well, cf. [34].

The inputs to the active learning approach are the data set X, and initial data set to be labeled $\hat{\mathrm{X}}_{init}$ and a fuzzification coefficient $\tilde{m}$ that controls the fuzziness of the membership values in G. The output consists of a data set of patterns to be labeled $\hat{\mathrm{X}}$. These labeled patterns are queried for labeling either automatically or manually by a human expert. The approach is summarized as the following sequence of steps:

FCM-based Labeling Algorithm

Input: X, $\hat{\mathrm{X}}_{init}$, M_{target}, $\tilde{m}$

Output: $\hat{\mathrm{X}}$

1. $\mathrm{D} = \mathrm{X} / \hat{\mathrm{X}}_{init}$
2. $\hat{\mathrm{X}} = \hat{\mathrm{X}}_{init}$
3. $M = \left| \hat{\mathrm{X}} \right|$
4. $L = \left| \mathrm{D} \right|$
5. Do
    a. Calculate g_{ik} using expression (7-1) for *k*=1…*L* and *i*=1…*M*
    b. Calculate f_k using expression (7-2) for *k*=1…*L*
    c. Determine pattern **z** with smallest fuzziness f_k
    d. Add **z** to $\hat{\mathrm{X}}$ and remove **z** from D
    e. *M*=*M*+1
    f. *L*=*L*-1
6. Until $M=M_{target}$
7. Return $\hat{\mathrm{X}}$

The returned data set $\hat{\mathrm{X}}$ is a subset of the original data set X and contains all the patterns that need to be labeled. One of the benefits of this approach to active learning is that it is directly

applicable to fuzzy clustering, and particularly partially supervised fuzzy clustering in this research.

An important question remains on how to initialize the labeled data set. One method which we propose in this study is to use FCM clustering to determine a small number of *centroid* points in the data. These *centroid* points are the points closest to the prototypes and form the initial labeled data set.

7.3. EXPERIMENTAL EVALUATION

The performance of the newly developed active learning approaches is evaluated in the context of partially supervised proximity fuzzy clustering, where the proximity labels $\hat{\mathrm{P}}$ are constructed from characteristic functions describing the class labels via expression (5-3). Note that since the proximity labels $\hat{\mathrm{P}}$ and constructed from characteristic functions, the proximity values in these set of experiments are Boolean. The objective is to use this proximity information to aid in the learning of the parameters of the kernel function in proximity fuzzy clustering using objective function (5-1). The weights in the objective function are determined according to class balance by the expression

$$w_i = \frac{M - \sum_{j=1}^{M} \hat{p}_{ij}}{M} \qquad \textbf{(7-3)}$$

where M is the number of labeled patterns, and $\hat{p}_{ij}$ is the proximity value for patterns i,j=1…M. Thus under-represented classes will receive higher weights and over-represented classes will receive smaller weights.

All experiments are repeated 20 times and the mean and standard deviation are recorded. We report the standard deviation to two decimal points. The stopping criterion used is convergence in the membership values where ε=1.0e^{-8}. The number of initial patterns labeled is equal to 2 since this is the minimum number of initial labeled patterns need for the active learning algorithm. The fuzzification coefficient was set to be equal in active learning and proximity fuzzy clustering, i.e. $m = \tilde{m}$.

We use differential evolution (DE) to minimize (5-1) and determine the optimal selection of kernel parameters. The population vectors in DE are real-valued thus the Gaussian kernel has only one search space parameter (kernel width σ) and the polynomial kernel has two parameters (θ and d). The differential gain (G), the size of the population (P) and the crossover rate (p_{cr}) are the most important parameters [190]. The values of DE parameters used for the Gaussian kernel are G_{factor}=0.1, P_{size}=20, and p_{cr}=1. The maximum number of iterations of DE was set to 500 iterations. The DE parameters for the polynomial kernel are G_{factor}=0.5, P_{size}=50, and p_{cr}=0.7. The polynomial kernel requires a larger value for the maximum number of iterations of DE since there are two kernel parameters compared with one for the Gaussian kernel thus the maximum number of iterations was set to 1,500. The values of the DE parameters were selected manually by observing the performance (i.e., classification rate) for the final solution of DE.

7.3.1. SYNTHETIC DATA

This section describes the experimental results for kernel parameter learning on the synthetic data sets with the Gaussian kernel. The classification rate (cr), reconstruction error (re) and normalized mutual information (nmi) were calculated from the partition produced by proximity fuzzy clustering with the learned kernel parameters for each run. Our synthetic data is given in Figure 18.

Table 15. Synthetic data – Gaussian kernel parameter learning (no active learning)

Data Sets	*M*	*Uniform Random*				
		m	σ^2	cr	re	nmi
Line (c=2)	3	1.4	99.2±356.3	89.8±9.5	0.52±0.35	0.55±0.38
	5	1.4	12.6±47.6	83.3±7.0	0.32±0.30	0.30±0.26
	10	1.4	12.6±52.0	82.0±6.2	0.22±0.10	0.26±0.25
	20	1.4	1.05±0.24	80.0±0.0	0.19±0.02	0.18±0.00
	50	1.4	1.04±0.10	80.0±0.0	0.18±0.01	0.18±0.00
Density (c=2)	3	1.4	73.2±312.5	95.8±6.0	0.20±0.10	0.83±0.16
	5	1.2	3.59±1.78	97.4±1.3	0.10±0.04	0.85±0.06
	10	1.2	4.21±1.63	96.8±1.3	0.092±0.026	0.83±0.06
	20	1.2	4.55±1.02	96.3±0.9	0.082±0.007	0.81±0.03
	50	1.2	4.97±0.48	95.8±0.6	0.077±0.002	0.79±0.02
Ring (c=2)	3	2.5	0.971±0.808	88.0±18.5	0.48±0.35	0.73±0.39
	5	2.5	141.1±624.1	100±0	0.22±0.03	1.0±0.0
	10	2.0	1.90±0.31	100±0	0.19±0.01	1.0±0.0
	20	2.0	1.95±0.10	100±0	0.19±0.00	1.0±0.0
	50	2.0	2.03±0.07	100±0	0.18±0.00	1.0±0.0
Zig-zag (c=3)	3	1.2	20.7±89.4	79.2±13.7	0.57±0.37	0.54±0.29
	5	1.2	1.77±4.72	88.8±15.5	0.42±0.30	0.76±0.33
	10	2.5	0.908±0.550	96.8±8.0	0.27±0.05	0.92±0.15
	20	1.4	0.722±0.103	99.9±0.5	0.25±0.03	0.99±0.02
	50	1.2	0.775±0.074	99.9±0.2	0.28±0.01	1.00±0.01

Table 16. Synthetic data – Gaussian kernel parameter learning (active learning)

Data Sets	*M*	*FCM-based Approach*				
		m	σ^2	cr	re	nmi
Line (c=2)	3	2.5	157.9±342.8	88.9±9.6	0.42±0.32	0.36±0.14
	5	1.7	1.26±0.00	80.0±0.0	0.17±0.00	0.18±0.00
	10	1.7	1.36±0.01	80.0±0.0	0.16±0.00	0.18±0.00
	20	1.4	0.883±0.002	80.0±0.0	0.20±0.00	0.18±0.00
	50	1.4	0.823±0.005	80.0±0.0	0.21±0.00	0.18±0.00
Density (c=2)	3	2.0	2.30±0.02	98.0±0.0	0.094±0.001	0.88±0.00
	5	1.4	2.54±0.00	98.0±0.0	0.10±0.00	0.88±0.0
	10	1.4	7.34±0.08	95.5±0.0	0.069±0.000	0.78±0.00
	20	2.5	5.88±0.01	95.5±0.0	0.34±0.00	0.78±0.00
	50	1.4	4.58±0.01	96.5±0.0	0.69±0.00	0.81±0.00
Ring (c=2)	3	1.2	1.90±0.01	100±0	0.20±0.00	1.0±0.0
	5	2.0	2.93±0.02	100±0	0.16±0.00	1.0±0.0
	10	2.0	2.27±0.01	100±0	0.18±0.00	1.0±0.0
	20	2.0	2.16±0.00	100±0	0.18±0.00	1.0±0.0
	50	2.0	2.12±0.00	100±0	0.18±0.00	1.0±0.0
Zig-zag (c=3)	3	1.7	0.00303±0.00216	70.8±4.3	0.99±0.01	0.31±0.13
	5	2.0	0.741±0.001	100±0	0.24±0.00	1.0±0.0
	10	2.0	0.755±0.000	100±0	0.24±0.00	1.0±0.0
	20	1.2	0.803±0.001	100±0	0.28±0.00	1.0±0.0
	50	1.2	0.979±0.007	98.8±0.0	0.23±0.00	0.95±0.00

Interestingly, all methods had difficulty learning the kernel parameter for the line data set; however, the kernel parameter was learned more accurately when using the FCM-based labeling approach for the other synthetic data sets. There is notable improvement for the zig-zag data set in particular. The standard deviations are much lower with FCM-based active learning approach. Figure 30 shows the labeled patterns (represented by large dots) chosen by FCM-based active learning.

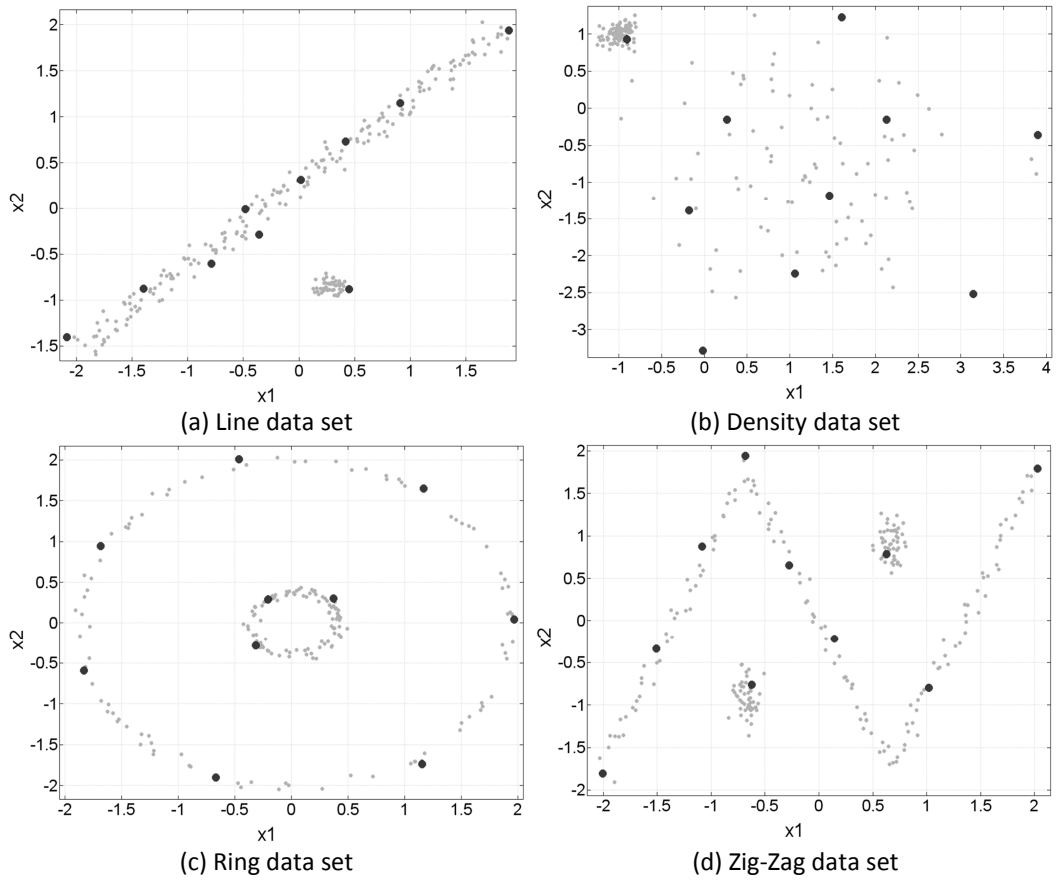


Figure 30. Labeled patterns in synthetic data sets

The parameters used in the production of these plots are *M*=10 and *m*=2. It is observed that the labeled patterns cover the data set very thoroughly as well as cover points at the extremes. Table 17 and Table 18 enclose the results with the polynomial kernel.

Table 17. Synthetic data – polynomial kernel parameter learning (no active learning)

Data Sets	*M*	*Uniform Random*					
		m	θ	d	cr	re	nmi
Line (c=2)	3	3.0	1.1±1.8	4.7±3.7	81.7±3.5	0.028±0.005	0.27±0.13
	5	3.0	0.8±1.5	5.7±3.1	83.9±4.5	0.028±0.007	0.35±0.17
	10	3.0	0.5±1.1	6.4±2.7	86.1±3.7	0.027±0.006	0.44±0.14
	20	3.0	0.0±0.0	6.1±2.0	87.4±2.6	0.024±0.003	0.49±0.09
	50	3.0	0.0±0.0	5.6±1.3	87.4±2.5	0.023±0.001	0.49±0.09
Density (c=2)	3	3.0	0.8±1.8	3.3±3.9	97.6±0.9	0.023±0.021	0.86±0.05
	5	3.0	0.0±0.0	1.7±2.3	97.2±0.6	0.013±0.007	0.85±0.03
	10	3.0	0.0±0.1	2.5±3.0	97.5±0.8	0.015±0.011	0.86±0.04
	20	3.0	0.0±0.1	1.2±0.6	97.1±0.3	0.011±0.002	0.84±0.01
	50	2.5	0.0±0.0	1.2±0.6	97.1±0.3	0.010±0.002	0.84±0.01
Ring (c=2)	3	2.5	1.8±1.9	5.9±4.2	80.7±20.3	0.11±0.06	0.53±0.46
	5	1.2	1.5±1.4	9.1±3.1	84.0±19.6	0.17±0.04	0.62±0.45
	10	2.5	3.2±0.9	11±2.2	100±0	0.057±0.001	1.0±0.0
	20	2.5	3.4±0.3	11±0.3	100±0	0.056±0.000	1.0±0.0
	50	3.0	3.5±0.2	11±0.4	100±0	0.056±0.001	1.0±0.0
Zig-zag (c=3)	3	1.2	0.5±1.1	4.9±3.9	72.7±8.0	0.15±0.05	0.36±0.04
	5	3.0	0.2±0.6	8.8±3.4	81.6±9.6	0.059±0.015	0.59±0.12
	10	3.0	0.1±0.3	10±1.2	87.0±3.7	0.069±0.012	0.65±0.05
	20	3.0	0.1±0.3	10±1.2	87.7±0.9	0.066±0.005	0.66±0.01
	50	3.0	0.0±0.0	11±0.7	88.0±0.3	0.069±0.003	0.67±0.00

Table 18. Synthetic data – polynomial kernel parameter learning (active learning)

Data Sets	*M*	*FCM-based Approach*					
		m	θ	d	cr	re	nmi
Line (c=2)	3	2.0	3.7±1.8	6.2±3.0	80.4±1.6	0.068±0.029	0.21±0.07
	5	3.0	0.1±0.0	4.0±0.0	89.2±0.0	0.024±0.000	0.54±0.00
	10	3.0	0.1±0.0	4.0±0.0	88.8±0.0	0.023±0.000	0.53±0.00
	20	3.0	0.0±0.0	6.0±0.0	88.4±0.0	0.023±0.000	0.52±0.00
	50	2.0	0.1±0.0	5.9±0.4	88.0±0.1	0.053±0.000	0.51±0.00
Density (c=2)	3	2.0	0.0±0.0	1.2±0.6	97.1±0.3	0.015±0.003	0.84±0.01
	5	1.2	0.0±0.0	1.3±0.7	97.2±0.4	0.036±0.010	0.84±0.02
	10	2.5	0.0±0.0	1.1±0.8	97.2±0.4	0.011±0.002	0.84±0.02
	20	1.7	0.0±0.0	3.0±0.0	98.0±0.0	0.034±0.000	0.88±0.00
	50	2.5	0.1±0.2	1.2±0.6	97.1±0.3	0.010±0.002	0.84±0.01
Ring (c=2)	3	1.7	2.8±1.8	6.9±3.4	96.1±10.2	0.12±0.04	0.89±0.26
	5	3.0	4.3±0.2	11±0.4	100±0	0.052±0.000	1.0±0.0
	10	3.0	3.8±0.1	11±0.4	100±0	0.054±0.000	1.0±0.0
	20	3.0	3.9±0.1	11±0.2	100±0	0.054±0.000	1.0±0.0
	50	3.0	3.8±0.1	11±0.2	100±0	0.054±0.000	1.0±0.0
Zig-zag (c=3)	3	1.4	0.0±0.0	6.3±2.5	78.6±9.2	0.12±0.03	0.55±0.11
	5	1.2	0.0±0.0	11±0.8	87.6±0.3	0.19±0.00	0.66±0.00
	10	3.0	0.0±0.0	11±0.4	88.0±0.1	0.068±0.001	0.67±0.00
	20	3.0	0.0±0.0	11±0.0	88.0±0.0	0.065±0.002	0.67±0.00
	50	3.0	0.0±0.0	11±0.0	88.0±0.0	0.066±0.002	0.67±0.00

There are notable improvements in the performance for the FCM-based active learning approach, especially for a small number of labeled patterns. The FCM-based results are very consistent as well. In many cases standard deviations of zero are shown in the table. This is a byproduct of the fact that the standard deviations are too small to be reported for the number of significant digits given in the table. In many cases, the standard deviations were not exactly zero

but they were very small as a result of consistent pattern selection via active learning. The polynomial kernel performs much better for the line data set than the Gaussian kernel.

7.3.2. MACHINE LEARNING DATA

Here we describe the experimental results for kernel parameter learning on the machine learning data sets [5] with Gaussian kernel. As before the results were averaged over 20 runs.

Table 19. Machine Learning data – Gaussian kernel parameter learning (no active learning)

Data Sets	*M*	*Uniform Random*				
		m	σ^2	cr	re	nmi
Breast (c=2)	3	3.0	244±673	89.3±11.8	0.29±0.35	0.61±0.23
	5	2.5	14.4±7.2	91.9±11.0	0.19±0.34	0.68±0.21
	10	1.2	20.2±8.6	96.2±0.5	0.097±0.013	0.76±0.02
	20	1.2	20.4±4.5	96.2±0.2	0.091±0.003	0.76±0.01
	50	1.2	21.1±2.7	96.3±0.2	0.089±0.006	0.76±0.01
Ionosphere (c=2)	3	1.4	78.5±242.1	70.0±4.3	0.50±0.38	0.14±0.10
	5	2.0	46.5±32.4	70.8±3.6	0.30±0.32	0.16±0.07
	10	3.0	50.1±30.3	72.9±0.6	0.15±0.05	0.17±0.03
	20	3.0	79.2±27.9	72.3±0.6	0.13±0.09	0.16±0.04
	50	3.0	90.0±12.5	72.2±0.5	0.098±0.018	0.15±0.01
Iris (c=3)	3	2.0	2.2±2.8	63.8±24.7	0.53±0.43	0.42±0.35
	5	1.4	3.4±2.6	80.8±10.7	0.14±0.21	0.61±0.14
	10	1.4	3.7±1.3	84.1±3.0	0.068±0.007	0.67±0.00
	20	1.7	3.4±0.7	84.0±0.0	0.054±0.007	0.67±0.00
	50	1.7	3.9±0.4	84.0±0.0	0.049±0.002	0.67±0.00
Wine (c=3)	3	1.2	10.5±7.6	84.9±23.1	0.35±0.38	0.70±0.37
	5	1.2	13.9±5.7	94.6±12.9	0.17±0.20	0.86±0.20
	10	1.2	18.4±2.8	97.6±0.3	0.11±0.00	0.90±0.01
	20	1.2	18.2±1.3	97.7±0.1	0.11±0.00	0.91±0.01
	50	1.2	18.8±0.9	97.8±0.0	0.11±0.00	0.91±0.00
Diabetes (c=2)	3	3.0	122±521	66.9±1.9	0.63±0.37	0.058±0.051
	5	2.5	14.9±13.4	68.6±1.3	0.24±0.13	0.10±0.01
	10	2.0	15.2±5.9	69.0±1.1	0.15±0.08	0.094±0.009
	20	2.5	19.8±3.3	69.5±0.3	0.12±0.03	0.094±0.003
	50	2.5	21.3±2.3	69.6±0.1	0.11±0.02	0.092±0.004
SPECT-F (c=2)	3	1.2	89.5±354.9	79.4±0.0	0.68±0.36	0.057±0.059
	5	1.4	30.1±51.0	79.4±0.0	0.50±0.38	0.072±0.053
	10	1.7	47.1±23.5	79.4±0.0	0.19±0.06	0.10±0.01
	20	2.0	50.0±9.4	79.4±0.0	0.16±0.02	0.091±0.007
	50	2.0	58.9±6.4	79.4±0.0	0.13±0.01	0.89±0.01

Table 20. Machine Learning data – Gaussian kernel parameter learning (active learning)

Data Sets	*M*	*FCM-based Approach*				
		m	σ^2	cr	re	nmi
Breast (c=2)	3	3.0	15.9±0.1	96.6±0.0	0.054±0.000	0.78±0.00
	5	3.0	15.9±0.0	96.6±0.0	0.054±0.000	0.78±0.00
	10	1.2	24.0±0.0	96.2±0.0	0.090±0.000	0.75±0.00
	20	2.0	28.8±0.1	95.6±0.0	0.051±0.000	0.73±0.00
	50	1.4	34.7±0.1	95.5±0.0	0.089±0.000	0.72±0.00
Ionosphere (c=2)	3	1.4	40.6±0.0	74.1±0.0	0.20±0.00	0.22±0.00
	5	3.0	67.4±0.0	73.2±0.0	0.14±0.00	0.17±0.00
	10	3.0	88.7±0.8	72.1±0.1	0.097±0.001	0.14±0.002
	20	3.0	142.7±0.3	72.1±0.0	0.052±0.000	0.14±0.00
	50	3.0	160.6±0.5	70.7±0.0	0.045±0.000	0.11±0.00
Iris (c=3)	3	1.7	3.7e-3±3.6e-3	36.1±0.7	0.99±0.00	0.47±0.014
	5	1.7	7.0±0.0	84.2±0.3	0.060±0.000	0.67±0.00
	10	1.7	7.1±0.0	84.7±0.0	0.061±0.000	0.68±0.00
	20	2.0	5.3±0.0	84.0±0.0	0.041±0.000	0.67±0.00
	50	2.0	4.5±0.0	84.0±0.0	0.044±0.000	0.67±0.00
Wine (c=3)	3	2.5	11.9±0.1	97.2±0.0	0.25±0.00	0.90±0.00
	5	1.4	11.5±0.1	97.8±0.0	0.11±0.00	0.91±0.00
	10	1.2	21.4±0.2	97.2±0.1	0.11±0.00	0.89±0.01
	20	1.2	26.7±0.1	96.6±0.0	0.12±0.00	0.88±0.00
	50	1.2	24.2±0.0	96.6±0.0	0.11±0.00	0.87±0.00
Diabetes (c=2)	3	2.5	273.6±533.2	66.8±1.9	0.57±0.49	0.055±0.037
	5	2.0	10.3±2.5	68.8±0.3	0.20±0.04	0.10±0.01
	10	3.0	31.4±0.1	69.4±0.0	0.076±0.000	0.087±0.000
	20	2.5	57.0±0.0	69.0±0.0	0.028±0.000	0.076±0.000
	50	3.0	52.3±0.0	69.1±0.0	0.032±0.000	0.080±0.000
SPECT-F (c=2)	3	1.4	36.7±0.1	79.4±0.0	0.22±0.00	0.10±0.00
	5	1.4	27.6±0.0	79.4±0.0	0.25±0.00	0.12±0.00
	10	1.4	193.7±88.8	79.4±0.0	0.10±0.09	0.079±0.019
	20	1.4	221.6±0.1	79.4±0.0	0.059±0.000	0.070±0.000
	50	1.7	168.1±0.8	79.4±0.0	0.053±0.000	0.073±0.000

The results given in Table 19 and Table 20 show that the FCM-based labeling approach performs better for a small number of labeled patterns on the breast cancer data set and other data sets. This demonstrates that a few well selected labeled patterns can greatly enhance the clustering performance. The results for the polynomial kernel are provided in Table 21 and Table 22.

Table 21. Machine learning data – polynomial kernel parameter learning (no active learning)

Data Sets	*M*	*Uniform Random*					
		m	θ	d	cr	re	nmi
Breast (c=2)	3	2.5	0.6±1.6	2.1±2.4	94.9±5.5	0.027±0.022	0.75±0.15
	5	1.2	0.5±1.4	2.2±2.0	95.3±3.0	0.087±0.060	0.75±0.10
	10	2.0	0.3±1.2	1.5±1.6	96.9±1.0	0.032±0.020	0.80±0.03
	20	2.0	0.0±0.1	1.5±0.9	96.6±1.1	0.034±0.015	0.79±0.04
	50	2.0	0.0±0.1	1.2±0.6	96.9±0.8	0.029±0.010	0.80±0.02
Ionosphere (c=2)	3	3.0	3.1±8.9	5.0±4.2	66.2±2.3	0.086±0.032	0.17±0.04
	5	1.4	20±37	6.6±4.0	66.7±3.5	0.18±0.04	0.19±0.06
	10	2.5	12±26	4.3±4.4	67.7±2.7	0.066±0.026	0.17±0.05
	20	3.0	32±41	6.4±4.4	68.1±3.2	0.070±0.014	0.21±0.06
	50	3.0	20±24	5.3±4.1	67.7±2.3	0.071±0.011	0.21±0.05
Iris (c=3)	3	3.0	0.4±0.9	5.3±4.1	79.7±7.8	0.049±0.032	0.58±0.17
	5	3.0	0.0±0.0	2.4±2.0	83.6±0.9	0.034±0.011	0.66±0.04
	10	3.0	0.0±0.0	1.7±1.3	83.0±0.6	0.031±0.005	0.68±0.03
	20	3.0	0.0±0.0	2.0±1.5	82.7±0.8	0.032±0.008	0.67±0.03
	50	3.0	0.0±0.0	1.8±1.0	82.8±0.7	0.030±0.002	0.67±0.03
Wine (c=3)	3	2.5	0.1±0.3	1.2±0.5	93.9±0.3	0.027±0.013	0.81±0.01
	5	2.5	0.1±0.1	1.4±1.2	92.5±6.0	0.036±0.048	0.80±0.07
	10	2.5	0.0±0.0	1.2±0.6	93.9±0.3	0.028±0.013	0.81±0.00
	20	2.5	0.0±0.0	1.2±0.6	93.9±0.3	0.028±0.013	0.81±0.00
	50	2.5	0.0±0.0	1.4±0.8	94.0±0.5	0.032±0.018	0.81±0.01
Diabetes (c=2)	3	1.7	1.0±1.6	4.2±3.9	67.6±2.2	0.14±0.09	0.12±0.04
	5	3.0	1.4±4.1	5.7±4.3	67.2±2.0	0.23±0.03	0.11±0.04
	10	2.0	1.3±4.7	5.4±4.0	67.3±2.0	0.19±0.08	0.11±0.04
	20	2.0	1.0±2.6	5.1±3.8	67.5±1.9	0.20±0.07	0.11±0.05
	50	3.0	0.5±1.3	4.4±3.4	67.9±1.5	0.24±0.02	0.12±0.02
SPECT-F (c=2)	3	1.4	2.4±2.3	5.4±3.2	79.4±0.0	0.19±0.08	0.15±0.03
	5	1.4	0.9±1.9	6.1±4.2	79.4±0.0	0.18±0.08	0.13±0.06
	10	1.7	1.3±3.0	5.6±4.4	79.4±0.0	0.16±0.09	0.15±0.05
	20	1.7	0.1±0.5	5.8±4.3	79.4±0.0	0.17±0.09	0.14±0.05
	50	1.4	0.0±0.0	2.4±0.9	79.4±0.0	0.086±0.005	0.17±0.01

Table 22. Machine learning data – polynomial kernel parameter learning (active learning)

Data Sets	*M*	*FCM-based Approach*					
		m	θ	d	cr	re	nmi
Breast (c=2)	3	2.5	0.0±0.0	1.0±0.0	97.2±0.0	0.015±0.000	0.81±0.00
	5	1.2	0.0±0.0	1.0±0.0	97.2±0.0	0.043±0.000	0.81±0.00
	10	1.4	0.0±0.0	1.1±0.4	97.1±0.6	0.043±0.013	0.81±0.02
	20	2.5	0.0±0.0	1.0±0.0	97.2±0.0	0.015±0.000	0.81±0.00
	50	1.2	0.0±0.0	1.0±0.0	97.2±0.0	0.043±0.000	0.81±0.00
Ionosphere (c=2)	3	3.0	60±1.5	11±0.2	71.8±0.1	0.070±0.000	0.29±0.00
	5	3.0	72±1.8	11±0.2	71.8±0.0	0.066±0.000	0.28±0.00
	10	2.5	95±0.1	11±0.0	72.9±0.0	0.070±0.000	0.26±0.00
	20	2.5	95±3.1	11±0.3	72.9±0.0	0.070±0.000	0.26±0.00
	50	3.0	0.0±0.0	2.0±0.0	69.5±0.0	0.062±0.000	0.19±0.00
Iris (c=3)	3	3.0	0.0±0.0	3.0±0.0	82.0±0.0	0.033±0.000	0.63±0.00
	5	2.5	0.0±0.0	1.2±0.6	83.1±0.6	0.029±0.001	0.69±0.02
	10	3.0	0.1±0.0	3.0±0.0	82.0±0.0	0.028±0.000	0.63±0.00
	20	3.0	0.0±0.0	3.0±0.0	82.0±0.0	0.031±0.000	0.63±0.00
	50	3.0	0.0±0.0	3.0±0.0	82.0±0.0	0.033±0.000	0.63±0.00
Wine (c=3)	3	2.5	0.0±0.0	1.1±0.4	93.9±0.3	0.026±0.010	0.81±0.00
	5	2.5	0.0±0.0	1.1±0.4	93.9±0.3	0.026±0.010	0.81±0.00
	10	2.5	0.0±0.0	1.0±0.0	93.8±0.0	0.023±0.000	0.81±0.00
	20	2.5	0.0±0.0	1.3±0.7	94.0±0.4	0.030±0.016	0.81±0.00
	50	2.5	0.0±0.0	1.1±0.4	93.9±0.3	0.026±0.010	0.81±0.00
Diabetes (c=2)	3	2.0	1.7±2.0	4.4±2.9	67.7±1.6	0.21±0.07	0.13±0.01
	5	1.7	13±0.5	11±0.4	65.1±0.0	0.075±0.000	0.12±0.00
	10	3.0	13±0.4	11±0.2	65.8±0.0	0.22±0.00	0.12±0.00
	20	1.7	19±0.4	11±0.2	65.6±0.1	0.072±0.000	0.12±0.00
	50	1.2	0.0±0.0	3.0±0.0	68.2±0.1	0.12±0.00	0.12±0.00
SPECT-F (c=2)	3	1.7	0.0±0.0	1.0±0.0	79.4±0.0	0.064±0.000	0.19±0.00
	5	1.7	0.0±0.0	1.0±0.0	79.4±0.0	0.064±0.000	0.19±0.00
	10	1.7	0.0±0.0	1.2±0.6	79.4±0.0	0.068±0.015	0.18±0.01
	20	1.7	0.0±0.1	1.1±0.4	79.4±0.0	0.066±0.011	0.19±0.01
	50	1.7	1.1±4.7	1.1±0.2	79.4±0.0	0.064±0.003	0.19±0.01

A significant improvement in clustering performance is reported for a small number of labeled patterns with the polynomial kernel.

7.4. DISCUSSION

A new active learning framework is presented which is directly applicable to fuzzy clustering approaches. The patterns selected by active learning are subsequently labeled with proximity information and used to learn the parameters of the kernel function in proximity fuzzy clustering. This provides a way to learn the parameters of the kernel function and support clusters of non-spherical geometries. We demonstrated the effectiveness of proximity fuzzy clustering on several synthetic and real data sets with two different kernel functions. There is a significant improvement in performance by labeling a few well selected patterns in comparison with random selection to learn the parameters of the kernel function. We concluded that active learning in the context of partially supervised fuzzy clustering can great enhance the discovery of structure even for a small number of labeled data points. Also, the patterns selected for labeling by active learning are very consistent as demonstrated by the small standard deviations in the performance. Finally, we have demonstrated that proximity-based kernelized fuzzy clustering is capable of learning its own kernel parameters using partially supervised learning. Futurework aims to address the issue of kernel selection in partially supervised learning, i.e. structural optimization of the kernel function.

8. MULTI-PROXIMITY FUZZY CLUSTERING

Proximity fuzzy clustering uses proximity information, i.e. degree of similarity, to cluster a set of objects into a partition. The proximity fuzzy clustering algorithm produces a partition using only proximity between pairs of data in contrast to more traditional methods that use distance between data and prototypes as done with FCM, K-Means and other clustering algorithms. A logical extension to proximity fuzzy clustering is to consider clustering from several different sources of proximity information. The new approach is termed multi-proximity fuzzy clustering.

Multi-proximity fuzzy clustering is closely related to ensemble clustering where the objective is to form a single partition from "*H*" views of the data as shown in Figure 31.

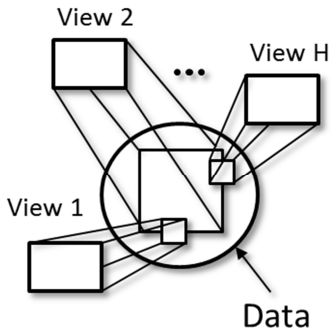


Figure 31. Views of data

Ensemble clustering is a two-step process:

(a) partition each view of the data, and
(b) aggregate the partitions into a single partition.

The problem is that producing the partitions for each view and then aggregating them to form a single partition are not independent steps and require some "cooperation" in order produce an optimal partition. Multi-proximity fuzzy clustering thereby combines both steps into a single multi-view clustering algorithm thereby allowing the reconciliation of differences in the views to occur during clustering rather than at aggregation.

8.1. MOTIVATION

Many ensemble methods fail to reconcile partitions when there are only a few sources including the CSPA and EAC algorithms. In this section, it is demonstrated how multi-proximity fuzzy clustering out-performs these traditional ensemble clustering approaches on a simple synthetic data set. A simple example where there are two projections (*H*=2) of a three dimensional data set consisting of three clusters is shown in Figure 32. These views are projections of the three dimensional data onto two different planes. Clearly this is a trivial example but similar situations occur where there are different ways of viewing or interpreting similarity between objects. For example, several experts can provide some proximity hints but it is highly unlikely that there will be complete agreement among the experts as to their correctness.

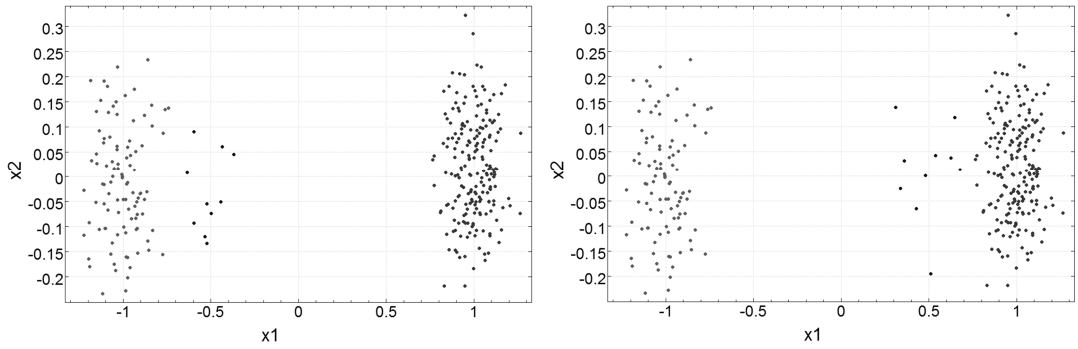


Figure 32. Demonstration data set of multi-proximity fuzzy clustering

A close inspection of the plots reveal that there are 10 data points that are close to the left cluster in view 1 and these same 10 points are close to the right cluster in view 2. All the other points are the same. By applying multi-proximity fuzzy clustering one obtains the following structure in the data shown in Figure 33 where the clusters obtained are circled in the figure. Since the views are the only visual representation available for this data, the cluster structure is shown in both views of the data.

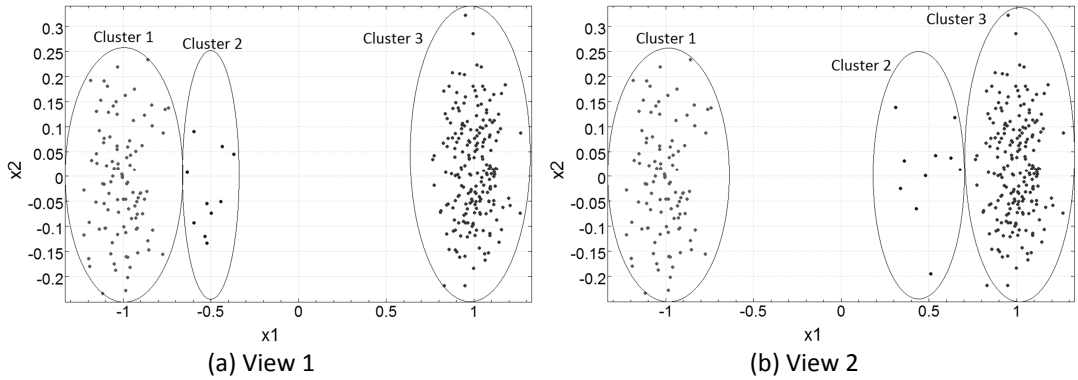
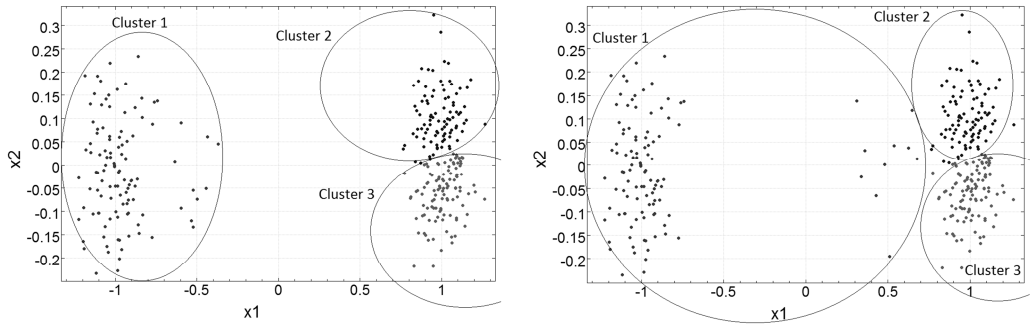


Figure 33. Multi-proximity fuzzy clustering on demonstration data set (*m*=1.5, *c*=3)

Multi-proximity clustering is able to easily pick out the 10 data points (Cluster 2) which appear in different positions in each view.

With ensemble clustering, both the CSPA and EAC algorithms produce the similar cluster structure, see Figure 34. Relational FCM is employed to aggregate the partitions.



(a) View 1 (b) View 2

Figure 34. Ensemble clustering on demonstration data set (*m*=1.5, *c*=3)

Ensemble clustering fails to recognize the cluster with 10 points in the data set. The reason is that there are not enough votes available to infer the correct partition.

8.2. MULTI-PROXIMITY CLUSTERING FRAMEWORK

In the case where there are multiple sources of proximity information [193], there can be significant disagreements on the level of proximity between pairs of objects. Thus a serious downfall of proximity fuzzy clustering is the inability to reconcile proximity information from several different sources. For example, suppose a group of experts independently labeled a data set. How might the multiple proximity sources be clustered and combined effectively? One solution is to cluster the proximity values of each expert labeled data set individually and then perform ensemble clustering to combine the results. Often this involves voting or use of the majority principle to reconcile differences in the clusterings. As shown in the previous section, this approach does not always work very well. However, the downside is that we are reconciling differences post-clustering. We propose an architecture called multi-proximity fuzzy clustering that produces a partition by accounting for data from several sources directly in the clustering as depicted in Figure 35.

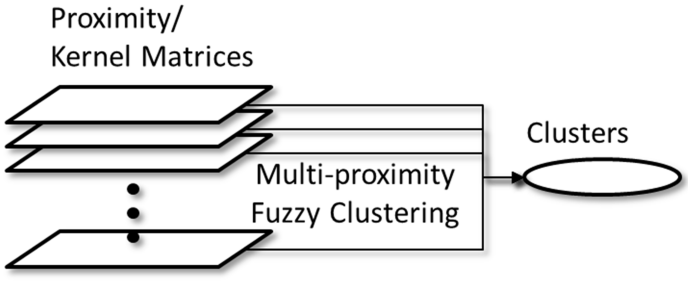


Figure 35. Multi-proximity fuzzy clustering framework

The inputs and outputs of the multi-proximity fuzzy clustering framework are shown in Figure 36.

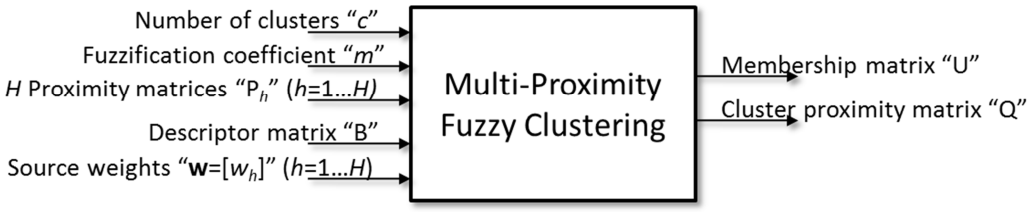


Figure 36. Multi-proximity fuzzy clustering architecture

Multi-proximity fuzzy clustering minimizes the following objective function with respect to membership U and cluster proximities Q subject to the constraint (6-2), i.e. $\sum_{i=1}^{c} u_{ik} = 1$:

$$Obj = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^{m} \sum_{h=1}^{H} w_h \left(\sum_{j=1}^{N} b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{ij} \right)^2 \right) \qquad \textbf{(8-1)}$$

where H is the number of proximity sources, w_h is the weight assigned to each source, $p_h\left(\mathbf{x}_k, \mathbf{x}_j\right)$ is the proximity hint between objects $\mathbf{x}_k, \mathbf{x}_j \in X$, $j,k=1\ldots N$, for each source, and $\mathrm{B} = \{b_{hk} \in \{0,1\} \mid h = 1\ldots H, k = 1\ldots N\}$ is a Boolean descriptor matrix that indicates the patterns that are available (1) or not-available (0) for each source $h=1\ldots H$. The Boolean matrix B provides a way for multi-proximity fuzzy clustering to consider sources where the proximity information may not be complete for all data points. The minimum of expression (8-1) is found by introducing a Lagrange multiplier to handle the constraint.

The cluster proximity values are found by determining an expression for q_{ij} where $\frac{\partial Obj}{\partial q_{ij}} = 0$.

$$\frac{\partial Obj}{\partial q_{ij}} = 2\sum_{k=1}^{N}\sum_{h=1}^{H} u_{ik}^{m} w_h b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{ij} \right) = 0 \qquad \textbf{(8-2)}$$

$$q_{ij} = \frac{\sum_{k=1}^{N}\sum_{h=1}^{H} w_h b_{hk} b_{hj} u_{ik}^{m} p_h\left(\mathbf{x}_k, \mathbf{x}_j\right)}{\sum_{k=1}^{N}\sum_{h=1}^{H} u_{ik}^{m} w_h b_{hk} b_{hj}} \qquad \textbf{(8-3)}$$

An expression for membership u_{ik} is derived by finding $\frac{\partial Obj}{\partial u_{ik}} = 0$.

$$\frac{\partial Obj}{\partial u_{ij}} = m u_{ik}^{m-1} \left(\sum_{h=1}^{N}\sum_{j=1}^{N} w_h b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{ij} \right)^2 \right) - \lambda = 0 \qquad \textbf{(8-4)}$$

where λ is the Lagrange multiplier for the constraint.

$$u_{ik} = \left(\frac{\lambda}{m\left(\sum_{h=1}^{H}\sum_{j=1}^{N} w_h b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{ij} \right)^2 \right)} \right)^{\frac{1}{m-1}} \qquad \textbf{(8-5)}$$

Applying the constraint conditions to expression (8-5) yields

$$\sum_{h=1}^{c} u_{ik} = \sum_{h=1}^{c} \left(\frac{\lambda}{m\left(\sum_{h=1}^{H} \sum_{j=1}^{N} w_h b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{hj} \right)^2 \right)} \right)^{\frac{1}{m-1}} = 1 \quad \textbf{(8-6)}$$

$$\left(\frac{\lambda f_k}{m} \right)^{\frac{1}{m-1}} = \sum_{h=1}^{c} \left(\sum_{h=1}^{H} \sum_{j=1}^{N} w_h b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{hj} \right)^2 \right)^{\frac{1}{m-1}} \quad \textbf{(8-7)}$$

Substituting expression (8-7) into expression (8-5) gives

$$u_{ik} = \frac{1}{\sum_{h=1}^{c} \left(\frac{\sum_{h=1}^{H} \sum_{j=1}^{N} w_h b_{hk} b_{hj} \left(p_i\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{ij} \right)^2}{\sum_{h=1}^{H} \sum_{j=1}^{N} w_h b_{hk} b_{hj} \left(p_h\left(\mathbf{x}_k, \mathbf{x}_j\right) - q_{hj} \right)^2} \right)^{\frac{1}{m-1}}} \quad \textbf{(8-8)}$$

The algorithm is given by the pseudocode.

Multi-Proximity Fuzzy Clustering Algorithm

Input: X, *c*, *m*, B, **w**, *H*

Output: V, U, Q

1. Generate a random initial membership matrix u_{ik} subject to the constraint (6-2)
2. Do
   a. Update q_{ij} from expression (8-3)
   b. Update u_{ik} from expression (8-8)
3. Until convergence in membership values has been observed, i.e. $\left\| \mathrm{U}(iter) - \mathrm{U}(iter-1) \right\|_{\mathrm{F}} < \varepsilon$
4. Estimate prototypes using expression (6-7)

The algorithm is very similar to proximity fuzzy clustering except where *H* different proximity functions are available – one for each proximity source.

8.3. LEARNING THE WEIGHTS

A partially supervised method for learning the weights is now described. This approach labels a subset of the data with class labels via active learning. The set of class labels is denoted as $Y_h=\{y_k | k=1\ldots M\}$ where $M<N$ is the number of labeled patterns in each data source $h=1\ldots H$. These labels are used to evaluate proximity fuzzy clustering on each of the proximity matrices. An active learning procedure is used to select the most informative patterns to label. The process is depicted in Figure 37.

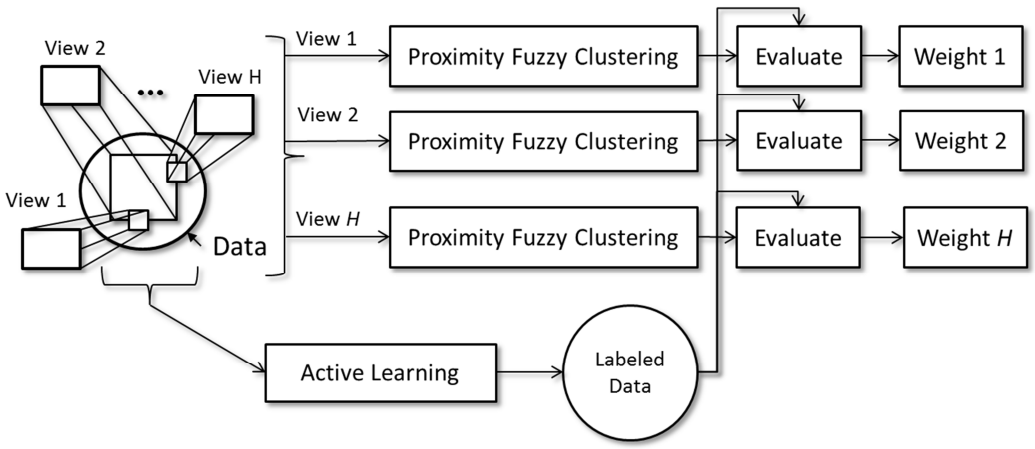


Figure 37. Learning the weights for each view/source

Once the weights are determined, they are used to execute multi-proximity fuzzy clustering. The idea is to cluster each source/view of the data with the original proximity fuzzy clustering algorithm, evaluate the performance of each and then determine a weight proportional to performance (larger weight equals better performance). In this way, the sources where the performance is very good are more important in multi-proximity fuzzy clustering. An active learning procedure is used to label a small yet effective data of size *M* for measuring the performance each view.

Multi-proximity fuzzy clustering which includes the learning of the weights is described by the following steps.

Multi-Proximity Fuzzy Clustering Algorithm with Active Learning of the Weights

Input: X, *c*, *m*, B, *M*, *H*, α
Output: V, U, Q, **w**

1. Determine a labeled data set $\hat{\mathrm{X}} \subset \mathrm{X}$ using active learning containing *M* patterns
2. For *h*=1…*H*
   a. Proximity Fuzzy Clustering with proximity measured by the function $p_h(\mathbf{x}_j,\mathbf{x}_k)$, $\forall j,k = 1...N$
   b. Determine the classification rate ω_h using labeled data set $\hat{\mathrm{X}}$
3. Calculate the weights according to the expression

$$w_h = \left(\omega_h\right)^{\alpha} \tag{8-9}$$

4. Normalize the weights, i.e.

$$w_h = \frac{w_h}{\sum_{i=1}^{H} w_h} \tag{8-10}$$

5. Perform multi-proximity fuzzy clustering

Here, $\alpha \in [0,\infty)$ is a parameter that controls the contraction or expansion of the weights. In this approach, the labeled patterns are important for evaluating the clustering. Hence, the idea is select a small number of well-selected patterns to assess the clustering performance for each proximity view. Better performing proximity sources will have a greater weight in the overall multi-proximity fuzzy clustering.

8.4. EXPERIMENTS

8.4.1. SYNTHETIC DATA

The first experiments involve setting the weight of each view to 1 and evaluating the performance of multi-proximity fuzzy clustering on synthetic data. We construct multiple views of the data using different combinations of kernel functions and kernel parameters. Table 23 shows the parameters and kernel functions that we use for each data set.

Table 23. Construction of the views on synthetic data

Data Set	*H*	Kernel Function	Kernel Parameters
Ring	10	Gaussian	$\sigma^2 \in \{0.01, 0.1, 0.5, 1, 2, 4, 8, 16, 25, 100\}$
Zig-Zag	6	Gaussian	$\sigma^2 \in \{0.1, 0.5, 1, 2, 4, 8\}$
Zig-Zag (with alternate views)	4	Gaussian	$\sigma^2 \in \{0.1, 0.5, 1, 2\}$

The performance of multi-proximity fuzzy clustering is given in Table 24. The value of *m* was chosen by changing its value over the set {1.2, 1.4, 1.7, 2.0, 2.5, 3.0} and selecting the value that produces the maximum classification rate. The value of *c* was set according to the number of known classes in the data set.

Table 24. Performance on synthetic data

Data Set	*c*	*m*	Classification Rate
Ring	2	2.0	100
Zig-Zag	3	1.3	83.2
Zig-Zag (with alternate views)	3	2.0	100

It was observed that by adding views produced by kernel parameters greater than 2 on the zig-zag data set significantly reduced the classification rate or accuracy of the clustering. This provides strong motivation for assigning weights to the views which we will analyze with experiments in a later section.

8.4.2. REAL-WORLD DATA

We continue to hold the value of the weights at 1 and evaluate the performance on real-world data. We construct multiple views of the data using different combinations of kernel functions and kernel parameters. Table 25 shows the parameters and kernel functions that we use for each data set.

Table 25. Construction of the views on real-world data

Data Set	*H*	Kernel Function	Kernel Parameters
Ionosphere	10	Gaussian	$\sigma^2 \in \{0.01, 0.1, 0.5, 1, 2, 4, 8, 16, 25, 100\}$
Iris	12	Gaussian Polynomial	$\sigma^2 \in \{0.1, 0.5, 1, 2, 4, 8\}$ $d \in \{2, 3\}$, $\theta \in \{0, 0.5, 1\}$
Iris	5	Gaussian	$\sigma^2 \in \{10, 25, 50, 75, 100\}$

The performance of multi-proximity fuzzy clustering is given in Table 26.

Table 26. Performance on real-world data

Data Set	*c*	*m*	Classification Rate
Ionosphere	2	2.0	72.1
Iris	3	2.0	66.7
Iris (with alternate views)	3	2.0	84.7

The performance on the iris data set is quite poor. This is largely due to the overlapping clusters being treated as one cluster and the large number of views that have equal weighting. The

performance improves when we reduce the number of views to 5. The performance of multi-kernel clustering on the ionosphere data set has improved performance particularly when compared with FCM and Gustafson-Kessel FCM (see Chapter 4).

8.4.3. EXPERIMENTS ON LEARNING WEIGHTS

Clearly from these experiments weighting each view equally does not always produce desirable performance. Therefore some procedure for optimizing the weights is necessary such as the one outlined in section 8.3. The FCM-based active learning framework is employed to select the labeled patterns. Performance is evaluated for the values of M={5, 10, 25, 50, 100}. The value of m was chosen by changing its value over the set {1.2, 1.4, 1.7, 2.0, 2.5, 3.0}. The value of c was set according to the number of known classes in the data set. The value of p which is used to determine the optimal set of weights was selected by manually adjusting the value to maximize the classification performance. Each experiment is repeated 20 times. The multiple views of the data are constructed using a kernel function with several different selections for the parameters.

8.4.3.1. SYNTHETIC DATA

Table 27 shows the parameters and kernel functions for each data set.

Table 27. Construction of the views on synthetic data

Data Set	H	Kernel Function	Kernel Parameters
Ring	10	Gaussian	$\sigma^2 \in \{0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16, 30\}$
Zig-Zag	10	Gaussian	$\sigma^2 \in \{0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16, 30\}$

The performance of multi-proximity fuzzy clustering is given in Table 28.

Table 28. Performance on synthetic data

Data Set	c	M	m	p	Classification Rate
Ring	2	5	2.0	1	100.0±0.0
Zig-Zag	3	10	1.4	10	100.0±0.0

The weights of the views are provided in Table 29.

Table 29. Weighting of the views

Data Set	Kernel parameters of each view									
	0.01	0.05	0.1	0.5	1	2	4	8	16	30
Ring	1.0	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Zig-Zag	0.349	1.0	1.0	1.0	0.349	0.107	0.107	0.107	0.107	0.107

Excellent performance is observed when actively learning the weights. In this way, it is not necessary to optimize the parameters of the kernel but rather select several different kernel functions and parameters and execute multi-proximity fuzzy clustering. The zig-zag data set required a much larger value for p in order to achieve the optimal performance.

8.4.3.2. REAL-WORLD DATA

We now conduct experiments on real-world data where the weights of the views are learned via the partially supervised learning mechanism. Table 30 shows the parameters and kernel functions that we use for each data set.

Table 30. Construction of the views on real-world data

Data Set	*H*	Kernel Function	Kernel Parameters
Ionosphere	10	Gaussian	$\sigma^2 \in \{0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16, 30\}$
Iris	10	Gaussian	$\sigma^2 \in \{0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16, 30\}$
Wine	10	Gaussian	$\sigma^2 \in \{0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16, 30\}$

The performance of multi-proximity fuzzy clustering is given in Table 31.

Table 31. Performance on synthetic data

Data Set	c	*M*	*m*	*p*	Classification Rate
Ionosphere	2	10	2.5	10	72.4±0.1
Iris	3	10	2.0	10	87.3±0.0
Wine	3	10	1.2	10	97.8±0.0

The weighs of the views are provided in Table 29.

Table 32. Weighting of the views

Data Set	Kernel parameters of each view									
	0.01	0.05	0.1	0.5	1	2	4	8	16	30
Ionosphere	0.107	0.107	0.107	0.107	0.107	0.107	0.349	0.349	0.349	0.349
Iris	1.05e-4	9.77e-4	9.77e-4	9.77e-4	9.77e-4	0.0282	0.0282	0.107	0.107	0.107
Wine	0.0282	0.0282	0.107	0.0282	0.107	0.107	1.0	1.0	1.0	1.0

Interestingly, the ionosphere data performs best with a small value of *p* indicative of larger weights and of the need for combining multiple views of the data to achieve high performance. The iris data set on the other hand required a large value of *p* in order to achieve a high classification rate. The classification rate for the iris data set is better than standard FCM clustering and KFCM-K clustering with the Gaussian kernel (85.3%) as reported in Table 9. Since we are using the Gaussian kernel, we must conclude that multi-proximity fuzzy clustering is an improvement over KFCM-K with the same kernel function. KFCM-K with the polynomial kernel achieves 88.7% which very close to the performance achieved by multi-proximity fuzzy clustering. An added benefit over KFCM-K is that there is no need to optimize the values of the kernel parameters to achieve these results.

8.5. DISCUSSION

The performance of weighted multi-proximity fuzzy clustering is quite good when we construct different views using a variety of kernel functions and parameters. Although the performance is competitive with other kernel methods on the data sets provided, the main advantage is that it avoids the need to optimize the parameters of the kernel. It also provides the ability to aggregate multiple and very different kernel functions (or proximity functions for that matter) together which is very useful when two or more kernel functions provide a view of the data in the form of a proximity matrix that are co-essential for accurate partitioning of the data.

9. GRAPH CLUSTERING

9.1. INTRODUCTION

Graph clustering is a common problem encountered in fundamental research and applications, cf. [171][28][149][174][211][209][55]. For instance, we can refer here to software engineering such as using call graphs to locate faults in software problems [55]. In the studies reported in the literature, the term *graph clustering* is used primarily with regard to clustering the nodes of a graph. In contrast, the focus of our study is on clustering a collection of graphs. There are a number of practical problems where one must find structure in a collection of graphs. One of these problems is in the domain of software engineering where the aim is to find patterns in the design of software systems. This information can form important advice for software engineers to facilitate software design reuse. While being practically relevant, the problems falling under this realm of graph clustering are quite challenging. One of the reasons is that clustering of such objects is poorly suited for traditional, well-established clustering approaches such as K-Means, hierarchical clustering, Fuzzy C-Means (FCM), self-organizing maps, and entropy-based clustering. These clustering algorithms assume the data constitute a set of vectors. None of these "traditional" algorithms can be readily applied to a broad class of real-world problems, where data are expressed solely by some relational information. Of particular importance is the problem of clustering a collection of directed unweighted acyclic graphs described by proximity relationships that quantify the closeness of pairs of graphs. One can envision that the proximity values indicate the closeness of graphs by measuring their "similarity" via the number of common nodes and edges in the graph. Given a way in which the notion of closeness becomes articulated, relational clustering seems to be an appealing way to consider.

One such relational clustering approach is the relational Fuzzy C-Means (FCM) algorithm [86][85], which is a well-known in the field of relational clustering. It works by assuming the relationships expressed in terms of Euclidean distances thereby implying that the objects being clustered (in our case graphs) can be represented as vectors located in Euclidean space, cf. [85]. A non-Euclidean version was introduced in [85]. In this method, one transforms non-Euclidean relational data into the Euclidean relational ones. Spectral clustering is a common relational clustering algorithm, which determines a Laplacian matrix from the relational data and using the principle components of the matrix, forms a vector feature space where traditional techniques such as K-Means and FCM are applied, cf. [58][138][211]. This approach is computationally expensive. The shortcomings of current developments in relational clustering give rise to the introduction of a new proximity-based fuzzy clustering algorithm in this study.

Clustering relational data is an important consideration; however, how the relational data are produced is equally important. We represent graphs as a string and measure their distance using the graph edit distance, cf. [174][62] , which is determined by considering the minimum number of "add" and "remove" operations required to transform one graph into the other. However, our case study involves clustering a set of directed unweighted acyclic graphs, particularly our case study includes special properties assigned to the nodes and edges that are specific to software engineering. Thus we necessarily introduce a new measure of distance based on the edit distance that accounts for the additional properties on the edges as well as the semantic similarity of the text describing the nodes.

A number of structural graph matching algorithms have been developed, cf. [70][137][132][150][106][175]. The authors in [70] utilize a random walk algorithm to find both exact and approximate graph matching. They evaluate their approach successfully with an image retrieval problem for inexact graph matching and an isomorphism data set for exact graph matching. An Expectation-Maximization (EM) algorithm is developed to matching graph for

inexact graph matching, cf. [137]. They do not consider additional attributes for the nodes and edges. They experiment with image data and conclude that their approach is successful in matching graphs based on structure. The weights of in the graph edit distance is analyzed in [150] where self-organizing maps (SOMs) are used to determine the cost of the edit operations in graph matching. They report significant improvement in performance for their approach to graph matching based on a nearest-neighbor classifier. A structural image recognition approach is introduced in [106] using the attributed relational graph (ARG) matching technique with modified operators in genetic algorithms. They report improvements in the convergence speed of genetic algorithms over the standard operators when applied to some image data sets. The authors in [175] show that the graph edit distance can be computed by first performing seriation of the graph to a string and then computing the well-known string edit distance for the problem of graph matching. A kernel function was developed by [149] based on the edit distance for classifying a collection of graphs. They use the nearest neighbor classifier to classify a collection of graphs. They report improved classification performance over support-vector machines through the use of the new kernel function.

A spectral-based clustering method is proposed in [211] for clustering a collection of trees. They embed the trees in a Euclidean space using the ISOMAP algorithm in order to compute distance and determine the Laplacian matrix. They report improved performance over traditional spectral approaches. A graph matching and clustering approach is introduced in [209] based on spectral methods, where the authors show that the spectral decomposition of the Laplacian matrix can be used to construct a feature space for clustering using a variety of different methods including principle component analysis (PCA), multi-dimensional scaling (MDS), and locality preserving projection (LPP). They evaluate their method on several data sets including shock trees and object recognition in images and report their method performs well for clustering graphs [209]. Due to the intrinsic properties of variability models from software engineering, we require a new approach to clustering a collection of graphs that differs from any previous approaches in the literature.

Several approaches have been presented as well in a similar field on the discovering of frequent sub-graph structures, cf. [93][118][228]. However, due to the intrinsic properties of variability models coming from software engineering that are not standard to traditional graphs, we require a new approach to clustering a collection of graphs that differs from any of the previous approaches in the literature.

The primary objective of this study is to develop a framework for partitioning a set of graphs using relational clustering. A secondary objective is to evaluate and compare two relational clustering approaches: the new proximity fuzzy clustering algorithm and relational FCM. We demonstrate the feasibility of relational clustering to the problem of clustering a set of graphs. We evaluate the proposed approach in the domain of software engineering requirements analysis, which aims to determine common design components in various software architectures. However, due to the additional inherent semantics in nodes of the graphs in software engineering requirements analysis, we propose a new measure of proximity between graphs that accounts for both the structure and semantics of the nodes. Thus, we also evaluate the newly developed graph clustering framework in the context of software engineering.

To give a more complete intuition of the problem, consider the collection of graphs in Figure 38. These graphs are abstract tree examples. When looking for a partitioning of this collection of directed unweighted acyclic graphs, the trees that structure their nodes in a similar manner should be grouped together. The distance between the trees in Figure 38 can be easily measured using, for example, the edit distance which is the minimum number of “add” and “remove” operations of both nodes and links that are required to transform one tree into the other.

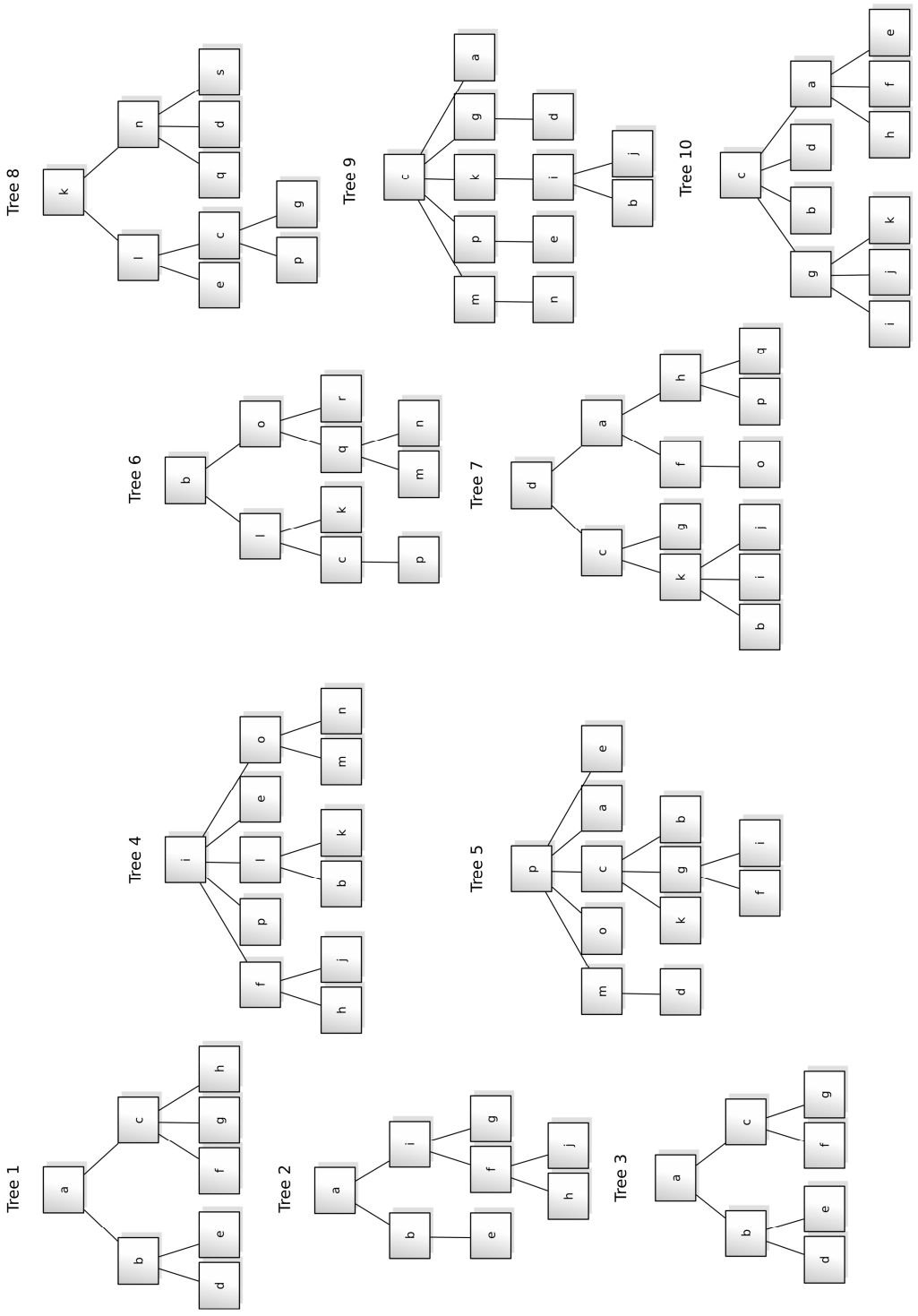


Figure 38. A collection of trees

Using the grey dotted lines, Figure 38 indicates the grouping of these trees by an expert based on how the tree structure the nodes *a, b, c, e, k, l*. It can be seen that trees 1, 2 and 3 are clustered

together as they are structurally very similar with respect to these nodes: all have node *a* as root, node *b* as child of node *a* and node *e* as child of node *b*. Note that these trees are also quite similar in the way they handle nodes *k* and *l*, they are absent in all of the trees. Using similar arguments, the expert has also grouped trees 6, 8 and 9 together: all trees have node *c* as root, nodes *a* and *l* as children of that root and node *e* as child of node *l*.

Given that the grouping of a collection of graphs based on their proximity is a very labor intensive and error-prone task, it is the focus of this chapter to determine the ability of clustering techniques to perform the grouping of a collection of graphs based on their proximity. In particular, we focus on the application of proximity-based and relational clustering for this purpose.

The novelty presented in this research is a two-fold focus. We introduce a new proximity fuzzy clustering algorithm and the known relational FCM. The proximity fuzzy clustering algorithm is evaluated alongside the relational FCM for comparison. We also introduce a new proximity measure based on the edit distance which is well suited for measuring the proximity between variability models in software engineering. A formal definition of variability models and its measure of proximity is given. An important novelty introduced is that the proposed measure of proximity includes both structural and semantic similarity between graphs. Previous approaches mostly focus on only structural similarity. The problem is applied and evaluated in the context of software engineering.

The notation used follows conventional fuzzy clustering notation where our data set X is a set of graphs G_k such that $X=\{G_k|k=1\ldots N\}$ where N is the number of graphs in the data set. The edit distance between two graphs $G_1,G_2\in X$ is given by $d(G_1,G_2)$ while the proximity is denoted by the function $p(G_1,G_2)$. The output of the clustering is the partition matrix $U=[u_{ik}]$ for $i=1\ldots c$, and $k=1\ldots N$ where c is the number of clusters. The prototypes are denoted by $V=\{\mathbf{v}_i|i=1\ldots c\}$. The cluster proximity values produced by proximity fuzzy clustering is expression by the matrix $Q=[q_{ik}]$ for $i=1\ldots c$, and $k=1\ldots N$. Graphs are indicated by capital letters and nodes are indicated by lower-case. We define the Euclidean distance to be $||\mathbf{x}||$ for any real-valued vector $\mathbf{x}$ and we define the cardinality using the notation $N=|X|$. The Boolean operators $\wedge$ and $\vee$ are AND and OR respectively. We employ the conventional notation for set operators.

9.2. Background – Edit Distance of Graphs

In the problem statement, we have outlined our focus to assess the applicability of various clustering approaches for the problem of grouping graphs based on how similar they are with respect the nodes they contain and how they are structured, e.g. how they are connected by edges. One of the critical components that is required before the grouping of graphs can be performed is the possibility to quantify the proximity of graphs. In this section, we outline the base concepts of the most well-known technique for such purposes, the *edit distance* of graphs.

Within the field of graph matching, the concept of a partial match between two graphs exists. This refers to the situation where one graph can only be partially matched to the other, because specific nodes, edges or properties do not exist in both thereby preventing a complete or perfect match. To quantify the quality of such a partial match, the *edit distance* [121] is used, which initially was computed by counting the minimal amount of re-labeling of nodes and edges together with the number of deletions and insertions of nodes and edges to transform a graph into another. A variety of edit distance definitions specializing this initial definition can now be found that address incorporate specific properties of graphs into their calculation, such as graph topology and the attributes of attributed graphs [62].

To give an intuitive illustration of how the edit distance of graphs is calculated, consider the three trees depicted in Figure 39. These two trees share a number of identical nodes and edges, but

there are differences in how the nodes are structured. Considering a simple edit distance calculation that consists solely of "delete" and "insert" operations of edges and nodes, the edit distance is the following.

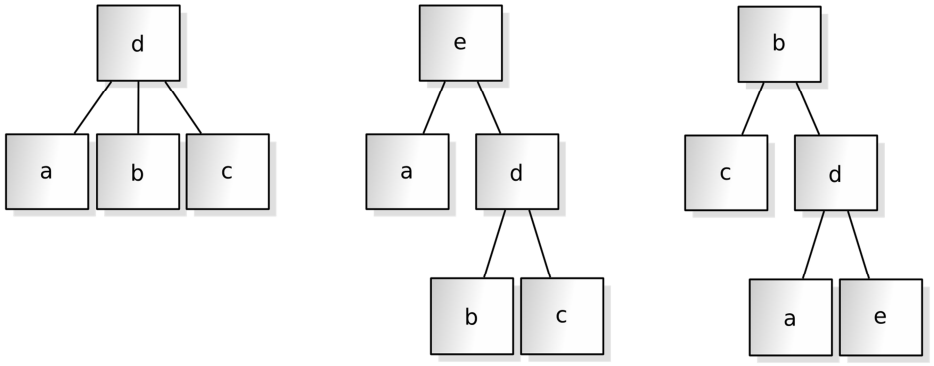


Figure 39. Example trees

First, we determine the edges and nodes of the leftmost tree that need to be inserted for it to be transformed in the rightmost tree. It can be seen that a node labeled "e" needs to be introduced, as well as an edge between this new node labeled "e" and the node labeled "a". This makes a total of two insert operations.

Second, we determine the nodes and edges of the leftmost tree that need to be inserted for it to be transformed in the rightmost tree. It can be seen that only the edge between the node labeled "d" and the node labeled "a" needs to be removed. This make a total of one delete operation.

The edit distance from the leftmost tree to the rightmost tree in Figure 39 therefore is equal to three operations. This value can be normalized by dividing it by the maximum edit distance theoretically possibly for these two trees. This corresponds to deleting all edges and nodes from the leftmost tree (seven delete operations) and inserting all the nodes and edges of the rightmost tree (nine insert operations), equating to a maximum edit distance of sixteen operations. The normalized edit distance then is equal to 4/16 or 0.25. Calculated in similar fashion, the edit distance between the left and right tree is 6/16 or 0.375, and between the middle and the right tree is 8/18 or 0.44.

9.3. RELATIONAL CLUSTERING OF GRAPHS

Clustering of relational data is a growing area in the literature especially when clustering objects that lack meaning in a *d*-dimensional real-valued vector space, i.e. $\mathbb{R}^d$. Methods such as hierarchical clustering, relational FCM [86][85], and spectral clustering [58][138] are common tools in clustering relational data. Spectral clustering is a newer relational clustering algorithm that utilizes the graph cut approach to cluster relational data [58][138]. The relational data is given as an affinity (or adjacency) matrix A, where the values indicate the level of similarity between objects in the data set. Spectral clustering then computes a Laplacian matrix [58] using the affinity matrix A. There are various types of Laplacian matrices that can be constructing including the unnormalized, and normalized versions [58][138]. Spectral clustering then extracts the "*c*" largest eigenvalues and corresponding eigenvectors where "*c*" is the number of clusters. The eigenvectors are arranged in columns and concatenated to form the *M*×*c* matrix which is then clustered using a standard cluster algorithm such as k-means, or FCM by treating all the rows of the matrix as data points [138]. A serious disadvantage of spectral clustering is that is scales poorly with large data sets and large number of clusters.

Relational Fuzzy C-Means (relational FCM) was introduced several decades ago that is computationally efficient and able to produce a fuzzy partition when provided with relational data rather vectorial data as with traditional clustering algorithms, cf. [86][85]. Relational FCM accepts a matrix of relationships as inputs denoted by $[r_{jk}]$ for j,k=1...N. This matrix is computed from the graphs according the expression

$$r_{jk} = \frac{d\left(G_j, G_k\right)}{\max_{\forall i,h=1\ldots N}\left(d\left(G_i, G_h\right)\right)} \tag{9-1}$$

for j,k=1...N where the distance function $d()$ which is based on the edit distance will be elaborated in Section 12.4.

Relational FCM is shown to optimize the same objective function as standard FCM, cf. [85]. There are two concerns with using relational FCM:

- The feature space is assumed Euclidean despite the fact that this may indeed not be true of all relational data, cf. [85]
- The algorithm inherits the same drawbacks from FCM including the tendency to form balanced clusters, i.e. roughly equal cluster sizes, even when the data set is highly unbalanced.

When clustering feature trees, our problem involves clustering unbalanced data using a set of relations that describe the similarity or proximity of each pair of graphs. The proximity between graphs $G_1,G_2 \in X$ is formalized as a function $p(G_1,G_2)$ that maps to the unit interval and satisfies the properties:

$$p\left(G_1, G_2\right) = p\left(G_2, G_1\right) \tag{9-2}$$

$$p\left(G_1, G_1\right) = 1$$

When $p(G_1,G_2)$=0, graphs G_1 and G_2 are entirely dissimilar and when $p(G_1,G_2)$=1 they are identical. The proximity between graphs can also be interpreted as a fuzzy relation denoting the degree of similarity between two the graphs G_1 and G_2. Proximity fuzzy clustering requires a matrix of proximity values which are computed from the proximity function with the expression

$$p\left(G_j, G_k\right) = 1 - \frac{d\left(G_j, G_k\right)}{\max_{\forall i,h=1\ldots N}\left(d\left(G_i, G_h\right)\right)} \tag{9-3}$$

for j,k=1...N. The proximity data is then clustered using proximity fuzzy clustering

9.3.1. Determining the number of clusters

An open problem is the choice of the number of clusters "c" and the value of the fuzzification coefficient "m." Much work has been done in this area, however the approaches are highly varied in both design and performance. However, proximity fuzzy clustering offers a simple and effective (unsupervised) performance index using the proximity relations directly. The number of clusters and the fuzzification coefficient can be determined by minimizing the reconstruction error given by

$$r_{error} = \sum_{k=1}^{N}\sum_{j=1}^{N}\left(p\left(G_j, G_k\right) - \tilde{p}_{jk}\right)^2 \quad \textbf{(9-4)}$$

where the a reconstructed proximity based on membership values is given by $\tilde{p}_{jk} = \sum_{i=1}^{c} min\left(u_{ij}, u_{ik}\right)$, cf [165]. In order to improve the robustness of the selected number of clusters and fuzzification coefficient, we look for a set of parameters that satisfies the following conditions:

- No two cluster proximity vectors are the same, i.e. $q_{ij} \neq q_{lj}$ for $i,l=1\ldots c, i \neq l$
- No cluster is "empty" where the ith cluster is empty iff $\forall k = 1\ldots N, i \neq \arg\max_{\forall j=1\ldots c} u_{jk}$

9.3.2. EXAMPLE

We demonstrate both proximity and relational FCM on the synthetic example given in Figure 38 and compare to relational FCM. The proximity values are given in Table 33. These values are computed using the edit distance described in Section 12.2 on the example given in Figure 38.

Table 33. Proximity values of the synthetic graph data set given in Figure 38

Trees	1	2	3	4	5	6	7	8	9	10
1	1	0.83	1	0.4	0.57	0.39	0.46	0.29	0.64	0.67
2	-	1	0.85	0.46	0.36	0.27	0.36	0.17	0.5	0.46
3	-	-	1	0.31	0.54	0.39	0.46	0.29	0.64	0.53
4	-	-	-	1	0.42	0.71	0.33	0.46	0.46	0.43
5	-	-	-	-	1	0.5	0.77	0.43	0.86	0.8
6	-	-	-	-	-	1	0.46	0.57	0.43	0.44
7	-	-	-	-	-	-	1	0.31	0.77	0.57
8	-	-	-	-	-	-	-	1	0.5	0.4
9	-	-	-	-	-	-	-	-	1	0.8
10	-	-	-	-	-	-	-	-	-	1

Proximity fuzzy clustering detects the number of clusters to be c=3 and the fuzzification coefficient to be m=2.6 under the same optimization vehicle discussed in the previous section. The structure discovered by proximity fuzzy clustering is given by the sets {Tree 1, Tree 2, Tree 3}, {Tree 4, Tree 6, Tree 8}, and {Tree 5, Tree 7, Tree 9, Tree 10}. The membership values for Trees 8 and 10 are quite low in their clusters. Figure 40 demonstrates how the structure of the clusters changes with the number of clusters (m is held at m=2.9) for proximity fuzzy clustering. The structure shown in Figure 40 is determined by labeling the graph G_k according to its maximum membership, i.e. $cluster - label\left(G_k\right) = \arg\max_{\forall i=1\ldots c} u_{ik}$ for k=1…N.

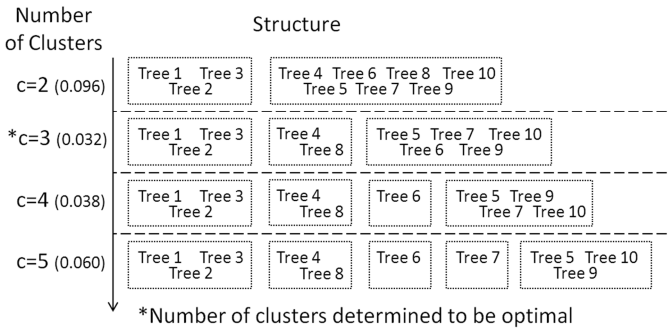


Figure 40. Synthetic example: number of clusters

There is an interesting trend shown in Figure 40, particularly, the consistency of the first cluster. In fact, the membership values of Tree 1, Tree 2 and Tree 3 and all close to 1 which is intuitive since the graphs are structurally quite similar. The membership of Trees 5 and 9 are fairly high as well. Otherwise the membership values are much lower. The structure of the clusters is the same with relational FCM. The membership value of Tree 8 however is very low thus it does not belong strongly to any cluster. Figure 41 shows concisely the membership values for *c*=3 for both proximity and relational FCM approaches.

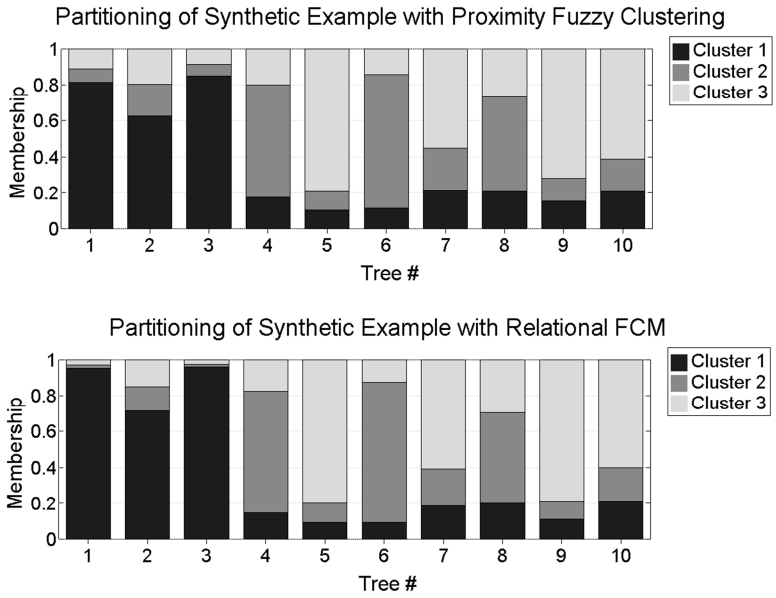


Figure 41. Partition for synthetic example (*c*=3)

The plots given in Figure 41 indicate the membership values on a floating bar plot. The horizontal axis consists of the Trees 1-10 from the problem where the vertical bars indicate the membership. The membership values of the clusters are stacked in a bar graph to visually indicate the degree of membership for that particular tree in each cluster. Each cluster is coded via, black, gray and light gray. There is no scaling of the membership values done since the membership values must add up to one (due to the constraint on U). For example, we can see that the membership of Tree 1, Tree 2 and Tree 3 are quite high (close to 1) for cluster 1. This is intuitive since Figure 38 shows that these trees have a lot of structural similarity. The fuzzy membership values are mostly the same for both proximity and relational FCM when *c*=3.

The objective function value calculated at each iteration is shown in Figure 42 for proximity and relational FCM.

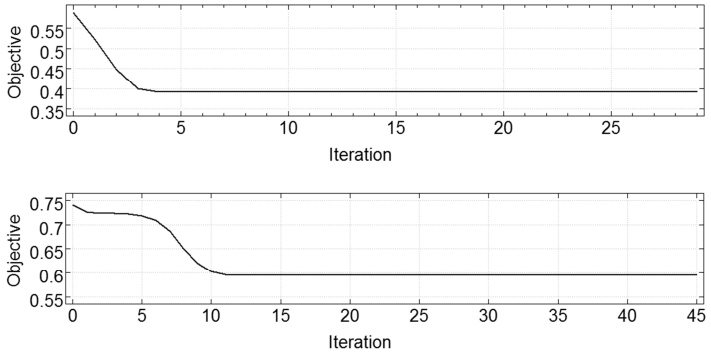


Figure 42. Objective function value at each iteration

Relational FCM appears to have converged slightly faster than proximity fuzzy clustering on this data set.

In summary, we have shown that both proximity fuzzy clustering and relational FCM are suitable tools for clustering graphs. Particularly, the structure formed by the clusters indicates the graphs that are most similar, ie. Tree 1, Tree 2 and Tree 3. The other graphs are much more distinct from each other and as a result their membership values are lower.

9.4. GRAPH CLUSTERING APPLIED TO SOFTWARE REQUIREMENTS ENGINEERING

9.4.1. MOTIVATION: IDENTIFYING BEST PRACTICES IN A KNOWLEDGE BASE OF VARIABILITY MODELS

The motivation for addressing the problem of graph clustering in non-Euclidean spaces originates from the current trend in software development to examine the design of software systems that are similar to the one under development to identify best practices and standard structures. This is done by mining a large body of software designs, typically graph-based models, for frequently occurring patterns describing a specific attribute of a software system. The underlying assumption for extracting such advice is that the more frequently a particular structure occurs, the more relevant this way of structuring is (intuitively, if many different software engineering experts use the same structure, it probably is a good way of doing it).

Here, we focus on this problem in the context of software product lines (SPL), a discipline of software engineering aimed at efficiently developing families of software systems. In this discipline, the *feature or variability model* [101] is used to describe the common and variable components of the various software systems that are support by the SPL. In SPL development, feature models are a specific type of variability model and other models that capture variability exist. To avoid confusion whenever we mention variability model, we refer to a feature model as it is used in SPL development.

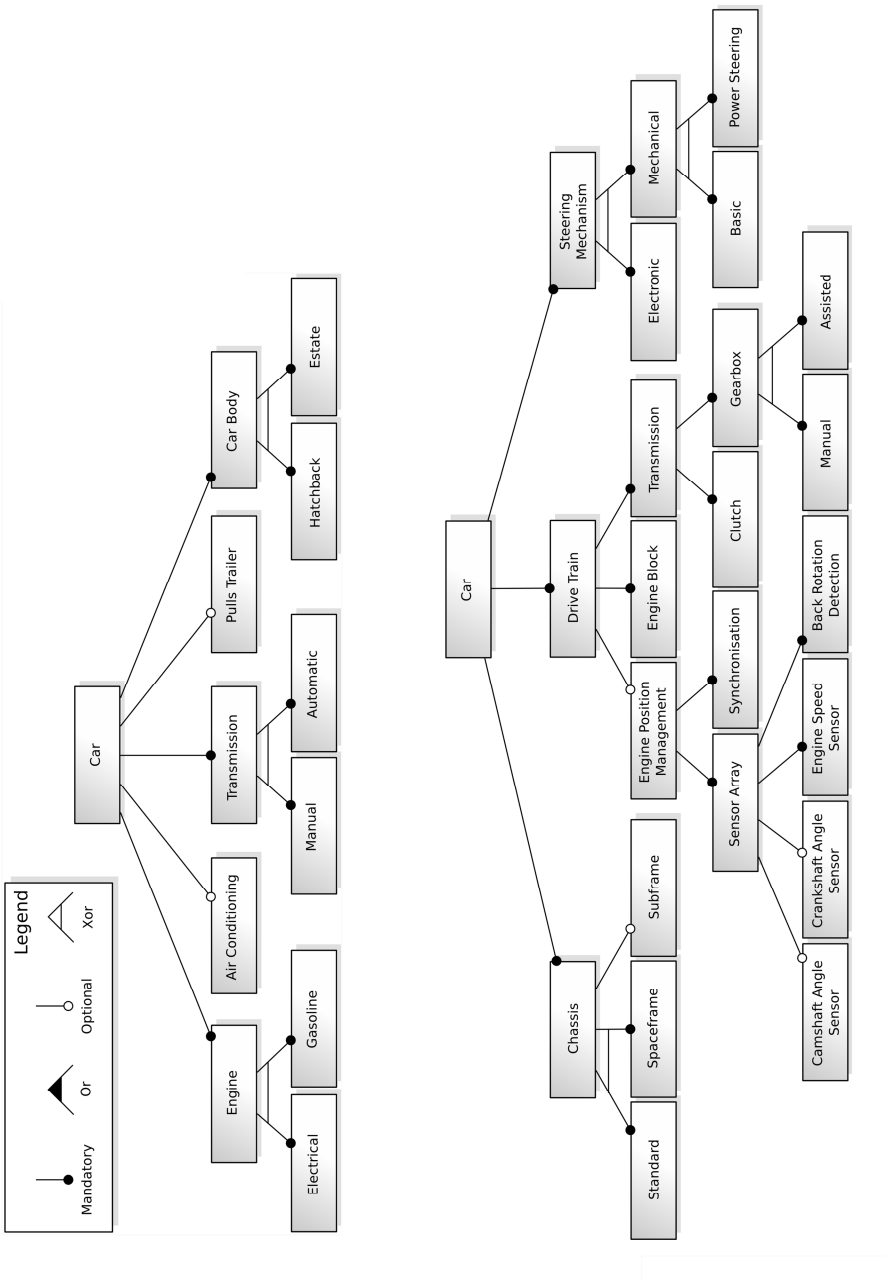


Figure 43. Variability Models of Automotive Software Product Lines

Variability models are used to describe a family of products in a single diagram, effectively defining logical constraints on the combination of properties that are allowed in a valid product. Consider for example the left-most variability model depicted in Figure 43. This model describes a family of cars and the constraints a car should adhere to. For example, there is an exclusive or-relationship between *Automatic* and *Manual*, indicating that a car can have only one of these types of transmission. Another example is *Air Conditioning* that is marked as optional, meaning that any car of this family can be fitted with air conditioning, but it is not mandatory.

To illustrate to problem of graph clustering in the context of software requirements engineering, consider the following example of software product lines for the automotive industry. In Figure 43, four possible variability models for automotive Software Product Lines are depicted.

Each of these models describes a set of automotive products and uses a distinct decomposition of concepts for this purpose. For example, the top-left model decomposes a car into five base concepts of which *Air Conditioning* and the ability to *Pull Trailers* is optional. It also prescribes that a car can only have either a manual or an automatic transmission. The top-right variability model on the other hand describes a more detailed decomposition of the car concept and its products are aimed at the high-performance/racing segment. As a result, this model structures nodes significantly different, with for example *Engine* now being a sub-concept of *Drive Train*.

When an SPL development team needs to design a new automotive SPL, the decomposition knowledge in these variability models can be used to structure the variability model of the new SPL. For example, consider the case where the development team is interested in how the concepts *Car*, *Engine*, *Transmission* and *Air Conditioning* should be structured with respect to each other. These concepts form the criterion for examining the knowledge base formed by the models in Figure 43.

By determining the most frequently occurring manner of structuring the criterion, the best practice can be identified. It can be seen that the two left-most models structure the *Engine* and *Transmission* concepts directly underneath the *Car* concept, whereas the two right-most models structure these concepts underneath a separate *Drive Train* concept. Using a function to describe the proximity of these models with respect to this criterion, a matrix can be derived that can be used to cluster the variability models.

9.4.2. DEFINITIONS

We define a variability model to be a labeled, directed, acyclic graph whose nodes correspond to (sub-)concepts and whose edges correspond to relations between these (sub-)concepts. Moreover, we extend this graph definition to contain the specific properties of variability models. We define each node to have a property indicating whether it is mandatory or optional. Also, a set of nodes in the graph can be subject to an Xor-constraint or an Or constraint. We use the following set-based notation for a variability model $T : (Nodes, CP, Oblig, Xor, Or)$.

In this specification, *Nodes* is the set of nodes, *CP* is a set of tuples (*a*,*b*) describing that node *a* in *Nodes* is a child of node *b* in *Nodes*. *Oblig* is the subset of nodes of *Nodes* that are mandatory. *Xor* is a set of sets of nodes that are mutually exclusive with respect to each other and *Or* is a set of sets of nodes that are optional with respect to each other. We assume that a node contains a textual description of the (sub-)concept of the SPL it represents. Moreover, we define the following:

1. **Knowledge base:** A set of variability models
2. **Criterion:** A set of node descriptions based on which semantics and structuring variability models are compared.
3. **Node similarity:** A value on the domain [0..1] indicating the degree to which the sub-concepts represented by two nodes correspond. This value is determined by a proximity function that analyses the textual descriptions of the nodes.
4. **Tree similarity:** A value on the domain [0..1] indicating the degree to which two trees are similar with respect to a given criterion. This value is determined by the proximity function that will be introduced in Section 12.4.3.

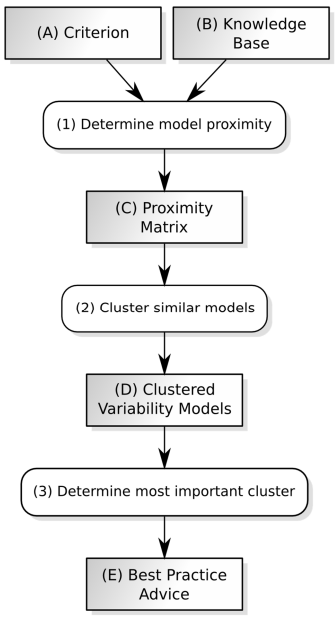


Figure 44. Process of identifying best practices in software requirements engineering

Given these definitions, the schematic representation of the steps for deriving best practices from a knowledge base of variability models is depicted in Figure 44. In this figure, a criterion (A) and knowledge base (B) are provided. The proximity of each pair of trees in the knowledge is then determined (1) to form a proximity matrix (C). Based on this proximity matrix, the variability models are clustered (2). The clustered models (D) are subsequently analyzed (3) to form the best practice advice (E). Note that we refer to *best practice* as the most common practice in use by software engineering experts.

Activity (1) involves calculating a proximity measure using the function that will be introduced in Section 12.4.3. Activity (3) on the other hand involves ranking the clustered that have been derived and ranking them according to the relevance of the best practice advice they provide. Although the actual realization of this step is beyond the scope of this thesis, this could for example be done by choosing the cluster with the largest cardinality, which corresponds to the fact that the pattern in this cluster is the most frequent in the knowledge base and therefore the most relevant. Here, we are primarily concerned with the second step (2) of this process, the clustering of variability models based on their proximity degree.

9.4.3. SPECIFICATION OF THE PROXIMITY FUNCTION FOR VARIABILITY MODELS

Variability models as they are used in Software Product Line development are not standard graphs; they define additional properties that are not found on traditional tree structures, such as the optionality of a node or its mutual exclusivity with one or more of its neighbors. As a result of these new properties, traditional definitions for the edit distance of tree are insufficiently accurate. To perform grouping of a collection of variability models therefore requires the definition of a new edit distance measure. In this section, we define this refined edit distance and its derived proximity measure and we demonstrate its usage on the automotive example introduced in Section 12.4.1.

We build up the new proximity measure in two stages. First, we introduce the edit distance measure that takes into account the structural similarity of the variability models being compared. After that, we extend this with the possibility to correct for partial semantic matches between the concepts represented by the nodes in the variability models.

9.4.3.1. STRUCTURAL PROXIMITY

One of the most important aspects from the software engineering point-of-view is that a developer is typically interested in grouping of a collection a variability models with respect to a small recurring pattern, i.e. the proximity generally should not be calculated for the entire graph but rather for a sub-graph

We now define the specialized proximity function for comparing variability models used in Software Product Line Development that is inspired on traditional edit distance concepts. Let *T* and *W* belong to X_{VM} which is a collection of variability models. We then define *T* and *W* to be *identical* with respect to criterion *S* when

a) All the nodes in *S* exist in both *T* and *W*
b) All the nodes in *S* have the same parents in *T* and *W*
c) All the nodes in *S* have the same logical role in *T* and *W*
d) All the nodes in *S* have the same siblings that are part of *S* in *T* and *W*

We consider *T* and *W* to be *similar* when these conditions are only partially fulfilled. Note that this definition explicitly considers properties that are specific to the semantic meaning of variability models, such as the logical role of nodes (optional, mandatory) and whether nodes are siblings (meaning they belong to the same concept, such as a transmission). Also note that a proximity definition for the structure of variability models has not been proposed before, hence all subsequent definitions original concepts.

We define the function $p_S()$ which determines the proximity of two variability models *T* and *W* with respect to criterion *S* using the amount of operations that are required to *T* and *W* equal. In our proximity function, we differentiate between four types of operations: operations on nodes, edges, logical roles of nodes and siblings of nodes. The calculation of the amount of these operations is detailed below. We will illustrate each step of our definitions by applying it to the variability models depicted in Figure 45 for *S* = {*a*, *b*, *c*, *d*, e, *f*}. The set based representation of these models is as given in Table 34.

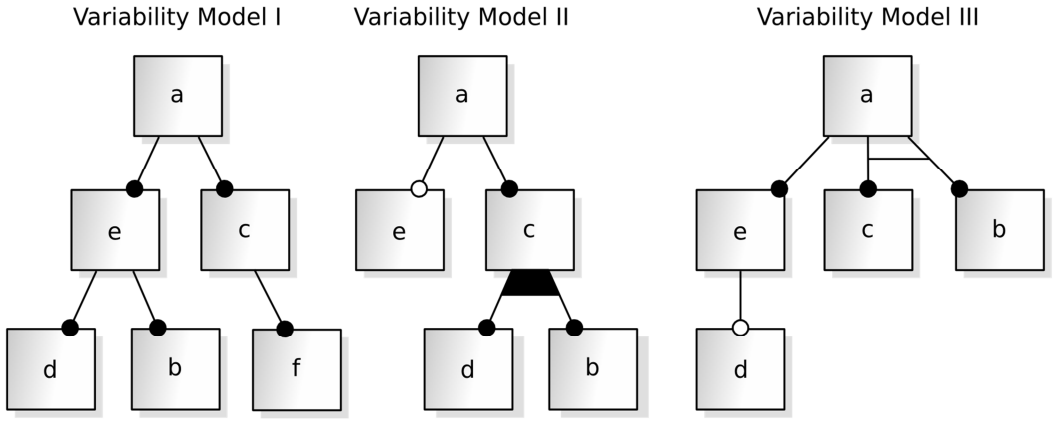


Figure 45. Variability Models for Sibling Similarity

Table 34. Set-based Representations of Variability Models

	Model I	Model II	Model III
Nodes	{ *a,b,c,d,e,f* }	{ *a,b,c,d,e* }	{ *a,b,c,d,e* }
CP	{ (e,a),(d,e),(b,e),(c,a),(f,c) }	{ (e,a),(d,c),(b,c),(c,a) }	{ (e,a),(d,e),b,a),(c,a) }
Oblig	{ a,b,c,d,e,f }	{ a,c }	{ a,e }
Or	{}	{}	{ { c,b } }
Xor	{}	{ {d,b} }	{}

Edit Operations for Nodes

The set *E* contains all the nodes of *S* that are part of either *T* or *W* but not both or neither.

$$E = \left(\left(Nodes_T \bigcup Nodes_W\right) \bigcap S\right) \setminus \left(\left(Nodes_T \bigcap Nodes_W\right) \bigcap S\right) \quad \textbf{(9-5)}$$

This set corresponds to the notion of "insert" and "delete" operations for nodes in traditional edit distance definitions. *E* in this case corresponds to the nodes that need to be addressed by such operations. Determining *E* for the model I and model II of Figure 45 results in: $E = \{a,b,c,d,f\} \setminus \{a,b,c,d\} = \{f\}$.

The resulting set corresponds to the fact that *f* is the only node of the criterion that is present in model I but not in model II. Alternatively, *E* for models II and III is empty as there is no node in the criterion that is present in only one tree: $E = \{a,b,c,d\} \setminus \{a,b,c,d\} = \{\}$.

Edit Operations for Edges

The set *P* that contains all nodes of *S* that occur in both models but have different parents in *T* and *W*:

$$P = \left(Nodes_T \bigcap Nodes_W\right) \bigcap S \setminus \left\{ \begin{array}{c} x \in \left(Nodes_T \bigcap Nodes_W\right) \bigcap S \\ such\,that\, \exists y \in \left(Nodes_T \bigcap Nodes_W\right) \\ where\left(x,y\right) \in \left(CP_T \bigcap CP_W\right) \end{array} \right\} \quad \textbf{(9-6)}$$

This set corresponds to the notion of "insert" and "delete" operations for edges in traditional edit distance definitions. *P* in this case corresponds to the edges that need to be addressed by such operations. When we determine *P* for the left and middle model in Figure 45, this results in: $P = \{a,b,c,d\} \setminus \{a,c,e\} = \{b,d\}$.

It can be seen that the nodes *a*, *c* and *e* are removed from the initial set of criterion nodes present in models I and II as they have the same parents in both models. The nodes *b* and *d* remain as they do have different parents. For models II and II, *P* becomes the following: $P = \{a,b,c,d\} \setminus \{a,c,e\} = \{b,d\}$.

Edit Operations for Logical Roles of Nodes

Third, the set *R* contains all nodes of *S* that have a different logical role in *T* or *W*:

$$R = \left(Nodes_T \cap Nodes_W\right) \cap S \setminus \left(Oblig_T \cap Oblig_W\right) \cup \left(Xor_T \cap Xor_W\right) \cup \left(Or_T \cap Or_W\right) \cap S \quad \textbf{(9-7)}$$

This set introduces a new concept in the edit distance definition corresponding to whether a node has a different logical role in the variability models (for example, it is mandatory in one model and optional in another). *R* in this case corresponds to the nodes whose logical roles need to be addressed by "insert" and "delete" operations. When we determine *R* for models I and II, this results in: $R = \{a,b,c,d\} \setminus (\{a,c\} + \{\} + \{\}) = \{b,d\}$.

Here, it can be seen that three sets of nodes are removed from the initial set of criterion nodes that occur in both I and II. First, the nodes *a* and *c* are removed as they are mandatory in both trees. Subsequently, the nodes that are part of Xor and then Or relations in both trees are removed (both these sets are empty). What remains is *b* and *d* as they are mandatory in I and Xor in II. When this is done for models II and III, this results in: $R = \{a,b,c,d\} \setminus (\{a\} + \{\} + \{\}) = \{b,c,d\}$.

Edit Operations for Sibling Degree

Fourth, the similarity of how the criterion nodes are structured as siblings in both trees is determined. This particular element of proximity relates to the specific semantics of the child parent relation of nodes in variability models. When node *a* is a child of node *b* in the context of a variability model, this means that node *a* is a sub-concept or specialization of node *b*. For example, in the models of Figure 43 it can be seen that an *Engine* is a sub-concept of *Car* and an *Automatic Transmission* is a specialization of *Transmission*.

As a result of the semantics of the parent child relation, having the same sibling in both models increases proximity as this implies the sibling nodes are related to the same parent concept in both models. For example, consider the three variability models depicted in Figure 45. According to traditional edit distance definitions, the left and right models are closer than the models on the left and in the middle. This as in the left and the right model, only node *e* has a different parent. However, due to their specific semantics the models on the left and in the middle are considered closer as nodes *d* and *e* are sub-concepts of the same parent in both, even when these parents do not correspond.

To accommodate for this, we enrich the proximity definition with the concept of *Sibling Similarity*. This corresponds to the degree in which a node has the same sibling in the variability models that are being matched. For the calculation of the sibling similarity, we assume the following two functions exist:

- DR_T (*a*): returns the distance of node *a* to the root in variability model *T*
- DCA_T (*a, b*): returns the distance to the first common ancestor of nodes *a* and *b* in variability model *T* measured from the node that is farthest from the root of *T*
- A_T (*a, b*): returns a Boolean indicating whether node *a* is an ancestor of node *b* in tree *T*

Using these functions, we define $SibDeg_T$ (*a, b*) to be the degree to which nodes *a* and *b* are considered siblings in tree *T*:

$$SibDeg_T(a,b) = \begin{cases} 0 & if\ A_T(a,b) \vee A_T(b,a) \\ 1 & if\ DR(a) = DR(b) = 1 \\ \dfrac{max(DR(a), DR(b) - DCA_T(a,b))}{max(DR(a), DR(b)) - 1} & otherwise \end{cases} \quad \textbf{(9-8)}$$

The intuition behind this function is that two nodes are siblings to a certain degree when they have a common ancestor in the variability model. The result of this function is equal to 0 when one node actually is an ancestor of the other and it is equal to 1 if they are proper siblings (have the same parent). It is a value between 0 and 1 when the common ancestor is not their direct parent but still below the root of the variability model. This corresponds to both belonging to a broader concept which is more specialized than the overall concept. For example, in the automotive models of Figure 43 the *Fuel Tank & Distribution* node and the *Diesel Engine* node are not proper siblings, but according to the result from expression (**9-8**) they are considered siblings to a degree 0.5 as their common ancestor is very close-by.

We then define the set $SS^a_{T\text{-}W}$ which contains all the siblings of node *a* in *T* and *W*:

$$SS^a_{T-W} = \left\{ x \in \left(Nodes_T \bigcup Nodes_W \right) \mid x \neq a \wedge \left(pt_T(x) = pt_T(a) \vee pt_T(x) = pt_W(a) \right) \right\} \quad \textbf{(9-9)}$$

Using this set, we can define the function $SameSibDeg_a(T,W)$ which returns a value on the domain [0..1] indicating the degree to which the node *a* has the same siblings in variability models *T* and *W*:

$$SameSibDeg_a(T,W) = \frac{\sum_{x \in SS^a_{T-W}} min\left(SibDeg_T(a,x), SibDeg_W(a,x) \right)}{\left| SS^a_{T-W} \right|} \quad \textbf{(9-10)}$$

Consequently, the fuzzy set *C*(*x*) where *x*∈*S* contains all the nodes of the criterion *S* that occur in both *T* and *W* to degree to which they have different siblings in both trees:

$$C(x) = \begin{cases} 1 - SameSibDeg_x(T,W) & if\ x \in S \setminus E \\ 0 & otherwise \end{cases} \quad \textbf{(9-11)}$$

This set corresponds to the notion of operations needed to ensure nodes have the same siblings in both models. This is weighted with respect to the distance the siblings are apart in both trees; the further they are apart the more severe the edit required. *C* in this case corresponds to the nodes that need to be addressed by such operations.

en applied to models I and II of Figure 45, nodes *a*, *b*, *c*, and *d* need to be considered for sibling similarity. We illustrate how the membership value of *b* in *C* is calculated using the definitions above. First the sibling set $SS^b_{I\text{-}II}$ is determined, which corresponds to the siblings of *b* in either I or II: $SS^b_{I\text{-}II} = \{d\}$.

Next, the sibling degree of *b* and *d* is determined in trees 1 and 2 using formula (**9-8**): $SibDeg_I(b, d) = 1$ and $SibDeg_{II}(b, d) = 1$. Using formula (**9-10**), the membership value of *b* in *C* can be calculated: $C(b) = 1 - SameSibDeg_b(I,II) = 1 - 1 = 0$. This calculation can be repeated for the other elements of *C* to result in following set: *C* = { 0/*a*, 0/*b*, 0/*c*, 0/*d* }

Note that this set describes a special case where all the nodes have exactly the same siblings in both variability models. When the same procedure is applied for comparing models II and III, the set *C* becomes the following: *C* = { 0/a, 1/b, 1/c, 1/d }. In this case, nodes *b*, *c* and *d* have completely different siblings in both models, with their original siblings either being far away or even changing role (becoming parent instead of sibling). Consequently, their membership in *C* is equal to 1.

These four sets now enable us to define the amount of operations required as follows:

$$d_S(T,W) = w_E|E| + w_P|P| + w_R|R| + w_C\sum_{x\in S} C(x) \qquad \textbf{(9-12)}$$

where w_E, w_P, w_R and w_C are weightings for the severity of the modifications required for elements of the respective sets, which must adhere to $w_E \geq w_P + w_R + w_C$. The reason for this constraint is that all nodes of *S* that are element of *E* are nodes that are completely missing in one of the models being compared. As a result, all the operations are required to correct this (the node needs to be inserted, it has to be attached to the right parent, it has to be given the correct logical role and it has to be grouped with its correct siblings). As a result, being an element of *E* corresponds to being part of *P*, *R*, *C* as well as requiring an additional operation. This is reflected by ensuring w_E adheres to the constraint. When we take $w_E = 4$ and $w_P = w_R = w_C = 1$, using our earlier calculations we get the following: $d_{\{a,b,c,d,f\}}(I,II) = 4*1 + 1*2 + 1*2 + 1*0 = 8$ and $d_{\{a,b,c,d,f\}}(II,III) = 4*0 + 1*2 + 1*3 + 1*3 = 8$.

Finally, we define the proximity of *T* and *W* with respect to *S* to be:

$$p_S(T,W) = 1 - \frac{d_S(T,W)}{w_E|S|} \qquad \textbf{(9-13)}$$

When we apply this function to our previous calculations, we get the following proximity results: $p_{\{a,b,c,d,f\}}(I,II) = 1 - (8 / (4*5)) = 0.6$ and $p_{\{a,b,c,d,f\}}(II,III) = 1 - (8 / (4*5)) = 0.6$.

These results indicate that the proximity of I and II is equal to II and III. When examining Figure 45, this can be explained from node *f*. While models I and II appear very similar, they treat node *f* differently. In I it is explicitly included, while II omits it completely.

9.4.3.2. GRAPH CLUSTERING REVISITED

In Section 12.3, we grouped the trees depicted in Figure 38 based on their proximity for nodes *a*, *b*, *c*, *e*, *k*, *l*. Consider the collection of variability models depicted in Figure 46, which are extended versions of the trees depicted in Figure 38.

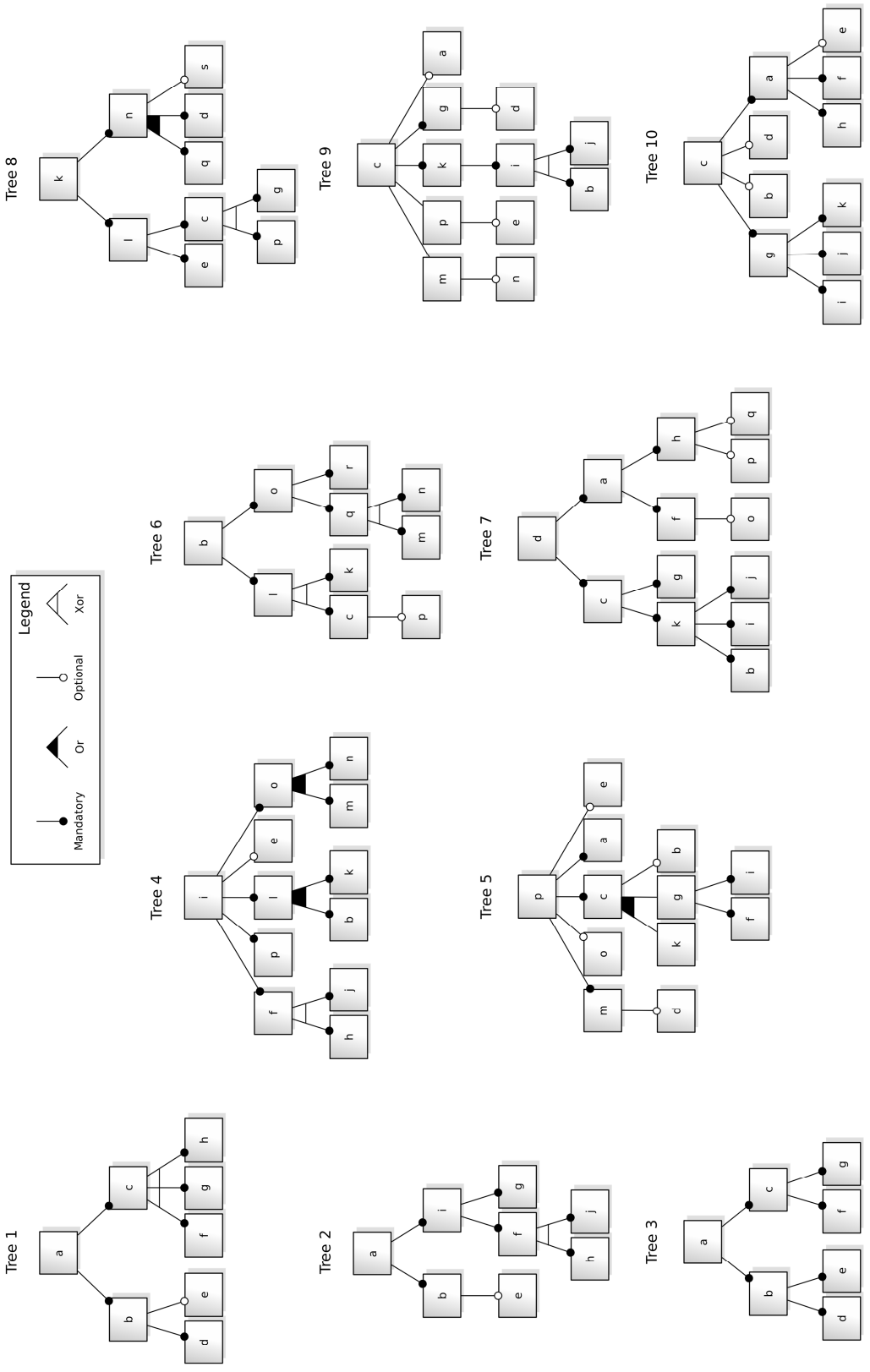


Figure 46. A Collection of Variability Models

The variability models in this collection have a number of additional properties as compared to the trees in Figure 38. The nodes have been attributed with properties such as being optional or part of an Xor-relation with other sibling nodes. If a traditional edit distance function is applied to determine the proximity of these models, these additional properties would be ignored as such a function is not capable of considering them. The results of clustering would be identical to the results described in Section 12.3.

However, the proximity function we have defined in the previous sub-section is capable of considering the additional properties introduced into variability models. To illustrate the use of this function, we will apply it as part of the clustering of the variability models depicted in Figure 46. We set the weights to w_P=w_R=w_C=1 and w_E=4 by default unless otherwise indicated in our experiments. First, we calculate the proximities of the variability models with respect to criterion *a*, *b*, *c*, *e*, *k*, *l*:

Table 35. Proximity values of the synthetic graph data set given in Figure 46

Trees	1	2	3	4	5	6	7	8	9	10
1	1	0.79	0.96	0.33	0.21	0.25	0.33	0.25	0.46	0.5
2	-	1	0.75	0.21	0.38	0.21	0.29	0.21	0.42	0.42
3	-	-	1	0.28	0.17	0.25	0.29	0.29	0.42	0.46
4	-	-	-	1	0.77	0.49	0.56	0.49	0.39	0.40
5	-	-	-	-	1	0.42	0.39	0.42	0.33	0.35
6	-	-	-	-	-	1	0.42	0.88	0.63	0.46
7	-	-	-	-	-	-	1	0.46	0.33	0.46
8	-	-	-	-	-	-	-	1	0.67	0.5
9	-	-	-	-	-	-	-	-	1	0.46
10	-	-	-	-	-	-	-	-	-	1

The proximity values given in Table 35 are quite a bit different from the proximity values given in Table 33 where the proximity values are lower when using the new measure of proximity. This is intuitive since the nodes are the same in both examples. The difference comes from the additional properties in the graphs that are specific to software engineering such as mandatory and optional nodes. These additional properties will always reduce the proximity when compared with the proximity values computed from the traditional edit distance in Table 35.

When performing proximity fuzzy clustering, the number of clusters detected is *c*=3 and the fuzzification coefficient is selected to be *m*=2.5 (both *c* and *m* are optimized simultaneously by minimizing r_{error} in Section 12.3.4). The structure discovered by proximity fuzzy clustering is given by the sets {Tree 1, Tree 2, Tree 3}, {Tree 4, Tree 5, Tree 7}, and {Tree 6, Tree 8, Tree 9, Tree 10}. Figure 47 demonstrates how the structure of the clusters changes with the number of clusters (*m* is held at *m*=2.5) for proximity fuzzy clustering.

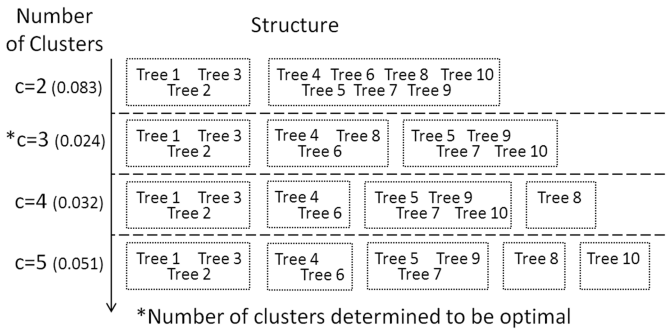


Figure 47. Synthetic example: number of clusters

As with the last synthetic example in 12.3.5, the first cluster consistently appears in the clustering regardless of the value of *c*. Tree 4, Tree 5, and Tree 7 also form a cluster with high membership values. According to the structure of the graphs given in Figure 46, Tree 10 is very different from all the other graphs where it is located in its own cluster by *c*=4. When considering relational FCM, the structure is nearly identical until c=5 where Tree 7 is taken out of the second cluster and placed in its own cluster and Tree 9 is retained in the third cluster. Figure 48 displays the membership values for *c*=3 for both proximity and relational FCM approaches.

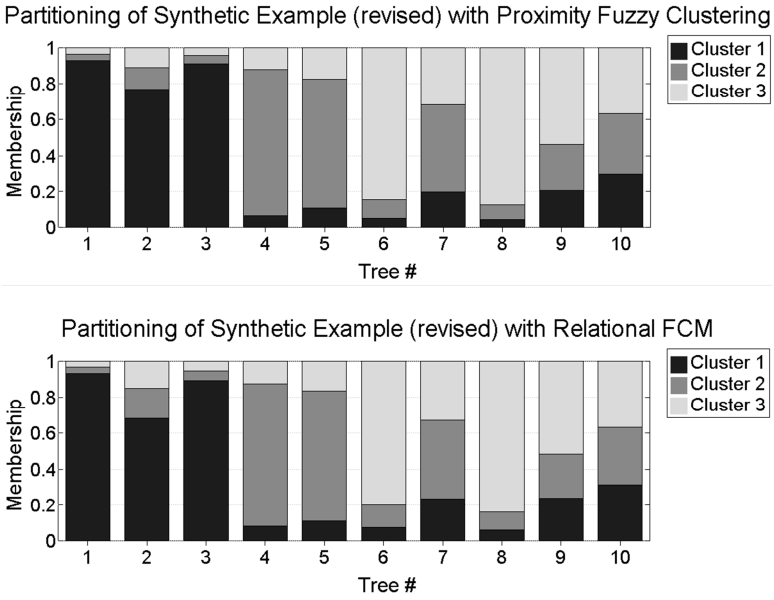


Figure 48. Partition for synthetic example (c=3)

The plots given in Figure 48 indicate the membership values on a floating bar plot. The horizontal axis consists of the Trees 1-10 from the problem where the vertical bars indicate the membership. The clusters are coded via black, gray and light gray. We can see that the membership values of Tree 1, Tree 2 and Tree 3 are quite high (close to 1) for cluster 1 and likewise the membership values of Trees 4 and 5 are fairly high in cluster 2 and the membership values of Trees 6 and 8 are fairly high in cluster 3. This is explained by the identical pair of siblings present in Tree 4 and Tree 5. Interestingly, Tree 10 has membership values that are roughly 1/*c* for both clustering techniques meaning that it is very different from all the other trees. The fuzzy membership values mostly the same for both proximity fuzzy clustering and relational FCM.

Changing the weights w_E, w_P, w_R, and w_C can have an effect on the proximity matrix depending on the structure and semantics of the graphs. If we set w_R=2 and leave all other weights at their default values then the differences in the logical roles of the nodes are emphasized. The number of clusters detected is c=2 and m=2.6 with the clusters being {Tree 1 – Tree 3} and {Tree 4 – Tree 10}. Increasing the weight has the effect of reducing the proximity values particularly among Trees 4 – 10, resulting in low membership values for those trees. Setting w_C=2 and leaving all other weights at their default values significantly reduces the membership values of most of the trees particularly Trees 2, 4, 5, 9 and 10; however, the overall clusters are unchanged. Similarly, setting w_P=2 with all other weights set to their default values causes no change in the final clusters but the membership values are significantly reduced with the exception of Trees 1 and 3. Finally, increasing the weight w_E to w_E=5 with all other weights set to their default values produces the clusters {Tree 1 – Tree 3} and {Tree 4 – Tree 10} where Trees 9 and 10 have very low membership values. Most of the other membership values are greater than 0.8 except for Tree 5 which is at 0.7.

In summary, we have shown that both proximity fuzzy clustering and relational FCM are suitable tools for clustering variability models which are graphs with very special properties, many of which are specific to software engineering.

9.5. Software Requirement Analysis – Case Study

9.5.1. Overview

As detailed in the previous section, the clustering of graphs based on their proximity is a relevant and increasingly prominent problem in software development and in particular in Software Product Line development. In addition, in previous section we have illustrated that a variety of (fuzzy) clustering techniques can be used to cluster graphs based on their respective proximity to each other.

To demonstrate and analyze the usability and accuracy of these clustering approaches within the context of clustering variability models for SPL development, we perform the clustering of a variety of variability models of the Crisis Management System (CMS) case, which was first published in [113]. In this section, we will first introduce the CMS case and details of its data sets that are used in this case study. This is followed by the derivation of proximity values for a variety of criteria and the description of the subsequent clustering results. Finally, we examine and compare the results of the various clustering methods with respect to the SPL development context.

9.5.2. Case Study: Crisis Management System

The CMS is a software system that automates the coordination of crisis situations, ranging from identifying the occurrence of a crisis, dispatching resources such as ambulances to reporting and storing the completion of the crisis. The objective of a CMS when faced with a crisis situation such as natural disasters or terrorist attacks is stated as [113]:

> *"Crisis management involves identifying, assessing, and handling the crisis situation. A crisis management system facilitates this process by orchestrating the communication between all parties involved in handling the crisis. The CMS allocates and manages resources, and provides access to relevant crisis-related information to authorized users of the CMS."*

The CMS covers a large set of responsibilities and the case description introduces approximately 35 textual requirements that describe the base functionality and quality of service that is to be

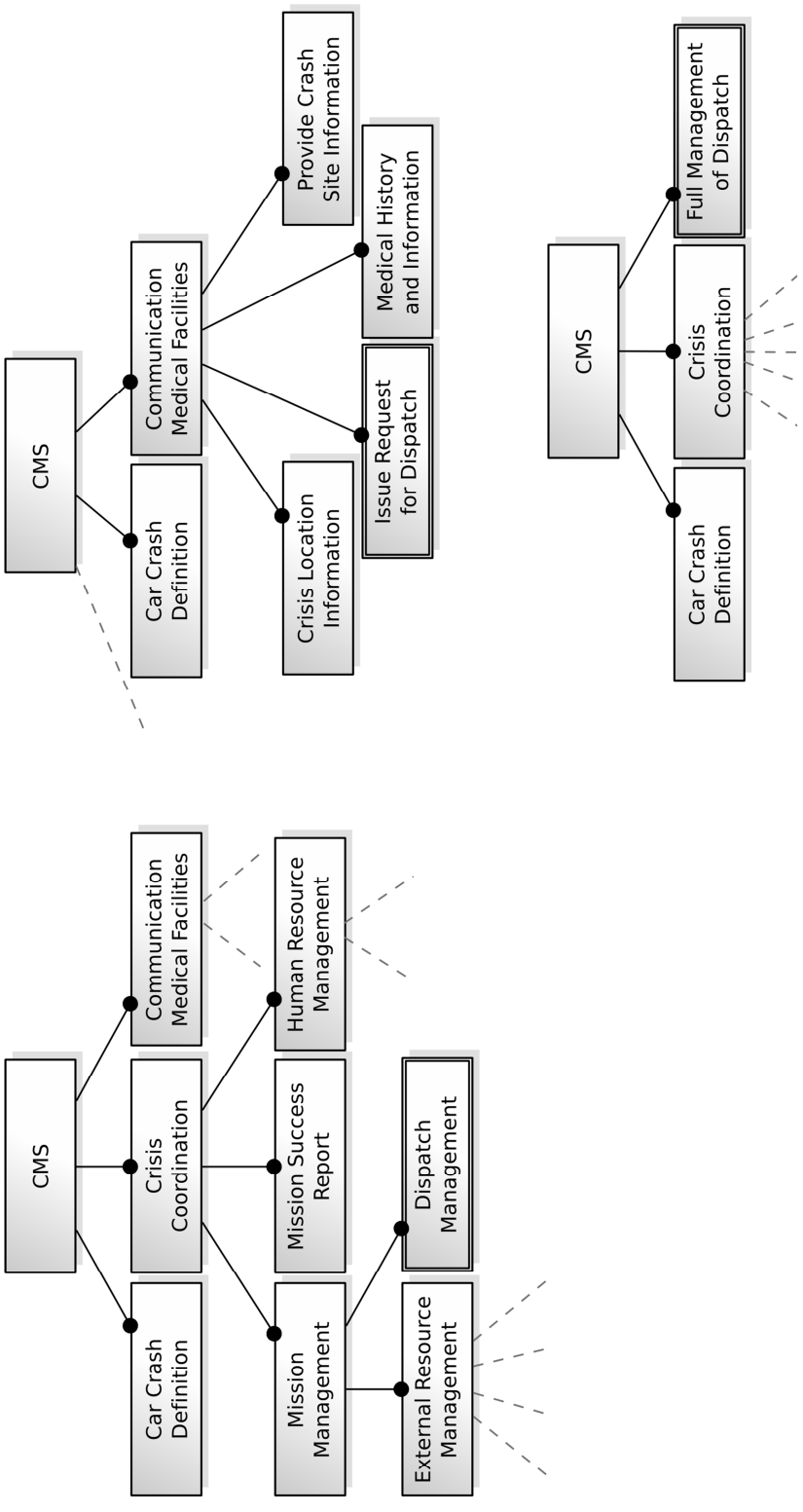


Figure 49. Variability Model Fragments of the Crisis management System

expected of the CMS. The case description in [113] also makes clear that the CMS case in essence describes an SPL, as a typical instantiation of the CMS will be more specialized, e.g. a car crash CMS or an airport CMS. To illustrate some of the functionality provided by a CMS SPL, consider the variability model fragments depicted in Figure 49.

In this figure, fragments of three possible decompositions of the CMS SPL are depicted based on the requirements specification provided by the case description. The actual variability models of the CMS contain a significantly large amount of nodes. Due to size and readability constraints, the inclusion of the full trees has been omitted. It can be seen that the various responsibilities, such as *Crisis Coordination* and *Communication Facilities* are part of the CMS and depending on the viewpoint and reusability needs end up in different segments of the variability models. Each of the nodes in these variability models is attributed with a description in natural language detailing the responsibilities of the sub-concept.

9.5.3. CRISIS MANAGEMENT SYSTEM DATA SETS

As stated in the previous subsection, the CMS case describes a range of crisis management systems which can be captured in a single SPL. But as we underlined in Section 12.5.2, there is no single correct variability model that contains the right decomposition of concepts for the CMS. Developers of the CMS SPL therefore need to access a knowledge base of variability models of CMS systems. To represent such knowledge bases, we have compiled two data sets of variability models that describe the Crisis Management System.

Data Set 1

The first data set consists of 15 variability models of the CMS that originate from various sources. Seven have been produced using an automated tool called Arborcraft [207]. Arborcraft was provided with the textual description of the CMS case and the derivation process was executed with various different parameters. Also, the CMS description was provided to expert software designers to provide three variability models. The data set is completed by including variability models that result from a related textual description of a more specialized crisis management system, which has resulted in one expert-derived and four Arborcraft derived variability models.

Data Set 2

The second data set consists of 14 variability models, again a combination of manually and automatically derived. The majority again is based the original CMS description and for the automated derivation an improved version of Arborcraft is used. Also, three manually derived variability models of the CMS description are included. This set is complemented with three models from the specialized CMS description.

The models in these sets have been clustered with respect to three criteria: *Logging*, *Missions* and *Strategies*. Each of these criteria defines a number of nodes based on which the proximity of the models in the data set is calculated and consequently the clustering is performed. Each node in a criterion is represented by a textual description of the sub-concept it represents.

9.5.4. *CLUSTERING RESULTS*

The data sets are clustered using proximity fuzzy clustering which determines the optimal number of clusters *c* and the optimal value for the fuzzification coefficient *m* automatically. We also perform relational FCM clustering for comparison with the same number of clusters. The results for data set 1 are shown in Table 36 and the results for data set 2 are shown in Table 37. The variability models M1-M29 are described in Appendix B.

Table 36. Clustering results for data set 1

Data Set 1	Proximity Fuzzy Clustering			Relational FCM		
	c	m	Clusters	c	m	Clusters
Logging	3	2.3	{M15}, {M7,M11,M12,M13}, {M1,M2,M3,M4,M5,M6,M8,M9,M10,M14}	3	2.0	{M8,M9,M10}, {M7,M11,M12,M13}, {M1,M2,M3,M4,M5,M6,M14,M15}
Missions	3	4.0	{M1,M2,M3,M4,M5,M6,M7,M11}, {M8,M9,M10}, {M12,M13,M14,M15}	3	2.0	{M1,M2,M3,M4,M5,M6,M7,M11}, {M8,M9,M10}, {M12,M13,M14,M15}
Strategies	2	4.0	{M1,M2,M3,M4,M5,M6,M7,M11}, {M8,M9,M10,M12,M13,M14,M15}	2	2.0	{M8,M9,M10,M12,M13,M14,M15}, {M1,M2,M3,M4,M5,M6,M7,M11}

Table 37. Clustering results for data set 2

Data Set 2	Proximity Fuzzy Clustering			Relational FCM		
	c	m	Clusters	c	m	Clusters
Logging	3	4.0	{M16,M19,M20,M24,M25,M26,M28}, {M17,M21,M22,M27}, {M18,M23,M29}	3	2.0	{M16,M19,M20,M24,M25,M26,M28}, {M17,M21,M22,M27}, {M18,M23,M29}
Missions	3	2.7	{M16,M17,M22,M23,M27}, {M20,M21,M24,M25, M26}, {M18,M19,M28,M29}	3	2.0	{M20,M21,M24,M25,M26}, {M16,M17,M22,M23,M27}, {M18,M19,M28,M29}
Strategies	2	4.0	{M18,M19,M20,M21,M24,M25,M26,M28,M29}, {M16,M17,M22,M23,M27}	2	2.0	{M18,M19,M20,M21,M24,M25,M26,M28,M29}, {M16,M17,M22,M23,M27}

The cardinality values (sum of membership values) of the fuzzy clusters are given in Table 38

Table 38. Cardinalities of the fuzzy clusters

Data	Criterion	Proximity Fuzzy Clustering	Relational FCM
Data Set 1	Logging	{2.5,4.3,8.2}	{4.1,4.6,6.2}
	Missions	{5.8,5.0,4.2}	{6.4,4.5,4.1}
	Strategies	{8.6,6.4}	{6.4,8.6}
Data Set 2	Logging	{5.2,4.8,3.9}	{5.5,4.6,3.9}
	Missions	{4.8,5.0,4.2}	{4.8,4.8,4.3}
	Strategies	{7.0,7.0}	{7.5,6.5}

The fuzzy membership values for proximity fuzzy clustering and relational FCM are given in the Figure 50 and Figure 51.

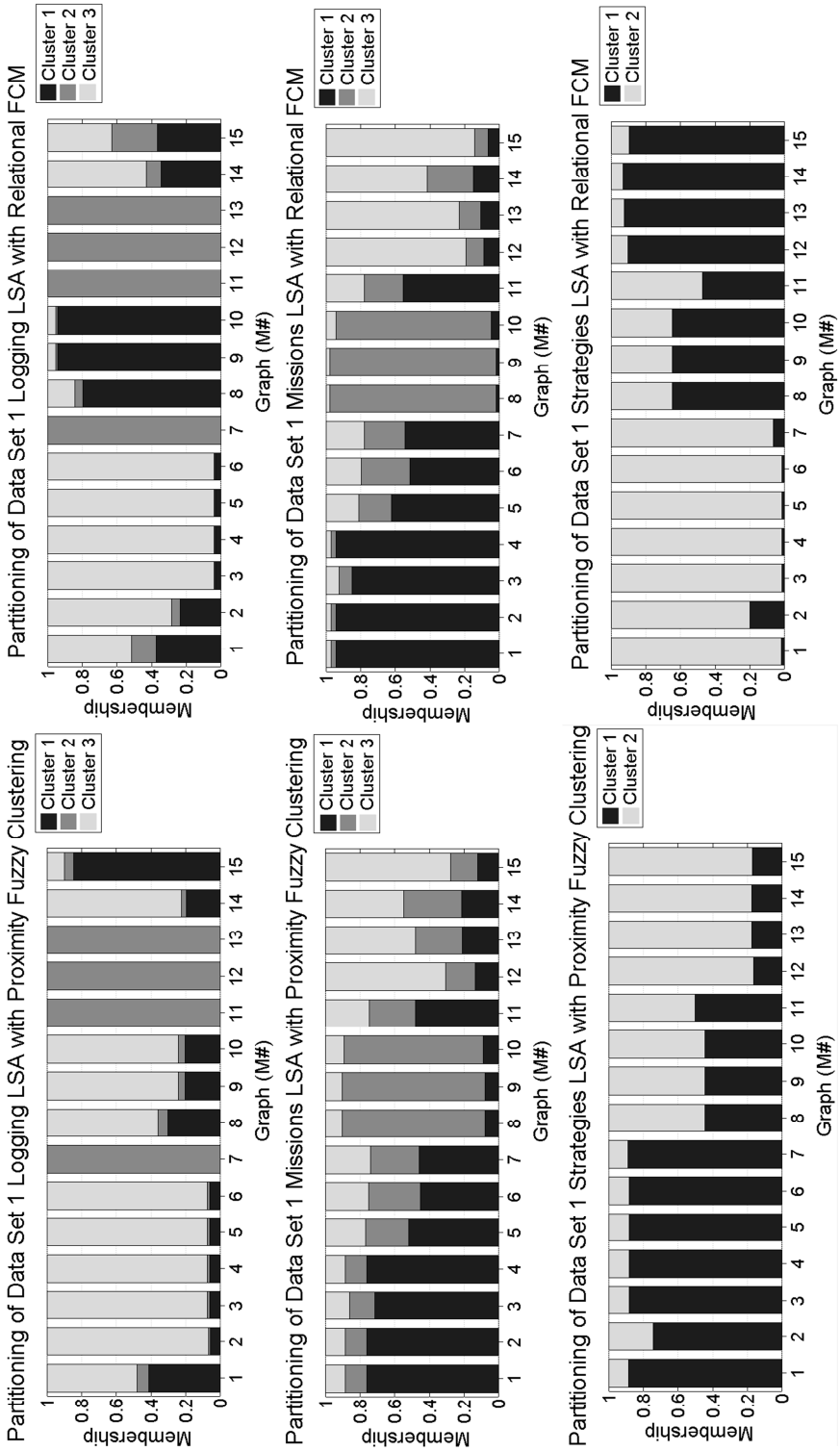


Figure 50. Partitioning of Data Set 1

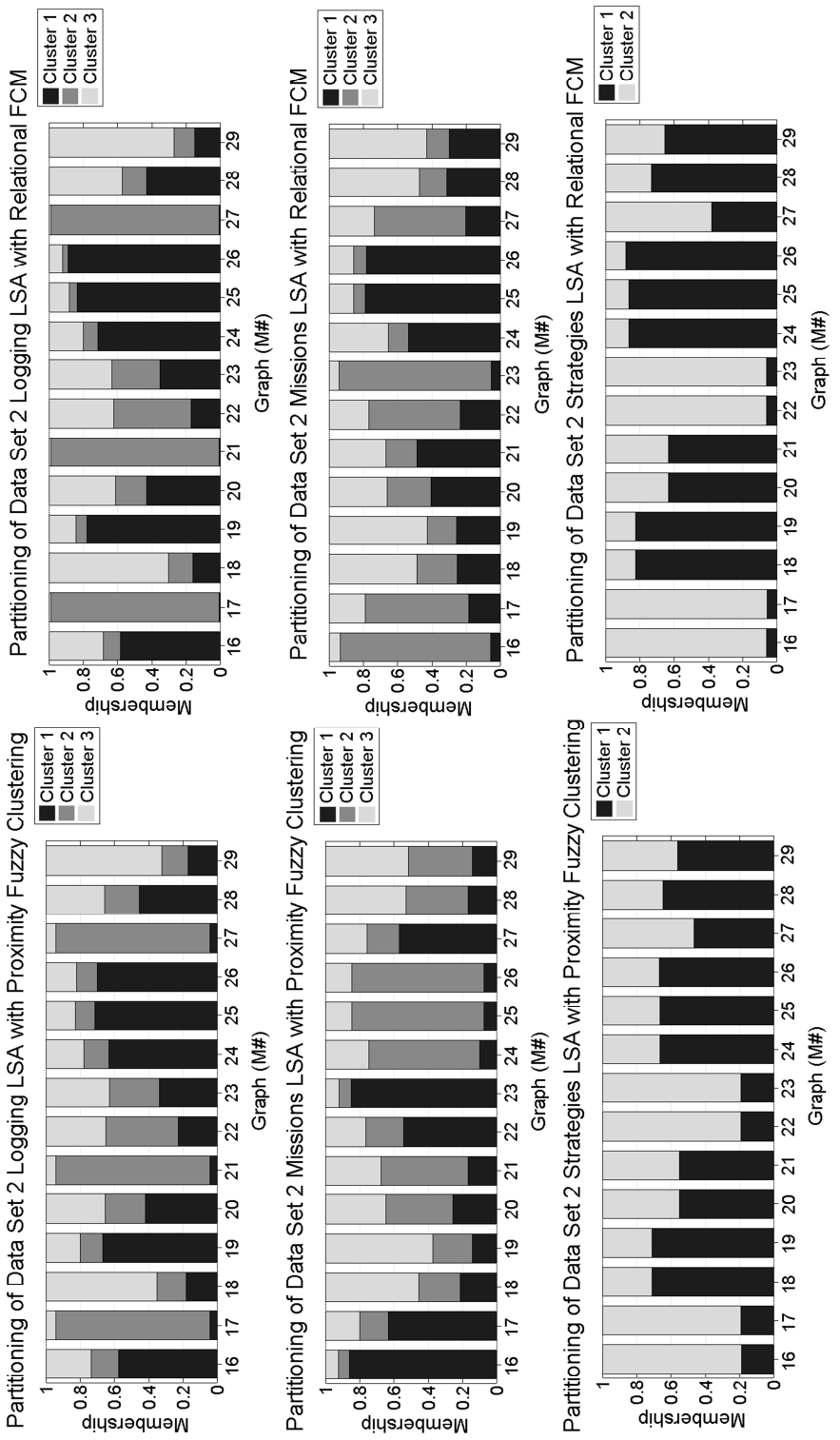


Figure 51. Partitioning of Data Set 2

Graphs M14 and M15 in the Logging Data Set 1 have considerably lower membership values with relational FCM which results in a significant difference in the overall clusters. The clustering produced by proximity fuzzy clustering is closer to the result expected by an expert since the relational FCM is unable to determine that M15 belongs in its own cluster as it is very different from the other graphs. The clustering results of proximity fuzzy clustering and relational FCM are otherwise quite similar.

We also performed a simple synthetic evaluation of proximity fuzzy clustering and relational FCM where the proximity matrix was the identity matrix and the number of clusters was equal to the number of graphs, i.e. the graphs were all extremely distinct. Proximity fuzzy clustering placed each graph in its own cluster as expected; however, relational FCM was unable to place each graph its own cluster since several clusters contained multiple graphs. As a result, we conclude that proximity fuzzy clustering is able to detect single-graph clusters quite well whereas relational FCM has difficulty detecting clusters consisting of only one graph. The situation where we have one or two graphs that are very different from each other and thus belong in their own clusters is quite common in software engineering.

9.5.5. *Clustering Results in the Context of Software Requirements Analysis*

9.5.5.1. *Deriving Structuring Advice from Clustering*

In the previous section, we have demonstrated that relational and proximity clustering can be applied to group variability models used in Software Product Line development for a particular criterion of interest. The application of these results lies in the structuring advice for new variability models that can be derived from the achieved clustering, as detailed in Section 12.4.2. In this section we interpret the clustering results of the previous section to demonstrate how such structuring advice can be derived from the clustering of the variability models.

To illustrate the use of the clustering as design advice, we examine the result when *Data Set 1* is clustered for the *Logging* criterion. The clustering that results when using proximity clustering is the following:

Cluster 1: {M15}

Cluster 2: {M7,M11,M12,M13}

Cluster 3: {M1,M2,M3,M4,M5,M6,M8,M9,M10,M14}

Each of these clusters contains variability models that structure the nodes described in the *Logging* criterion in a similar manner. It can be seen that the cluster 3 with 10 elements by far is the largest cluster. This means the structuring of the nodes in the *Logging* criterion used in this cluster occurs more frequently than the structuring used in the other clusters.

When we examine the variability models contained in these clusters, we find they all represent a distinct way of structuring the nodes in the *Logging* criterion. The models in cluster 3 contain (almost) all the nodes described in the Logging criterion. Moreover, the models invariably group these nodes closely together to form an important sub-concept of the variability model. This is illustrated by the model fragments depicted in Figure 52.

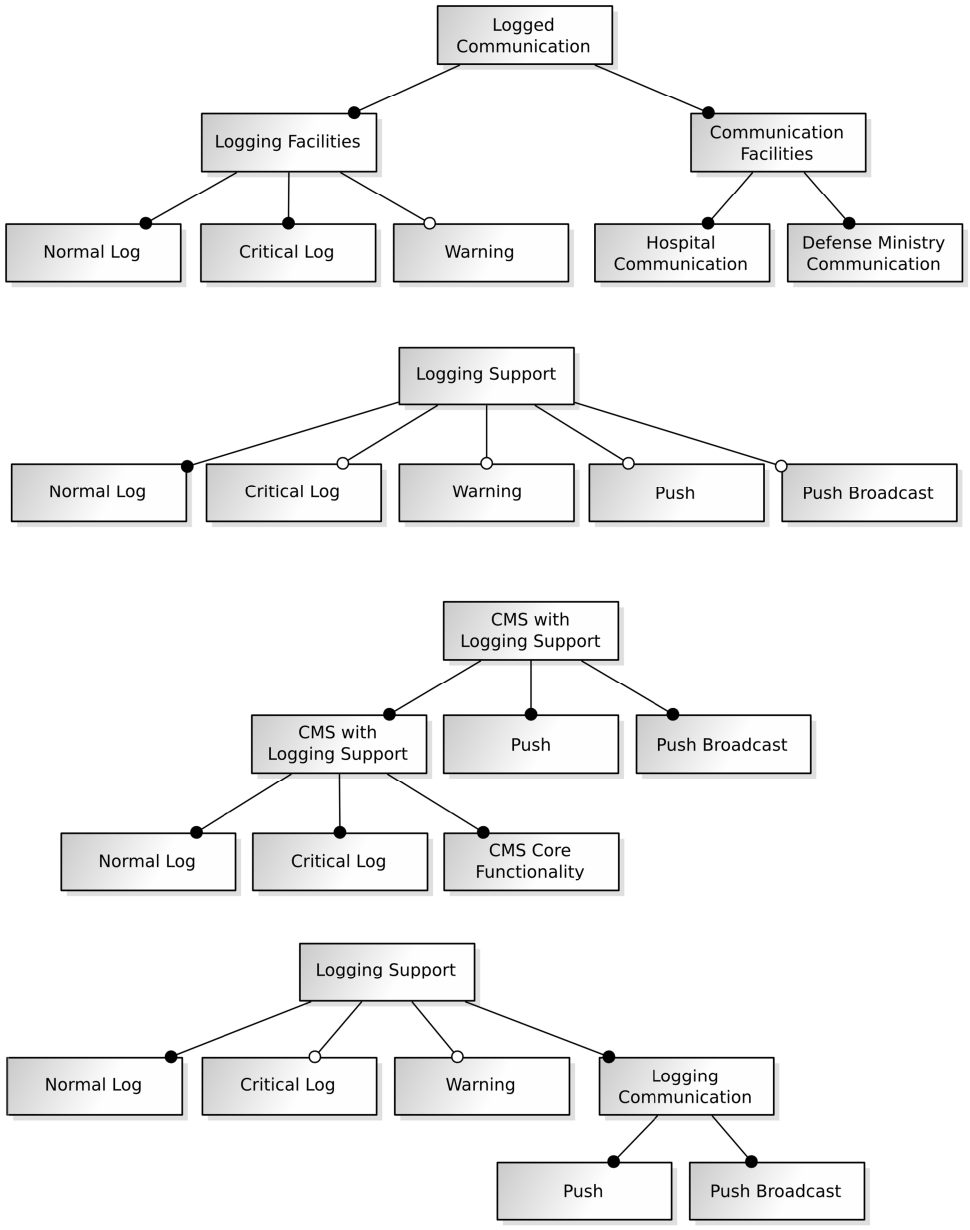


Figure 52. Fragments of Variability Models of Cluster 1

In this figure, fragments of four variability models in cluster 3 are depicted. In each of these four fragments, the four nodes of the *Logging* criterion, *Normal Log*, *Critical Log*, *Warning* and *Push Broadcast*, are grouped closely together. In particular the nodes *Normal Log* and *Critical Log* are always grouped under the same parent. Also, it can be seen that the node *Push Broadcast* frequently occurs with the sibling node *Push*. Software developers can interpret this as advice to consider including this functionality as it appears to be strongly related to the *Logging* criterion.

The models contained in the cluster 2 do not contain any of the nodes in the criterion as they describe crisis management systems that do not incorporate facilities for logging activities during

crisis management. When analyzing variability models this is likely to occur and it can also be interpreted as structuring advice; rather than indicating how the nodes of criterion are to be structured, the advice that can be derived from this cluster is to not include logging facilities in the new crisis management system. This advice can prevent the inclusion of a surplus of functionality and over-engineering of a new Software Product Line.

The single variability model contained in cluster 1 only contains a small amount of the nodes of the *Logging* criterion which have scattered across the variability model. The lackluster structuring hints at a badly designed variability model compared to the other models in the data set. This is underlined by the fact that the cluster only contains a single model. Based on the low cardinality, the cluster can be dismissed as irrelevant for structuring advice.

Interesting to note is the difference in clustering that results from relational clustering for the *Logging* criterion with data set 1. In this clustering model M8, M9 and M10 are grouped separately. When these models are examined, it is clear that they group the nodes of the *Logging* criterion together, but at a different location in the models (further away from the root). While technically this can be seen as different structuring advice, considering the semantics of variability models the grouping provided by proximity clustering comes closer to the intuition of software developers.

9.5.5.2. *Reflecting on structuring advice from clustering*

The structuring advice that can be derived from clustering results as demonstrated in the previous section cannot be seen as sufficient for actual decision making during software development activities. Rather, the role of such information lies in providing decision support, providing the software developer with potentially interesting pieces of information that can trigger new ideas to improve the design and development of the software system. However, it is very important that the developer be provided with accurate figures about the quality of the structuring advice to consider it accordingly.

The numerical properties provided by our approach enable the developer to assess the quality of the structuring advice. The clustering that results consists of fuzzy sets. The membership degree of each variability model in a specific cluster can be used as in indicator for the importance of the structuring advice it contains. When a model has a membership value close to 0, its structuring advice should be considered less for that particular cluster. In the case study this can be seen by examining Figure 52. The fragment at the right top depicts a somewhat different way of structuring the logging functionality than the other three fragments in spite of belonging to the same cluster. Before merging all the logging concepts, part of the CMS functionality is combined with it. As it turns out, the model to which this fragment belongs also has a lower membership value than the other three, which means the importance of this fragment for the derived structuring advice can be reduced.

Moreover, the use of fuzzy sets allows us to provide a more accurate representation of the cardinalities of the clusters, which is used by the developer to identify the most important structuring advice. For example, in Table 36 it can be seen that the clusters for the *Mission* criterion and data set 1 have a cardinality of 8, 4 and 5. However, Table 38 indicates the actual cardinalities of the fuzzy sets from which they are derived are 5.8, 5.0 and 4.2, which means the choice for the cluster with the highest cardinality is not as trivial as it seems. By including such information in the structuring advice, its value and credibility becomes more apparent to the software developer. This can then be used by the developer to refine the clustering or further examine interesting elements provided by the clustering result.

9.6. CONCLUSIONS

We presented a novel approach to clustering a collection of graphs, particularly we have successfully demonstrated the application in the domain of software requirements engineering. A novel proximity function, which measures the proximity between pairs of graphs, is also presented based on a modified edit distance which includes measuring the proximity between nodes (ie. software attributes) in the variability model according to the requirements description. We do this in order to produce a more accurate measure of proximity between variability models. Our clustering results closely match the expected partitions produced by an expert who independently and manually partitioned the graphs.

In addition, we demonstrated how this approach can be applied to search for best practices in structuring new variability models during Software Product Line development by examining a knowledge base of existing models. By clustering the models in the knowledge base based on a provided criterion of interest, frequently used ways of structuring the criterion can be identified and used to provide decision support to the software developer on how to structure the model. Moreover, the quality and reliability of this advice can be communicated to the developer by using attributes and performance indicators of our approach, such as the cardinalities and cohesion of the resulting clusters.

Although the data used in our case study is derived from real-world variability models, acquiring a complete set of real variability models is rather difficult. An important future direction of this research therefore focuses on acquiring a collection of real variability models. We aim to apply the system in practice in order to evaluate its effectiveness and suitability to software engineers in future work. The tool will be used to provide software developers valuable advice in reusing the design aspects of a *real* software project.

10. TIME SERIES PREDICTION WITH PROXIMITY-BASED FUZZY CLUSTERING

Data that changes with time is a very common occurrence in fields such as economics, environmental sciences, astronomy, physics, chemistry, biology, and many others. A common question presented is that when given a sequence of time-varying data, are there temporal tendencies in the past that provide clues for potential future time series behaviour? More specifically how can historical information be used to effectively predict future data samples? The aim of this study is to build a time series prediction model to accomplish that. Mathematically the system is described by an expression (**10-1**) that predicts such data one sample into the future

$$\hat{x}(t) = G\left(x(t-1), x(t-2), \ldots, x(t-q_{order})\right) \quad \textbf{(10-1)}$$

where t is the time index of the sample to be predicted, $\hat{x}(t)$ is the predicted sample, $x(t-1), x(t-2), \ldots, x(t-q_{order})$ are known historical samples in the time series, and G is the underlying model of the time series of order q_{order}. A logical extension to this system is to predict several samples, say n-samples, into the future.

Traditionally one would use an autoregressive (AR) model in place of F; however, such a model has limiting prediction power for several reasons

1. Model linearity – not able to handle non-linear temporal behavior
2. Model stationarity – not able to handle non-stationarity in time series

In order to circumvent these issues, a recurrent neural network can be used in place of F; however, a neural network is a black-box with no known way to interpret the meaning of the network weights once learned. Additionally a recurrent neural network often has difficulty when learning from a time series with highly varying dynamics since the network needs to learn all aspects of the dynamics of the series using only the global model or the "big picture." A more beneficial approach would be to build several smaller models that exceed well on learning and predicting "localized" time series dynamics.

We propose an architecture that addresses both the transparency issue and the global modeling issue using a fuzzy inference system (FIS) designed in such a way to break the model into a collection of rules encapsulating local models that exceed at prediction of localized regions in time series behaviour [75]. Our goal is to create a fuzzy rule-based system such as the one depicted in Figure 53.

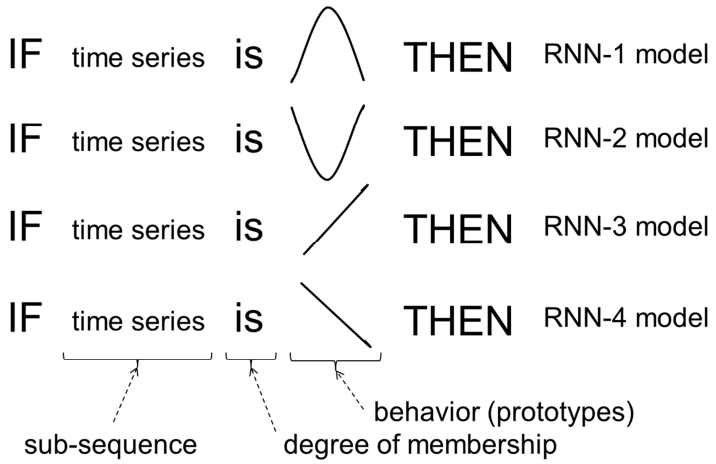


Figure 53. Example of Fuzzy Rule-based Architecture

In this example, the recurrent neural networks (RNN) form predictive models that are specialized for specific recurring patterns discovered in the time series via clustering. These recurring patterns form the antecedents (or conditions) of the rules. The motivation behind this architecture is that the rules provide important information regarding the structure of the time series. Therefore, discovering structure in the time series via clustering is an important focal point in this chapter. Standard Fuzzy C-Means (FCM) clustering was used in the past to determine the antecedents of the fuzzy rules with an Adaptive Neural Fuzzy Inference System (ANFIS), cf. [75]. Unfortunately, FCM has serious limitations since it is constrained to using the Euclidean distance which is not well suited for measuring the distance between the time series. This study aims to address this issue by using an improved FIS prediction architecture that replaces FCM with proximity-based fuzzy clustering alleviating this problem.

We follow the standard notation used in fuzzy modeling, time series and statistics. The finite length discrete-time series is denoted by $\{x(t)\}$ t=1,2,...,*T* where *T* is the number of samples in this series. The predicted values of the time series at instant "*t*" is denoted by . A data set X of *N* sub-time series (temporal segments) formed on a basis of $\{x(t)\}$ is constructed as $\mathrm{X}=\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_t,\ldots,\mathbf{x}_N\}$ where $\mathbf{x}_t=\left[x(t+d-1),x(t+d-2),\ldots,x(t)\right]^{\mathrm{T}}$ is a vector of dimensionality *d* which represents a sub-time series of *d* samples for *n*=1,2...*T* such that the size of the data set is *N*=*T*-*d*. The expectation or mean of a time series taken from statistics is given by $\mu_{x(t)}=E\{x(t)\}=\frac{1}{T}\sum_{t=1}^{T}x(t)$.

10.1. MODEL ARCHITECTURE

The topology of individual rules in the TSK (Takagi-Sugeno-Kang) fuzzy model used for prediction of time series shares some features that are common to all rule-based models. There are also some features that are specific to the dynamics of the underlying time series. The general architecture of the TSK model is shown in Figure 54. The underlying rules come in the form

$$\text{IF } \mathbf{x}_t \; is \; f_i(\mathbf{x}_t) \text{ THEN } y_i(t)=g_i\left(x(t-1),\ldots,x(t-q_{order})\right) \qquad \textbf{(10-2)}$$

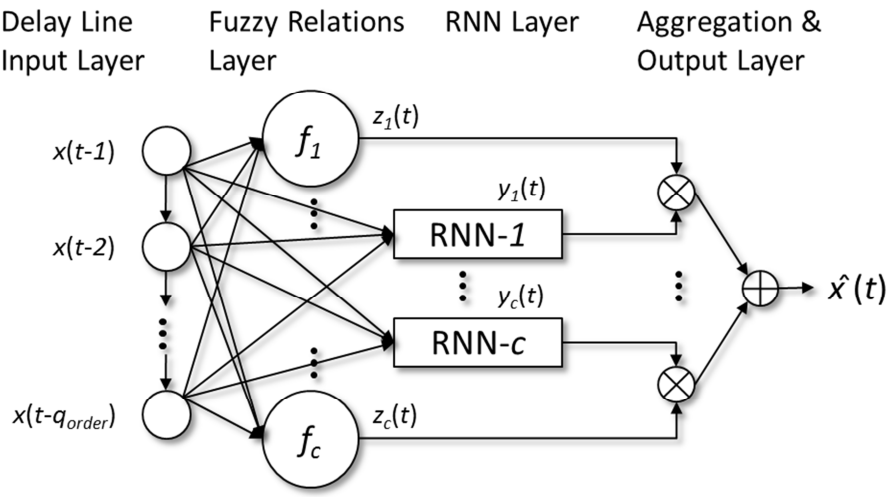


Figure 54: An architecture of the TSK model

The network exhibits several layers:

Input layer. The inputs $x(t\text{-}1)$, $x(t\text{-}2)$, …, $x(t\text{-}d)$ are provided through a tapped delay line of length "d." The length of the delay line is reflective of the dynamics of the time series.

Layer of fuzzy relations. This layer consists of "c" fuzzy relations (multidimensional fuzzy sets) f_1, f_2, …, f_c formed in the Cartesian product of the tapped samples. At this layer we transform the multidimensional samples into the corresponding degrees of activation (degrees of firing) of the corresponding rules.

Conclusion (RNN) layer. The layer consists of local recurrent neural networks of order $q_{order} \le d$, RNN-1, RNN-2, …, RNN-c which accept inputs from the input layer. The local nature of the processing here pertains to the fact that the results of processing provided by the RNN-j are taken into account by the rule-based system when the corresponding inputs are within the realm of the fuzzy relation f_i (which triggers acceptance of these results at the degree of activation of the condition part f_i).

The sub-models f_i for i=1,2,…,c for the rule consequents is a recurrent neuron expressed as

$$y_i(t) = g_i\left(x(t-1),\ldots,x(t-q_{order})\right) = g\left(b_i + \sum_{j=1}^{q_{order}} x(t-j)\,w_{ij}\right) \qquad \textbf{(10-3)}$$

where g is the activation function, b_i is the bias, and w_i, j=1,2,…,q are the weights of the time delayed input vector. The details of the recurrent neuron model are depicted in Figure 55.

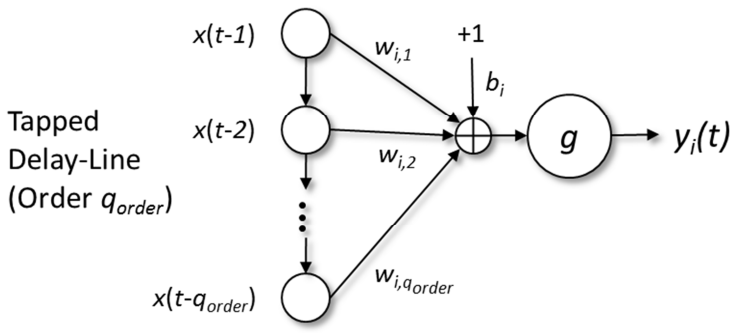


Figure 55: Processing realized by the neuron

Aggregation layer. At this layer, the results produced by the local RNNs are aggregated given the respective levels of "firing" of the condition parts, f_1, f_2, …, f_c. The aggregation comes in the same format as commonly reported in rule-based systems. More specifically, let $z_i(t)$ denote the activation level of the i-th condition, that is $z_i[n] = A_i(\mathbf{x}_n)$ while the results produced by the networks are equal to $y_1(t)$, $y_2(t)$, …,$y_c(t)$, respectively. The result of aggregation is then taken as a weighted sum

$$\hat{x}(t) = \sum_{i=1}^{c} z_i(t)\, y_i(t) \qquad (\mathbf{10\text{-}4})$$

The aggregation is designed to combine the predictive power of the local models to form a global model that performs exceedingly well.

10.2. NETWORK DESIGN

The development of the network consists of two main phases. First, we construct information granules – fuzzy sets in the input space which constitute a condition part of the rules. Second, we optimize the parameters (connections) of the recurrent neural networks.

10.2.1. GRANULATION OF INPUT SPACE THROUGH PROXIMITY FUZZY CLUSTERING

We introduce a new algorithm for learning the fuzzy relations of the if-then rules by using proximity-based fuzzy clustering. Our goal is to discover clusters in the data through proximity fuzzy clustering. These clusters represent common patterns or behaviour in the time series which will then be used to form antecedents in the fuzzy rules. The granulation of the time series is given in terms of the prototypes of the clusters. The prototypes are therefore commonly observed sub-sequences in the time series.

An important problem is the selection of the proximity function in proximity fuzzy clustering. The commonly used correlation function is an appropriate choice since correlation measures the similarity in terms of behaviour between two time series. In other words, the assumption is that highly correlated time series should be placed together in the same cluster. The proximity between two sub-sequence time series $\mathbf{x}_{t_1}, \mathbf{x}_{t_2} \in \mathbf{X}$ vectors of dimensionality (length) d is given by

$$p\left(\mathbf{x}_{t_1}, \mathbf{x}_{t_2}\right) = \frac{1}{2}\left(1 + \frac{E\left\{\left(\mathbf{x}_{t_1} - \mu_{\mathbf{x}_{t_1}}\right)\left(\mathbf{x}_{t_2} - \mu_{\mathbf{x}_{t_2}}\right)\right\}}{\sigma_{\mathbf{x}_{t_1}}\sigma_{\mathbf{x}_{t_2}}}\right) \quad \textbf{(10-5)}$$

where the mean is given by $\mu_{\mathbf{x}_{t_1}} = E\left\{\mathbf{x}_{t_1}\right\}$ is the expectation value of the time series, and $\sigma_{\mathbf{x}_{t_1}} = \sqrt{\mathrm{E}\left\{\left(\mathbf{x}_{t_1} - \mu_{\mathbf{x}_{t_1}}\right)^2\right\}}$ is the standard deviation of the time series. The linear transformation with the 0.5 constants is used to ensure the proximity values are in the interval [0,1]. Thus time series that are highly correlated statistically will have high proximity values close to 1 and time series that are highly anti-correlated statistically will have low proximity values close to 0. This is highly desirable as correlated time series have very similar dynamic behaviour while anti-correlated time series have very different dynamic behaviour. We demonstrate the benefits procured by clustering a simple time series shown in Figure 56 with *d*=10 using proximity fuzzy clustering and Fuzzy C-Means (FCM).

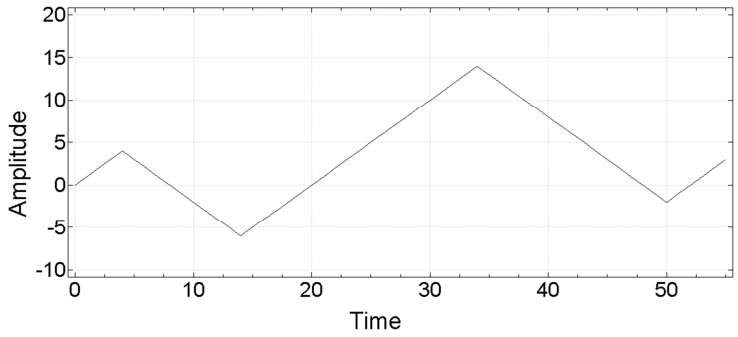
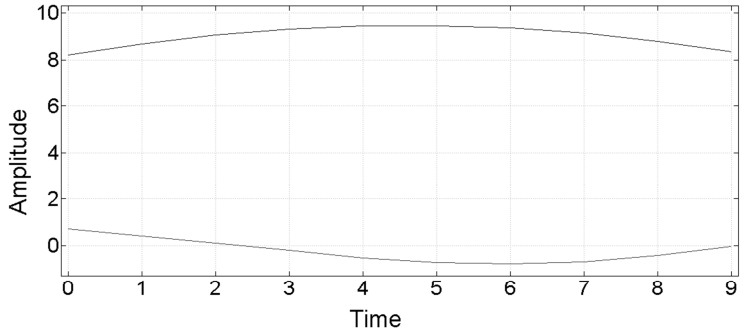


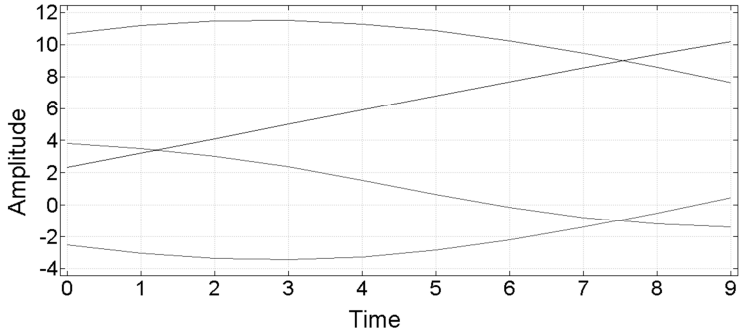
Figure 56. Example Time Series

The time series was dividing using a horizon length of *d*=10. The prototypes produced by clustering for *c*=2 and *c*=4 is shown in Figure 57 for FCM and proximity fuzzy clustering.

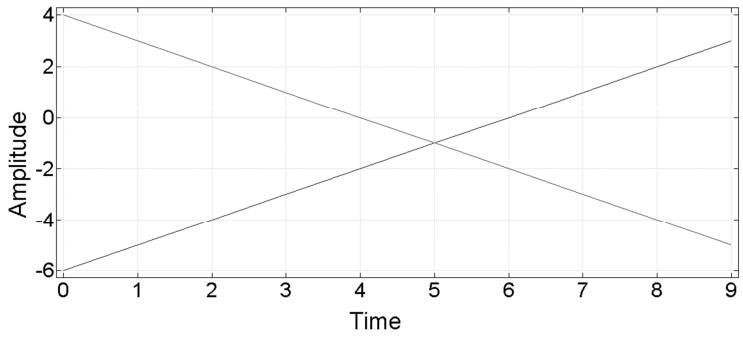
This simple example illustrates the utility of proximity fuzzy clustering to time series clustering including the learning of the antecedents in the TSK time series prediction described in this chapter. The prototypes produced by FCM do not resemble the original time series and exhibit the sinusoidal behaviour as discussed in [139]. However, proximity fuzzy clustering produces prototypes that are more informative as they more closely resemble the behaviour found in the original time series. When the number of clusters is four, the clusters represent behaviour that can be attributed with linguistic descriptions that directly correspond to behaviour observed in various parts of the time series, i.e. up, down, peak, valley.



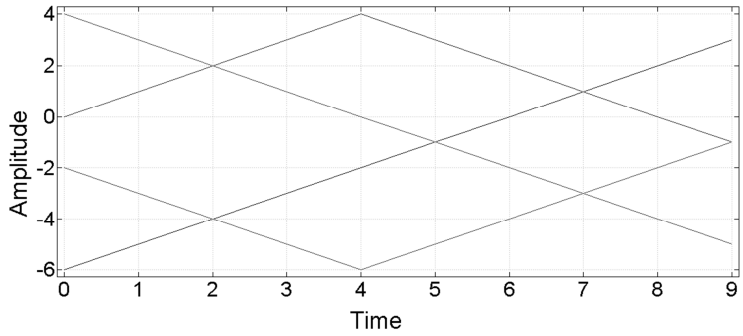
(a) FCM *c*=2, *m*=2



(b) FCM *c*=4, *m*=2



(c) Proximity fuzzy clustering *c*=2, *m*=2



(d) Proximity fuzzy clustering *c*=4, *m*=2

Figure 57. Prototypes for clustering with FCM (a) & (b) and proximity fuzzy clustering (c) and (d) for *c*=2 and *c*=4 respectively

We cannot employ FCM as a clustering vehicle since the calculation of the prototypes depends on the assumption of a Euclidean feature space. Thus we employ proximity fuzzy clustering. The prototypes are determined post-mortem clustering by selecting the time series with the maximum proximity to the *implied* cluster prototype q_{ij} according to the following expression

$$\mathbf{v}_i = \mathbf{x}_{k_i} \text{ where } k_i = \arg \max_{\forall j=1\ldots N} q_{ij} \quad \textbf{(10-6)}$$

for i=1…c. Hence, the prototypes summarize the time series dynamics captured by the antecedent in the rule. A method of calculating membership $f_i(\mathbf{x}_t)$ post-training is then accomplished using the expression

$$f_i(\mathbf{x}_t) = \frac{1}{\sum_{h=1}^{c} \left(\frac{p(\mathbf{x}_t, \mathbf{x}_t) - p(\mathbf{x}_t, \mathbf{v}_i)}{p(\mathbf{x}_t, \mathbf{x}_t) - p(\mathbf{x}_t, \mathbf{v}_h)} \right)^{\frac{2}{m-1}}} = \frac{1}{\sum_{h=1}^{c} \left(\frac{1 - p(\mathbf{x}_t, \mathbf{v}_i)}{1 - p(\mathbf{x}_t, \mathbf{v}_h)} \right)^{\frac{2}{m-1}}} \quad \textbf{(10-7)}$$

for i=1…c given a new input $\mathbf{x}_t$. The level of membership $f_i(\mathbf{x}_t)$ in the antecedent controls the strength of contribution of the local models for each rule i=1…c.

10.2.2. LEARNING OF RECURRENT NEURAL NETWORKS

The weights of the neurons are determined by off-line gradient descent which consists of several iterations of on-line gradient descent. The mean squared error [139] is the objective function of the on-line gradient descent algorithm described as

$$MSE(t) = \frac{1}{2}\left(x(t) - \hat{x}(t)\right)^2 \quad \textbf{(10-8)}$$

where $MSE(t)$ is the squared error at time t. The update rule for the weights of the neurons in the rule consequents is

$$\Delta w_{ij}(t) = \eta e(t) z_i(t) g'\left(net_i(t)\right) x(t-j) \quad \textbf{(10-9)}$$

where η is the learning rate and $net_i(t) = b_i + \sum_{j=1}^{q_{order}} x(t-j) w_{ij}(t)$, cf. [139]. he term $w_{ij}(t)$ denotes the weights at the current time "t." We have explicitly used the notation $w_{ij}(t)$ to emphasize that the values of the weights change with any time a new data sample is considered in the optimization process. The update rule described by equation (**10-9**) governs the on-line learning procedure since the weights are updated without seeing the entire time series that is the weights are updated using the current sample and previous samples. It is possible to improve convergence by evaluating the update rule given in (**10-9**) several times when a new sample is presented to the system rather than once [139]. The number of times the on-line update rule is applied each time a new sample point is presented to the system is referred to as the number of adaptive passes.

Off-line learning is accomplished by successively completing several on-line learning iterations over the entire time series. In this chapter, an on-line learning iteration is referred to as an epoch.

10.3. EXPERIMENTS

Several synthetic and real-world experiments are conducted. The Mackey Glass is a common time series prediction benchmark used in literature [96]. The Wolf sunspot time series for years from 1749-2005 obtained from [192] is another common time series prediction benchmark. The Lorenz time series obtained from [205] is also used for time series prediction. Three real data sets obtained from the website [94] were used including annual oil prices from 1870 to 1997 [94], time series of star magnitude brightness [94], and daily IBM stock price from 1980 to 1992 [94]. The mean is subtracted from the time series and each time series is scaled to amplitude values well within the interval [-1,1] due to squashing feature of the activation function. The activation function *g* used in the following experiments is sigmoid, namely

$$g(u) = \frac{2}{1-e^{u}} - 1 \tag{10-10}$$

To evaluate the overall prediction performance, the mean squared error (MSE) and non-dimensional error index [96] (NDEI) is used. The NDEI is the root-mean-squared (RMS) error divided by the standard deviation of the test time series being predicted, i.e.

$$NDEI = \frac{\sqrt{MSE}}{\sigma_{\mathbf{x}}} = \frac{\sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(x(t)-\hat{x}(t)\right)^2}}{\sigma_{\mathbf{x}}} \tag{10-11}$$

where $\sigma_{\mathbf{x}}$ is the standard deviation of the test time series $x(t)$. To avoid including the transition region in the error calculations, the test time series is divided in half and the performance indices are calculated for the latter half of the test time series.

10.3.1. EXPERIMENTAL DATA

The experiment signals used to evaluate the performance of the FIS time series prediction model are described here including Mackey glass, Wolf sunspot, Lorenz, oil prices, star brightness and daily IBM stock prices. The mean is subtracted from each time series and the time series scaled by a scaling factor to reduce the standard deviation of the time series to less than one. This is a direct result of the inability of the recurrent neural networks to predict values outside of this interval due to the activation function.

MACKEY GLASS

The Mackey Glass [96] is a commonly used continuous time-delay differential equation in non-linear prediction that exhibits chaotic behaviour. The time series is governed by the following formula

$$\frac{dx}{dt} = \frac{a_2 x(t-a_4)}{1-x(t-a_4)^{a_3}} - a_1 x(t) \tag{10-12}$$

where $a_1, a_2, a_3, a_4 \in \mathbb{R}$ are constants. The values of the parameters most commonly used in literature are a_1=0.1, a_2=0.2, a_3=10 and a_4=17 where at the initial time t_0=0, $x(t_0)$=1.2 [96]. The differential equation is approximated by the 4th order Runge-Kutta algorithm with the time step equal to 1. The training time series set consists of 500 samples as done in [96] while the testing time series has 2499 samples.

WOLF SUNSPOT

The Wolf sunspot time series for 1749 to 2005 was obtained from http://www.spaceweather.com/java/archive.html [192]. The sunspot time series is the monthly average number of sunspots on the sun. The scale factor is 1/500. The training time series consists of the first 1500 samples and the testing time series consists of the last 1581 samples.

LORENZ

The Lorenz time series is a long synthetic chaotic time series obtained from http://www.physics.emory.edu/~weeks/research/tseries1.html [205]. The time series was scaled by 1/50. The training time series consists of the first 2000 samples and the testing time series consist of the last 2000 samples. There are a total of 16384 samples.

OIL PRICES FROM 1870 TO 1997

The average annual price of oil time series is a small data set with 128 samples obtained from [94]. The first 64 samples were used for training and the last 64 samples were used for testing. The time series was scaled by a factor of 1/50.

STAR BRIGHTNESS

The brightness of a star was measured for 600 successive midnights and the data set is obtained from [94]. The time series was scaled by a factor of 1/30. The first 300 samples were used for training and the last 300 samples were used for testing.

DAILY IBM STOCK PRICE FROM 1980 TO 1992

The daily closing IBM stock price from 1980 to 1992 is obtained from [94]. The time series was scaled by a factor of 1/100. The first 1500 samples were used for training and the last 1833 samples were used for testing.

10.3.2. ONE-STEP PREDICTION RESULTS

A number of experiments were performed on synthetic and real-world data described in the previous section. The optimal parameters were chosen by running each data set 10 times across a set of parameters and the mean and standard deviation are reported. The parameter set was determined separately for each data set by manually running experiments to determine approximate parameter values. This information was used to construct a different set of parameters for each data set. The mean squared errors (MSE) and NDEI (shown in parenthesis) are given in Table 39 and are compared to the best results obtained in [75].

Table 39. One-step prediction results

Data set	New Proximity Architecture		TSK-NFIS[1]	AR	Neural Network
	Train	Test	Test	Test	Test
Mackey-Glass	5.94e-6±2.5e-7 (2.36e-2±3e-4)	1.17e-5±2.2e-7 (2.97e-2±3e-4)	2.18e-5 (4.06e-2)	3.20e-5 (4.92e-2)	2.89e-5 (4.66e-2)
Lorenz	7.36e-7±3.17e-7 (5.36e-3±1.34e-3)	1.57e-6±8.2e-7 (7.62e-3±2.45e-3)	3.26e-6 (1.13e-2)	5.99e-8 (1.55e-3)	1.32e-5 (2.29e-2)
Wolf Sunspot	1.02e-3±1e-5 (0.399±0.001)	1.58e-3±5e-5 (0.415±0.006)	1.32e-3 (0.380)	1.36e-3 (0.385)	2.17e-3 (0.486)
Oil Prices	4.76e-3±1.9e-4 (0.395±0.007)	2.62e-2±9e-4 (0.662±0.012)	2.37e-2 (0.629)	2.44e-2 (0.638)	2.54e-2 (0.650)
Star Brightness	3.04e-4±1e-6 (5.81e-2±1e-4)	3.08e-4±1e-6 (5.88e-2±1e-4)	3.31e-4 (6.09e-2)	3.22e-4 (6.01e-2)	3.11e-4 (5.91e-2)
IBM Stock	2.01e-4±1e-7 (4.85e-2±2e-5)	4.55e-4±1.7e-5 (0.101±2e-3)	2.22e-4 (7.11e-2)	1.82e-4 (6.43e-2)	2.36e-4 (7.31e-2)

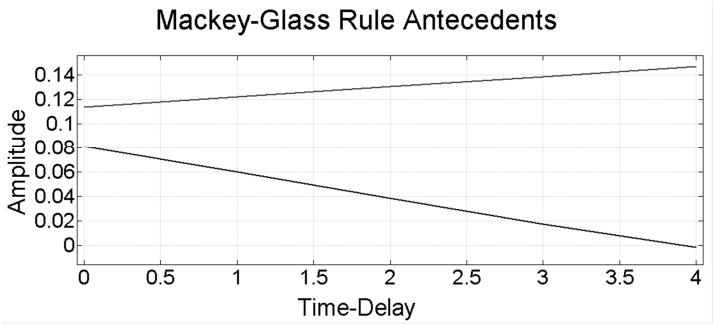
There is significant improvement on the synthetic data sets (Mackey-Glass and Lorenz) and a small improvement on the star brightness data set over the TSK-NFIS architecture; thus, the new proximity architecture is a marked improvement in the quality of the prediction. The results are fairly similar for the other data sets with the exception of the IBM stock data where the results are notably worse. An interesting result is that the complexity of the model, i.e. number of rules "*c*," order of consequent "q_{order}," and order of fuzzy relations "*d*," is significantly reduced on several of the data sets when compared to the architectures given in [75], particularly the Mackey-Glass, Wolf Sunspot and IBM stock data sets. Also, the optimal number of rules is consistently c=2 for all data sets where each rule indicates a distinctive trend, i.e. "up" and "down." This result is intuitive because the Euclidean distance depends greatly on the magnitude of the time series amplitude where as the correlation measure is invariant to the total magnitude. This results in much fewer and simpler rules than previous architectures such as TSK-NFIS for similar and in some cases better performance. The parameters used in the new proximity-based time series prediction architecture are given in Table 40.

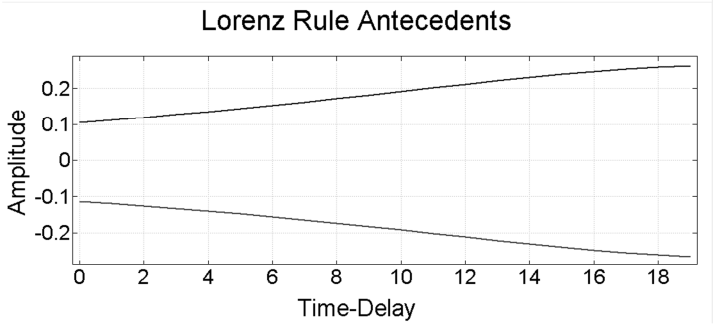
Table 40. Proximity-based time series prediction architecture parameters

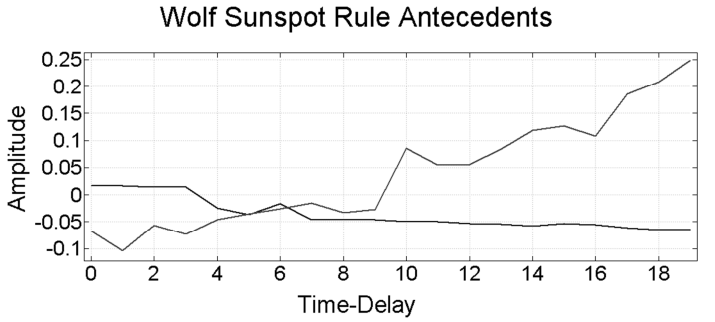
Data set	*d*	*m*	q_{order}	*c*	Epochs	η	Activation Function
Mackey-Glass	5	2	2	2	1000	0.1	Sigmoid
Lorenz	20	1.3	3	2	500	0.1	Sigmoid
Wolf Sunspot	20	3	8	2	500	0.1	Sigmoid
Oil Prices	4	2	4	2	1000	0.1	Sigmoid
Star Brightness	3	1.7	3	2	1000	0.1	Sigmoid
IBM Stock	5	1.3	2	2	1000	0.1	Sigmoid

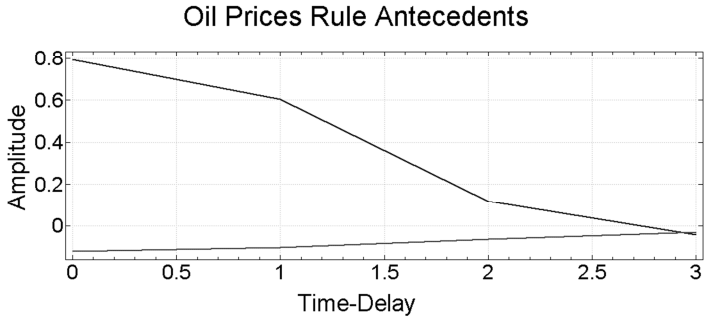
The cluster prototypes (i.e. fuzzy rule antecedents) produced by proximity fuzzy clustering are shown in Figure 58.

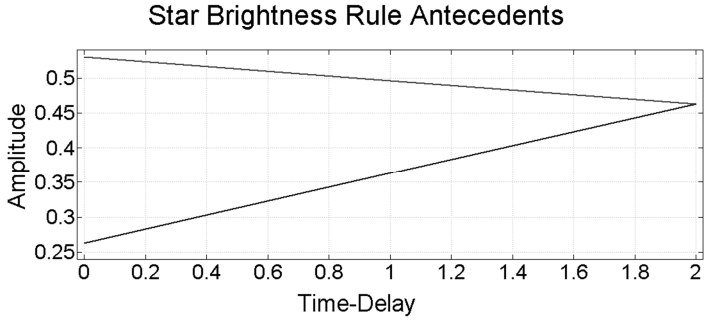
[1] The results and associated parameters for TSK-NFIS, AR, and Neural Network models are found in [75]


Mackey-Glass Rule Antecedents
Amplitude
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.5
1
1.5
2
2.5
3
3.5
4
Time-Delay


Lorenz Rule Antecedents
Amplitude
0.2
0.1
0
-0.1
-0.2
0
2
4
6
8
10
12
14
16
18
Time-Delay


Wolf Sunspot Rule Antecedents
Amplitude
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
2
4
6
8
10
12
14
16
18
Time-Delay


Oil Prices Rule Antecedents
Amplitude
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
Time-Delay


Star Brightness Rule Antecedents
Amplitude
0.5
0.45
0.4
0.35
0.3
0.25
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Time-Delay

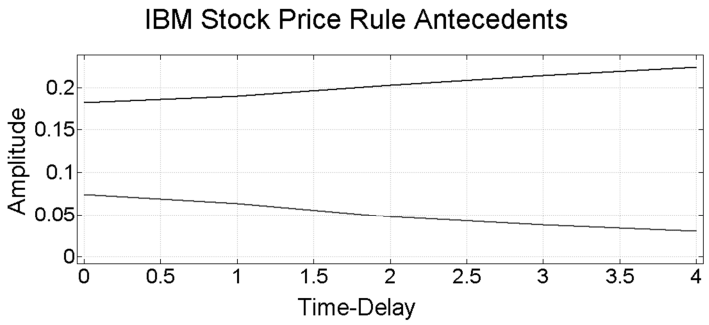
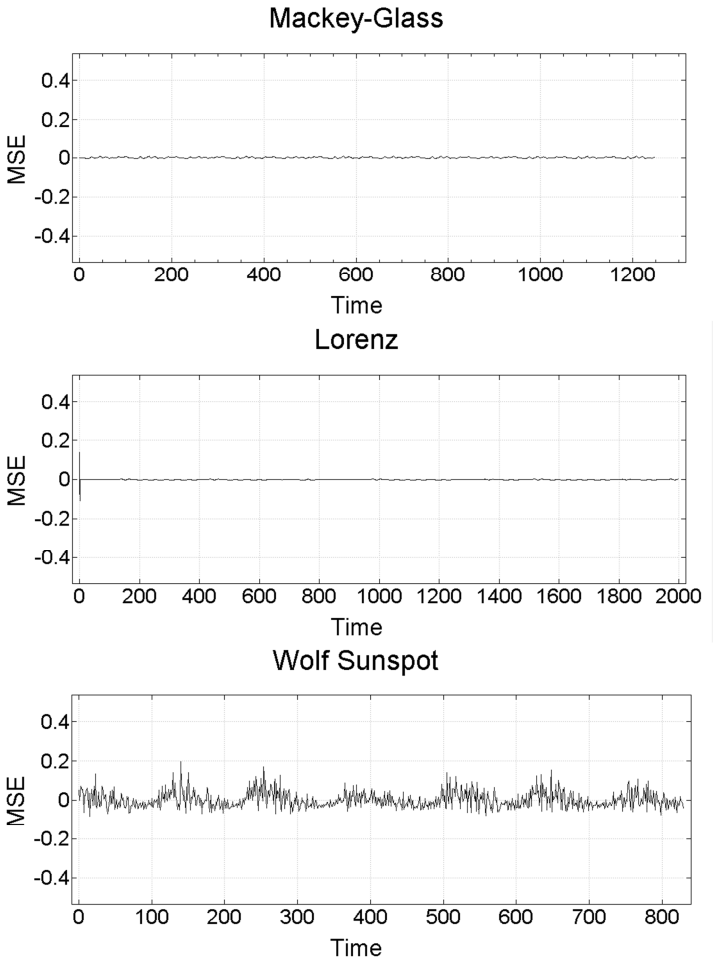


Figure 58. Rule antecedents of the proximity architecture for c=2

The error time series for each data set are shown in Figure 59.



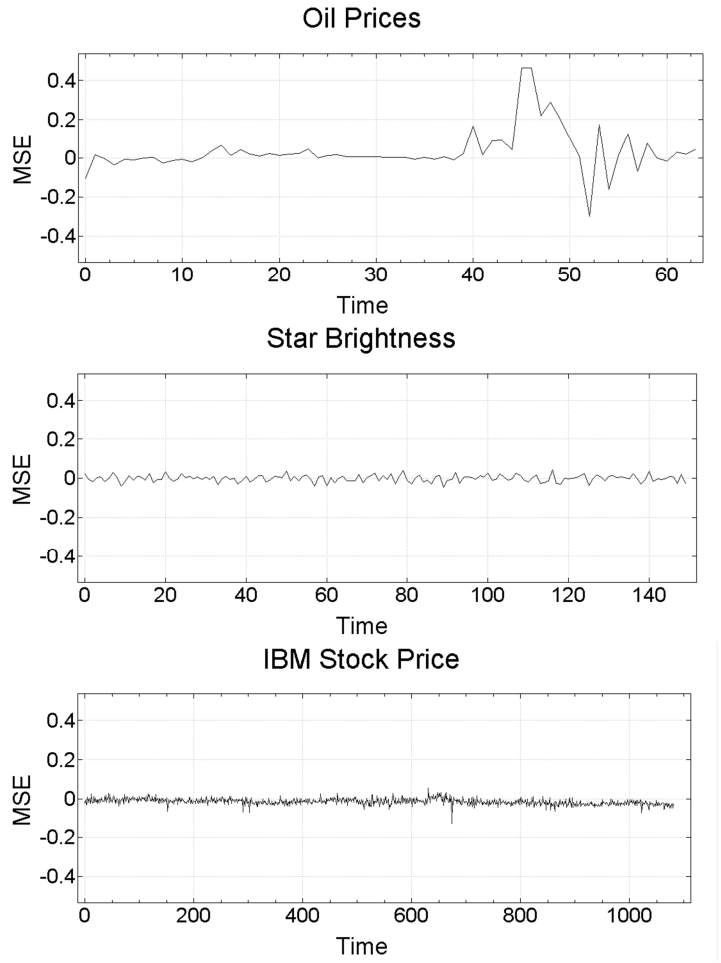


Figure 59. Error time series for Mackey-Glass (t=1749–2999) (a); Lorenz (t=14384–16383) (b); Wolf Sunspot (t=2249–3080) (c); oil prices (t=64–127) (d); star brightness (t=450–599) (e); IBM stock prices (t=2250–3332) (f).

10.3.3. P-STEP PREDICTION RESULTS

The next set of experiments were conducted by performing p-step prediction where we use our one-step predictor architecture to predict several consecutive samples. The optimal parameters of the architecture are selected in the same manner as one-step prediction. We report the results for the best performance of five runs for each architecture since the performance can vary dramatically depending on the initial set of weights. The results for p-step prediction are shown in Table 41.

Table 41. p-step prediction results

Data set	New Proximity Architecture		TSK-NFIS[2]	AR	Neural Network
	Train	Test	Test	Test	Test
Mackey-Glass	1.53e-4 (9.77e-2)	3.77e-3 (0.603)	3.31e-3 (0.565)	1.39e-2 (1.16)	1.39e-2 (1.16)
Lorenz	3.99e-3 (0.426)	7.96e-3 (0.557)	8.49e-3 (0.693)	1.39e-2 (0.887)	1.11e-2 (0.887)
Wolf Sunspot	1.41e-3 (0.470)	2.28e-3 (1.09)	1.89e-3 (0.989)	4.39e-3 (1.51)	3.38e-3 (1.33)

The parameters of the proximity architecture used in producing the results in Table 41 are provided in Table 42.

Table 42. Parameters for p-step prediction

Data set	d	m	q_{order}	c	Epochs	η	Activation Function
Mackey-Glass	50	3	8	2	120	0.025	Sigmoid
Lorenz	20	1.1	10	3	75	0.0005	Sigmoid
Wolf Sunspot	20	3	10	2	400	0.0005	Sigmoid

It is observed that the performance of the proximity architecture is better than the other methods on the Lorenz data set. The performance on the Mackey-Glass and Wolf Sunspot are similar to that of TSK-NFIS and much better than the AR and neural network architecture. It is worth noting that the number of rules with the proximity architecture is significantly smaller than the number of rules for TSK-NFIS on all of these data sets. The number of rules for the Mackey-Glass, Lorenz, and Wolf-Sunspot is significantly higher for the TSK-NFIS architecture where c=7, 41 and 16 respectively. Thus there is much less complexity in the rule-base for the proximity architecture with comparable performance. Also, the p-step test performance of the proximity architecture on some of the data sets, most notably the Wolf Sunspot data set, are very consistent with each run starting with a different random configuration of weights. This is unlike the results reported in [75] where there was significant variability in performance with each run.

[2] The results and associated parameters for TSK-NFIS, AR, and Neural Network models are found in [75]

11. STRUCTURAL MUSICAL SEGMENTATION

The application of partially supervised clustering to structural musical segmentation is the central theme of this chapter. Partially supervised learning will be evaluated in a time series segmentation problem involving the determination of structural transitions in digital music files, e.g. the time at which the verse transitions into the chorus. Sections are devoted to the problem description, formulation of the feature space, and our approach to structural musical segmentation. Finally the proposed approach is compared with several other state-of-the-art approaches developed in the literature.

11.1. PROBLEM FORMULATION AND MOTIVATION

Music contains high-level structural information in the form of phrases or melodies that are often repeated during the song. In Western styles of music, its structural components consist of intro, verse, chorus, bridge, outro, etc. Much can be gained by using computers to learn the location of the boundaries of these structural components in songs through a process called structural musical segmentation and herein simply called musical segmentation. Musical segmentation is depicted in Figure 60.

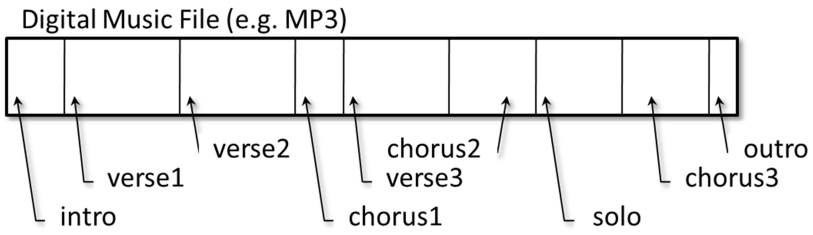


Figure 60. Structural musical segmentation

The motivation of this research is to improve upon the current advancements in musical segmentation: the search for better performing systems is actively on-going. A potential area of improvement is with the modeling and clustering of the features to form the segments. It is assumed that the number of segments is known in this study. The focus is on the determination of clusters which are then interpreted as segments.

The motivation behind musical segmentation are its many potential applications including music thumbnail generation for sampling music from a very large database of music (such as iTunes), fast-forward mechanisms for jumping to the next musical component, music information retrieval, music summarization, and aiding in search or browse features of a large music database. For example, musical segmentation can aid powerful search queries that look for matches of songs with similar choruses. Consider a query for all the songs in the database that have a chorus very similar to "Hey, Jude." In order for such a query to be possible, a structural musical segmentation algorithm is necessary. Also, consider the query find all songs with a "loud" introduction. Another possibility is to do a search on songs with a certain sequence of structure such as intro, verse, verse, chorus, verse, chorus, solo, outro. These types of queries could be used to narrow searches and help listeners or potential purchasers of music to find songs with styles they like. A segmentation algorithm for music is also essential in thumbnail generation since one must find the most representative part of the song, usually the chorus, to present a preview of the song for listeners or potential buyers.

The problem of musical segmentation is far from complete however since most research has only been conducted on popular music databases rather than on other styles of music like classical

music, Asian music and so on. As well, the performance of the musical segmentation systems is not very high yet. In addition, expert labeling of the music is subjective since the structural boundaries between segments are ill-defined. Also, nearly all musical segmentation systems require the number of segments to be defined *a priori* which presents problems in real systems since this is not easily known beforehand.

A short conference paper was recently presented regarding musical segmentation, c.f. [145]. This work is extended in this study by applying proximity fuzzy clustering. To recap, the MPEG7 features are used to train a hidden Markov model (HMM) in order to construct a feature space for clustering (i.e. a time series of HMM state histograms). The clustering algorithm used FCM with an augmented distance function that is the weighted sum of Euclidean distances between the histograms and the time stamps of each sample in the time series, c.f. section 3.6 (FCM-DFS). The results are mixed with some songs providing very good results and other songs below expected performance when compared to a simple reference clustering technique also used in [37].

The proposed segmentation procedure is to use proximity fuzzy clustering with some proximity hints. An overview is provided in Figure 61.

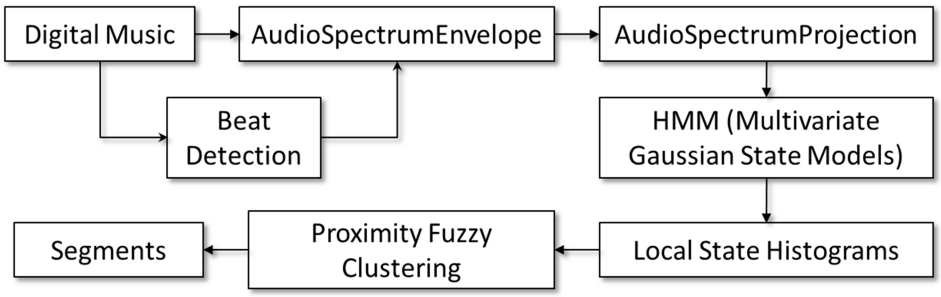


Figure 61. Schematic Overview of Musical Segmentation

The steps are nearly identical to [145] and [37] except where the clustering is replaced by FCM-DFS in [145] and constrained k-means in [37]. Clearly the construction of the feature space is an important research question in itself. The reason for choosing this approach is that the local state histograms (i.e. the feature space) are shown to be well suited for clustering in [37]. Proximity fuzzy clustering improves upon the approach by replacing constrained k-means clustering with partially supervised kernel-based fuzzy clustering.

11.2. The Feature Space

The MPEG-7 standard, well-known for describing media and its digital storage, is employed in the automatic structural segmentation of music. A considerable amount of pre-processing is performed identical to that in [37] and [145]. All music files were converted to mono uncompressed wave format and had a sampling rate of 44.1kHz. A band spacing of $1/8^{th}$ octave is used to construct a series of power log-frequency spectrums called the AudioSpectrumEnvelope audio descriptors as outlined in the MPEG7 standard [111]. It is called a log-frequency spectrum since the sub-bands are logarithmically spaced. The hop size (denoted hopSize) described in the standard was set to the period of the beat detected in the song. The rationale is that each sample (i.e. spectrum) then corresponds to one beat in the song. The software tool Matlab-XM (http://mpeg7.doc.gold.ac.uk/mirror/index.html) was used to detect the beat (hopSize). This is the same software used in [145]. The authors in [37] report that accurately detecting the beat is not essential in the construction of the feature space. The spectrums in the AudioSpectrumEnvelope features are normalized by the L_2-norm. The

dimensionality of each spectrum is then reduced to 20 dimensions with principal components analysis (PCA) thereby producing a feature vector of 21 dimensions called AudioSpectrumProjection in the MPEG-7 standard where the 21st dimension is the relative power of the spectrum at that beat.

The steps in producing the AudioSpectrumProjection features is accomplished according to the MPEG7 standard [111]. A hidden Markov model (HMM) is trained on the AudioSpectrumProjection sequence where multivariate Gaussian distributions are used to model the outputs of the states. The objective in constructing an HMM is to determine a sequence of states in the song with the Viterbi algorithm. Since each sample in AudioSpectrumProjection corresponds to a beat in the music, every beat corresponds to a state produced by the Viterbi algorithm. The authors in [37] convincingly demonstrate that a small number states are unable to capture the high-level structural information of the song. Thus, the number of states used to construct the feature space is n=80. The objective is to use clustering to find patterns in this sequence of states. However, the state sequence lacks any meaningful metrics for clustering. Hence, a local histogram of states is constructed denoted by $\mathbf{h}(t)=[h_1(t), h_2(t),\ldots h_n(t)]$ for $t=1\ldots T$ where n is the number of states, T is the length of the song, and $h_i(t)$ $i=1\ldots n$ is the count of the number of occurrences that state "i" appears in a window of w samples at time t. The size of the window is set to w=11 samples where this value was determined by manually selecting its value and monitoring clustering performance. The histograms are normalized, i.e.

$$\mathbf{h}(t)=\frac{\mathbf{h}(t)}{\sum_{i=1}^{n} h_i(t)} \qquad \textbf{(11-1)}$$

for all values of $t=1\ldots T$.

11.3. SEGMENTATION: A PROXIMITY APPROACH

A novel method of segmenting the local state histograms using proximity clustering is presented in this section. Most segmentation approaches in the literature deal with single variable time series. However, our problem involves the segmentation of a multivariate time series consisting of local histograms $\mathbf{h}(t)$ at time interval t. One way to accomplished this is to employ partially supervised clustering where the temporal proximity of the histograms is included in the clustering as additional domain knowledge. The goal is to segment the multivariate time series of histograms into its structural components in the music, i.e. intro, verse, chorus, bridge, etc. Under this framework, clusters produced by proximity fuzzy clustering form segments in the time series.

A common measure of distance between two histograms is the Jensen-Shannon divergence, i.e.

$$d_{hist}\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)=\frac{1}{2}\left(kl\text{-}div\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)+kl\text{-}div\left(\mathbf{h}(t_2),\mathbf{h}(t_1)\right)\right) \qquad \textbf{(11-2)}$$

where *kl-div* is the asymmetric Kullback-Leibler divergence computed via the expression

$$kl\text{-}div\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)=\sum_{i=1}^{n} h_i(t_1)\log\left(\frac{h_i(t_1)}{h_i(t_2)}\right) \qquad \textbf{(11-3)}$$

The proximity between histograms $\mathbf{h}(t_1)$ and $\mathbf{h}(t_2)$ at times t_1 and t_2 respectively is given by

$$p_{hist}\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)=\exp\left(-\frac{1}{\sigma^2}d_{hist}\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)\right) \qquad \textbf{(11-4)}$$

where σ^2 is the kernel width parameter. One can use this measure of proximity to cluster the histograms with proximity fuzzy clustering in order to form a series of segments in the time series.

Some additional domain knowledge is required to ensure that the clusters contain histograms that are contiguous in time. Fortunately, there is time information associated with each histogram for *t*=1…*T* that can be exploited. One way to measure the temporal proximity of histograms is to measure the proximity of histograms with respect to time; for example, **h**(t) and **h**(t+1) are close in proximity. Therefore the temporal proximity between two histograms can be expressed as

$$p_{time}\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)=\exp\left(-\frac{hopSize^2}{\delta^2}\left|t_1-t_2\right|^2\right) \quad \textbf{(11-5)}$$

where δ^2 is the kernel width of the temporal term and the *hopSize* is the number of seconds per beat (or sample). The two kernel functions can be combined through addition since the sum of two kernel functions is also a kernel function, cf. [64]. Thus, proximity is expressed as a convex sum of two kernel functions in order to maintain the properties of the proximity function, i.e.

$$p\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)=\left(1-\alpha\right)p_{hist}\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)+\alpha\left(p_{time}\left(\mathbf{h}(t_1),\mathbf{h}(t_2)\right)\right) \quad \textbf{(11-6)}$$

where $\alpha\epsilon[0,1]$ is a parameter that denotes the amount of temporal knowledge used. Note that $\alpha=1$ results in no contribution from the histogram proximity function and $\alpha=0$ removes the effect of including temporal information from the clustering.

Finally, proximity fuzzy clustering is invoked using (11-6) as the measure of proximity between histograms.

11.4. EXPERIMENTS

11.4.1. DATA

The data used to evaluate the musical segmentation algorithm are ten songs from Coldplay:

- "Yellow" (S=10)
- "A Message" (S=8)
- "Fix You" (S=10)
- "Swallowed in the Sea" (S=8)
- "Talk" (S=9)
- "A Rush of Blood to the Head" (S=11)
- "In My Place" (S=10)
- "Politik" (S=9)
- "God Put a Smile Upon Your Face" (S=11)
- "The Scientist" (S=9)

The number of known segments is given in parenthesis (see Appendix C for details). Each song was segmented manually by listening to the songs and the words to determine the points of structural change in the music. The manual segmentation was performed by a trained and well-versed musician. The locations of the boundaries between segments described in Appendix C are treated as ground truth in this problem.

11.4.2. EVALUATION CRITERIA

A variety of indices are used to evaluate the performance of musical segmentation. The classification rate is used in these experiments, see section 2.1.11. The boundary f-value is also used and is a direct comparison of the boundaries produced by the segmentation algorithm, i.e. clustering, and the boundaries given by a human expert in Appendix C.

There are two steps to calculating the boundary f-value

a) Determine the location of the boundaries between segments from the partition matrix
b) Compare these boundaries with ground truth

The first step in producing the boundary f-value is to determine boundaries from the membership matrix U=[u_{it}] produced by the clustering for t=1…T and i=1…c (T is the length of song, and c is the number of clusters/segments). This is accomplished by taking the maximum of the membership values according to the expression

$$L_t = \arg \max_{\forall i=1\ldots c} \left(u_{it} \right) \qquad \textbf{(11-7)}$$

where L_t is the index of the cluster, i.e. the cluster label. The boundaries are determined by returning all values of t where $L_t \neq L_{t-1}$ for t=2…T. This expression enforces continuity with respect to time in each segment. The values of t where $L_t \neq L_{t-1}$ are stored in the set of boundaries {b_1, b_2, … b_M} . The number of segments denoted by S is one more than the number of boundaries, i.e. S=M+1. Note that the number of segments "S" does not necessarily equal the number of clusters "c" used in the clustering. When clusters are homogeneous and contiguous in time, S=c and therefore the clusters are identical to the segments. This is a desirable property when using clustering algorithms to segment a time series. However, in some cases S>c, meaning that some clusters were divided into more than one segment in different parts of the song as illustrated in Figure 62.

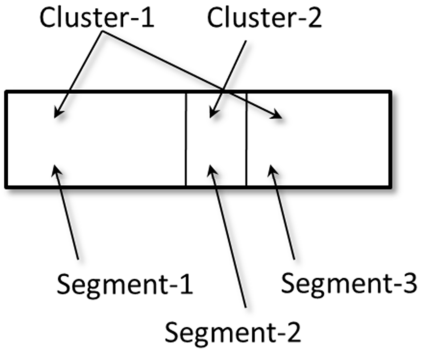


Figure 62. Cluster contiguity example

In this example, although the number of clusters is two, the number of segments is three, i.e. S=3 and c=2. It is also possible for S<c. In this case, one of the clusters is empty.

The second step in computing the f-value is to compare the boundaries with the ones given in Appendix C. This is accomplished by determining n_θ, the number of boundaries that fall with a threshold, θ=*hopSize*w*, of the human expert labeled boundaries in Appendix C. The boundary precision and recall are given by

$$boundary\text{-}precision = \frac{n_\theta}{M} \quad \textbf{(11-8)}$$

$$boundary\text{-}recall = \frac{n_\theta}{B} \quad \textbf{(11-9)}$$

where *B* is the number of boundaries specified in Appendix C. The boundary f-value is

$$boundary\text{-}f\text{-}value = \frac{2(precision)(recall)}{precision + recall} \quad \textbf{(11-10)}$$

11.4.3. RESULTS

The proximity fuzzy clustering algorithm is compared with k-means constrained clustering [37] and FCM-DFS [145] on the ten songs. The number of clusters is set to the number of known segments (see Appendix C). The results are repeated 20 times. The parameters used in the experiments were determined manually by monitoring the performance, i.e. classification rate and boundary f-value, and selecting the parameters with the best performance. The mean and standard deviations of the number of boundaries *M*, classification rate, and boundary f-value (bf-value) are reported in Table 43.

Table 43. Musical segmentation performance

Song	Proximity Fuzzy Clustering			FCM-DFS			K-Means Constrained		
	M	CR (%)	bf-value	*M*	CR (%)	bf-value	*M*	CR (%)	bf-value
1	9.8±1.6	**80.1±2.0**	**0.53±0.09**	**9.2±0.7**	77.5±0.5	0.47±0.02	9.6±1.1	78.8±3.5	**0.53±0.11**
2	**7.0±0.0**	88.8±0.0	**0.62±0.00**	**7.0±0.0**	**89.2±0.0**	**0.62±0.00**	11.3±1.1	78.2±5.0	0.49±0.07
3	**9.0±0.0**	**83.8±2.5**	0.56±0.09	**9.0±0.0**	83.0±0.0	0.50±0.00	9.9±1.6	77.4±5.8	**0.60±0.08**
4	**8.4±0.9**	**87.8±2.6**	**0.55±0.12**	10.9±2.8	83.4±1.6	0.33±0.10	10.5±1.7	73.2±5.6	0.42±0.12
5	**8.0±0.0**	**78.7±0.9**	0.37±0.05	**8.0±0.0**	77.4±3.7	0.27±0.14	11.0±1.2	74.0±4.4	**0.44±0.07**
6	10.4±1.0	**89.1±1.2**	**0.63±0.07**	**10.2±0.9**	83.6±0.1	0.32±0.07	14.0±1.6	88.0±2.3	0.58±0.08
7	**9.0±0.0**	**84.6±1.7**	**0.77±0.06**	**9.0±0.0**	82.0±1.4	0.68±0.05	10.1±0.8	82.4±2.2	0.73±0.04
8	8.8±1.5	84.0±1.9	**0.73±0.06**	**8.0±0.0**	**88.3±1.0**	0.65±0.05	15.1±1.3	71.2±3.6	0.64±0.06
9	**10.1±0.2**	**87.0±2.6**	**0.93±0.05**	11.7±0.7	83.9±2.6	0.78±0.05	11.9±1.5	76.7±4.5	0.82±0.08
10	8.4±1.2	73.8±1.1	0.40±0.02	**8.0±0.0**	**74.7±0.0**	0.27±0.00	8.7±1.0	68.6±5.1	**0.41±0.08**

The best performances are highlighted in bold. Proximity fuzzy clustering outperforms the other algorithms in nearly every case. The parameters are reported in Table 44.

Table 44. Musical segmentation parameters

Song	Proximity Fuzzy Clustering				FCM-DFS		K-Means Constrained		
	m	σ^2	δ^2	α	*m*	α	d_{ML}	σ^2	λ
1	1.4	350	400	0.025	1.8	0.1	16	2	0.1
2	2.0	150	750	0.075	2.0	0.15	16	1	0.1
3	1.2	200	550	0.05	2.0	0.1	16	2	0.1
4	2.0	250	750	0.075	1.5	0.05	16	2	0.1
5	1.3	275	450	0.075	1.4	0.2	24	2	0.2
6	1.4	275	450	0.03	1.5	0.1	16	2	0.1
7	1.2	200	600	0.05	1.4	0.15	16	1	0.15
8	1.4	275	450	0.05	1.4	0.15	16	1	0.15
9	1.4	250	450	0.025	1.5	0.05	16	2	0.15
10	1.3	275	450	0.025	1.7	0.05	16	3	0.15

The neighborhood size parameter d_{ML} from K-Means constrained clustering is set to 16 for most of the songs as suggested in the author's work, cf. [122]. Several observations about the experiments are noted. The temporal information was very important for the song "Talk" (song 5). FCM-DFS had difficulty accurately detecting the boundaries compared with the other two methods. The K-Means constrained approach generally produced more segments, particularly

on the song "*A Rush of Blood to the Head*" (song 6). Overall, proximity fuzzy clustering performs very well.

11.5. CONCLUSIONS

The proximity fuzzy clustering algorithm is shown to perform exceedingly well on segmenting this collection of music from various albums of the band Coldplay. The diverse sounds in the music is particularly challenging for segmentation algorithms. Despite the challenge, proximity fuzzy clustering is able to achieve high accuracy in both the classification rate and boundary f-value while keeping the number of segments low. For example, in the some "*God Put a Smile Upon Your Face*," (song 9) proximity fuzzy clustering is able to very accurately determine the boundaries. K-Means constrained clustering produces more segments than the other methods particularly on the song "*A Rush of Blood to the Head*" (song 6). Proximity fuzzy clustering improves the segmentation performance in nearly all the songs.

12. CONCLUSIONS

The relatively new kernel-based fuzzy clustering algorithm was evaluated in our research. It was demonstrated that non-spherical cluster shapes are possible and that the choice of the kernel function and its parameters is critical in accurate performance. Unfortunately, there are no suitable frameworks provided in the literature for optimizing the structure and the parameters of the kernel function. Our results led us to develop and analyze a comprehensive framework in partially supervised clustering where some additional domain knowledge, either in the form of proximity hints or class labels, is used to improve the clustering performance, achieve non-spherical cluster shapes, and optimize the kernel function.

The proximity fuzzy clustering algorithm is a novel approach to clustering where the kernel function is embedded in the concept of proximity. We have evaluated proximity fuzzy clustering on several synthetic and widely used real-world data sets in order to better understand the benefits of the framework in its ability to automatically determine the optimal kernel parameters and to produce non-spherical cluster shapes. We then successfully applied and demonstrated the benefits procured through this relationship of kernels and proximity through several different applications including

- structural musical segmentation
- time series clustering and prediction
- graph clustering

For example, with time series clustering and prediction, the feature space is poorly suited for the common assumption that the Euclidean metric is a suitable distance function. Thus, we are able to demonstrate that using proximity to measure the similarity between time series produces better performance and simplified models. We have shown that there is a significant benefit to partially supervised learning and proximity fuzzy clustering when compared with traditional clustering approaches. The problem of clustering a set of graphs is an important problem that can be solved using proximity fuzzy clustering and relational FCM. Our approach is successfully demonstrated using a case study in software engineering by clustering variability models – a task that not been done yet in the literature.

Our research is significant in both theoretic and practical advances as our proposed framework

- allows for a variety of non-hyper-spherical cluster geometries
- reduces the number of clusters
- improves the accuracy of the clustering performance
- provides meaningful mechanisms for determining the optimal set of parameters
- provides a mechanism for active learning
- reconciles proximity from multiple difference sources

A serious limiting factor in partially supervised learning is that there are many sources and forms of domain knowledge. Thus, the theoretical developments in partially supervised learning is still very much incomplete since a number of open research problems remain including

- selecting the proximity function (structural optimization),
- obtaining the proximity labels in partially supervised learning,
- ensuring the labels given by a human expert are of high quality,
- developing new frameworks for other forms of domain knowledge

We envision that while partially supervised clustering can offer substantial advantages over the existing “traditional” clustering, it opens up interesting new applications in the future that could

otherwise not be contemplated. The music recommendation system described in the introduction (see Figure 2), is an example of a system that is not realizable in a fully supervised learning environment due to the diversity of music and the many songs that must be labeled. Therefore, the application of partially supervised learning to the problem of recommending music is an interesting future application of this research.

REFERENCES

1. Abdallah S, Theory and evaluation of a Bayesian music structure extractor, Proc. ISMIR, 2005, pp. 420-425.
2. Ajmera J, McCowan I, Bourlard H, Speech/music segmentation using entropy and dynamism features in a HMM classification framework, Speech Communication, **40**, 2003, pp. 351-363.
3. Alon J, Sclaroff S, Kollios G, Pavlovic V, Discovering clusters in motion time-series data, Proc. of the 2003 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, 2003, pp.375-381.
4. Amari S, Wu S, Improving support vector machine classifiers by modifying kernel functions, Neural Networks, **12**, 1999, pp.783-789.
5. Asuncion A, Newman DJ, UCI Machine Learning Repository, Irvine, CA: University of California, School of Information and Computer Science, 2007 [online] Available: http://archive.ics.uci.edu/beta/.
6. Babuska R, van der Veen PJ, Kaymak U, Improved covariance estimation for Gustafson-Kessel clustering, Proc. of the 2002 IEEE Int. Conf. on Fuzzy Systems, 2002 pp.1081-1085.
7. Baghshah MS, Shouraki SB, Kernel-based metric learning for semi-supervised clustering, Neurocomputing, In Press, Available online 23 December 2009, DOI: 10.1016/j.neucom.2009.12.009.
8. Bagnall AJ, Janacek GJ, Clustering time series from ARMA models with clipped data, Machine Learning, vol. **58**(2), 2005, pp.151-178.
9. Bagnall AJ, Janacek G, Iglesia B, Zhang M, Clustering time series from mixture polynomial models with discretised data, Proc. of the 2nd Australasian Data Mining Workshop, 2003.
10. Baragona R, Genetic algorithms and cross-correlation clustering of time series, 2000, http://citeseer.ist.pse.edu/baragona00genetic.html, Accessed on August 8th, 2007.
11. Bargiela A, Pedrycz W, Granulation of Temporal Data: A Global View on Time Series, in Proc. of the 22nd Int. Conf. of the North American Fuzzy Information Processing Society, pp.191-196, Chicago, July, 2003.
12. Basalto N, Bellotti R, Carlo FD, Facchi P, Pantaleo E, Pascazio S, Hausdorff clustering of financial time series, Physica A, **379**, 2007, pp.635-644.
13. Basu S, Banerjee A, Mooney RJ, Active semi-supervision for pairwise constrained clustering, Proc SLAM Int. Conf. on Data Mining, 2004, pp. 333-344.
14. Ben-Hur A, Horn D, Siegelmann HT, Vapnik V, Support vector clustering, J. of Machine Learning Research, **2**, 2001, pp.125-137.
15. Benoudjit N, Werleysen M, On the kernel widths in radial-basis function networks, Neural Processing Letters, vol. 18, 2003, pp. 139-154.
16. Bensaid AM, Hall LO, Bezdek JC, Clarke LP, Partially supervised clustering for image segmentation, Pattern Recognition, **29**(5), 1996, pp. 859-871.
17. Berkhin P, A survey of clustering data mining techniques, Grouping Multidimensional Data, Springer, Berlin, Heidelberg, 2006, pp. 25-71.
18. Berthold MR, Ortolani M, Patterson D, Hoppner F, Callan O, Hofer H, Fuzzy information granules in time series data, International Journal of Intelligent Systems, **19**, 2004, pp.607-618.

19. Bezdek JC, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum, New York, 1981.

20. Bezdek JC, Coray C, Gunderson R, Watson J, Detection and characterization of clustering substructure I. Linear structure: Fuzzy c-Lines, Journal on Applied Mathematics, vol. 40(2), 1981, pp. 339-357.

21. Bezdek JC, Coray C, Gunderson R, Watson J, Detection and characterization of cluster substructure II. Fuzzy c-Varieties and convex combinations thereof, Journal on Applied Mathematics, vol. 40(2), 1981, pp. 358-372.

22. Bi LP, Huang H, Zheng ZY, Song HT, New heuristic for determination Gaussian kernel's parameters, Proc. 4th Int. Conf. on Machine Learning and Cybernetics, Gaungzhou, 2005, pp. 4299-4304.

23. Bicego M, Murino V, Figueiredo MAT, Similarity-based clustering of sequences using hidden Markov models, **2734**, 2003, pp.95-104.

24. Bisdorff R, Electre-like clustering from a pairwise fuzzy proximity index, European Journal of Operational Research, **138**, 2002, pp. 320-331.

25. Boratyn GM, Datta S, Datta S, Biologically supervised hierarchical clustering algorithms for gene expression data, in Proc. 28th IEEE Int. Conf. Engineering in Medicine and Biology Society, 2006, pp. 5515-5518.

26. Bouchachia A, Pedrycz W, Enhancement of fuzzy clustering by mechanism of partial supervision, Fuzzy Sets and Systems, **157**, 2006, pp.1733-1759.

27. Borer S, Gerstner W, A new kernel clustering algorithm, Proc. of the 9th Int. Conf. on Neural Information Processing, **5**, 2002, pp.2527-2531.

28. Brandes U, Gaertler M, Wagner D, Engineering graph clustering: models and experimental evaluation, ACM Journal of Experimental Algorithmics, vol. 12, 2008, doi:10.1145/1227161.1227162.

29. Brouwer RK, A method of relational fuzzy clustering based on producing feature vectors using FastMap, Information Sciences, **179**, 2009, pp. 3561-3582.

30. Burges CJC, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery, vol. 2, 1998, pp. 121-167.

31. Capitani P, Ciaccia P, Warping the time on data streams, Data & Knowledge Engineering, **62**, 2007, pp.438-458.

32. Casey M, General sound classification and similarity in MPEG-7, Organised Sound, **6**(2), 2001, pp. 153-164.

33. Ceccarelli M, Maratea A, Improving fuzzy clustering of biological data by metric learning with side information, Int. Journal of Approximate Reasoning, doi:10.1016/j.ijar.2007.03.008, 2007

34. Chang H, Yeung, DY, Cheung WK, Relaxational metric adaptation and its application to semi-supervised clustering and content-based image retrieval, Pattern Recognition, **39**, 2006, pp. 1905-1917.

35. Chen B, Liu H, Bao Z, Optimizing the data-dependent kernel under a unified kernel optimization framework, Pattern Recognition, **41**, 2008, pp. 2107-2119.

36. Cheng CH, Cheng GW, Wang JW, Multi-attribute fuzzy time series method based on fuzzy clustering, Expert Systems with Applications, 2007, doi:10.1016/j.eswa.2006.12.013

37. Chen JR, Making subsequence time series clustering meaningful, Proc. of the 5th IEEE Int. Conf. on Data Mining, 2005, pp.114-121.

38. Chen WB, Zhang C, A robust method for biological sequence clustering, IEEE Int. Conf. on Information reuse and Integration, 2006, pp.286-291.

39. Chen Y, Reilly KD, Sprague AP, Guan Z, SEQOPTICS: a protein sequence clustering method, Proc. of the 1st Int. Multi-Sym. on Computer and Computational Sciences, 2006, pp.69-75.

40. Chiang JH, Hao PY, A new kernel-based fuzzy clustering approach: support vector clustering with cell growing, IEEE Transactions of Fuzzy Systems, **11**(4), 2003, pp.518-527.

41. Chiu B, Keogh E, Lonardi S, Probabilistic discovery of time series motifs, Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp.493-498.

42. Cristianini N, Kandola J, Elisseeff A, Taylor JS, On kernel-target alignment, Advances in Neural Information Processing Systems, vol. 14, 2002, pp. 367-373.

43. Daniels K, Giraud-Carrier C, Learning the threshold in hierarchical agglomerative clustering, in Proc. 5th Int. Conf. on Machine Learning and Applications, 2006, pp. 270-278.

44. Davies MEP, Plumbley MD, Beat tracking with a two state model, Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, **3**, 2005, pp. 241-244.

45. Davis PJ, Leonhard Euler's Integral: a historical profile of the Gamma function, American Mathematical Monthly, vol. 66, 1959, pp. 849-869.

46. Davidson I, Ravi SS, Agglomerative hierarchical clustering with constraints: theoretical and empirical results, Knowledge Discovery in Databases, **3721**, 2005, pp. 59-70.

47. Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R, Indexing by latent semantic analysis, Journal of the American Society for Information Science, vol. 41, 1990, pp. 391-407.

48. Delacourt P, Wellekens CJ, DISTBIC: a speaker-based segmentation for audio data indexing, Speech Communications, **32**, 2000, pp. 111-126.

49. Denoeux T, Masson MH, EVCLUS: Evidential clustering of proximity data, IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, vol. 34(1), 2004, pp. 95-108.

50. Denton A, Density-based clustering of time series subsequences, In Proc. 3rd Workshop on Mining Temporal and Sequential Data in conjunction with The 10th ACM SIGMKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004.

51. Dhillon IS, Guan Y, Kulis B, Kernel k-means, spectral clustering and normalized cuts, Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 551-556.

52. Dorado A, Pedrycz W, Izquierdo E, User-driven fuzzy clustering: on the road to semantic classification, Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, **3641**, 2005, pp. 421-430.

53. Duda RO, Hart PE, Pattern Classification and Scene Analysis, John Wiley & Sons: New York, 1973.

54. Eberhart R, Kennedy J, A new optimizer using particle swarm theory, Proc. 6th Symposium on Micro Machine and Human Science, 1995, 39-43.

55. Eichinger F, Bohm K, Huber M, Mining edge-weighted call graphs to localize software bugs, Lecture Notes In Artificial Intelligence, vol. 5211, 2008, pp. 333-348.

56. Fan J, Xie W, Some notes on similarity measure and proximity measure, Fuzzy Sets and Systems, vol. 101, 1999, pp. 403-412.

57. Figueiredo MAT, Jain AK, Unsupervised learning of finite mixture models, IEEE Transactions on Pattern Analysis and Machine Intelligence, **24**(3), pp.381-396, 2002.

58. Filippone M, Camastra F, Masulli F, Rovetta S, A survey of kernel and spectral methods for clustering, Pattern Recognition, vol. 41, 2008, pp. 176-190.

59. Fred ALN, Jain AK. Data clustering using evidence accumulation. In Proc. Int. Conf. Pattern Recognition, 2002, pp. 276-280.

60. Gacek A, Pedyrcz W, A genetic segmentation of ECG signals, IEEE Trans. On Biomedical Engineering, **50**(10), 2003, pp.1203-1208.

61. Gao S, Maddage NC, Lee CH, A hidden Markov model based approach to music segmentation and identification, Proc. IEEE Int. Conf. Information, Communications and Signal Processing, **3**, 2003, pp. 1576-1580.

62. Gao X, Xiao B, Tao D, Li X, A survey of graph edit distance, Journal of Pattern Analysis & Applications, vol. 13(1), 2010, pp. 113-129.

63. Gamma E, Helm R, Johnson R, Vlissides J, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

64. Genton MG, Classes of kernels for machine learning: a statistics perspective, Journal of Machine Learning Research, vol. 2, 2001, pp. 229-312.

65. Girolami M, Mercer kernel-based clustering in feature space, IEEE Transactions on Neural Networks, **10**(5), 1999, pp.1000-1017.

66. Golay X, Kollia S, Stoll G, Meier D, Valavanis A, Boesiger P, A new correlation-based fuzzy logic clustering algorithm for fMRI, Magnetic Resonance in Medicine, **40**(2), 1998, pp.249-260.

67. Goldin D, Mardales R, Nagy G, In search of meaning for time series subsequence clustering: matching algorithms based on a new distance measure, Proc. of the Int. Conf. on Information and Knowledge Management, Arlington, Virginia, 2006, pp.347-356.

68. Goodwin MM, Laroche J, A dynamic programming approach to audio segmentation and speech/music discrimination, Proc IEEE Int. Conf. Acoustics, Speech and Signal Processing, **4**, 2004, pp. 309-312.

69. Gordon AD, A review of hierarchical classification, Journal of the Royal Statistical Society, Series A (General), **150**(2), 1987, pp. 119-137.

70. Gori M, Maggini M, Sarti L, Exact and approximate graph matching using random walks, IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 27 (7), 2005, pp. 1100-1111.

71. Goto M, A chorus detecting method for musical audio signals. Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 2003, pp. 437-440.

72. Goutte C, Toft P, Rostrup E, Nielsen FA, Hansen LK, On clustering fMRI time series, NeuroImage, **9**, 1999, pp.298-310.

73. Graves D., Pedrycz W., Discovering structure in labeled data, proc. Fuzzy Information Processing Society, NAFIPS 2008, 2008, pp.1-6.

74. Graves D, Pedrycz W, Fuzzy C-Means, Gustafson-Kessel FCM, and Kernel-Based FCM: A Comparative Study, in Advances in Soft Computing, **41**, 2007, pp. 140-149.

75. Graves D, Pedrycz W, Fuzzy prediction architecture using recurrent neural networks, Neurocomputing, doi:10.1016/j.neucom.2008.07.009, 2008.

76. Graves D, Pedrycz W, Kernel-based fuzzy clustering and fuzzy clustering: a comparative experimental study, Fuzzy Sets and Systems, **161**, 2010, pp. 522-543.

77. Graves D, Pedrycz W, Performance of kernel-based fuzzy clustering, Electronic Letters, **43**(25), 2007, pp. 1445-1446.

78. Graves D, Pedrycz W, Structural segmentation of music with fuzzy clustering, In Proc. Acoustics Week in Canada, Vancouver, BC. Canadian Acoustics, October 2008.

79. Grira N, Crucianu M, Boujemaa N, Active semi-supervised clustering for image database categorization, Pattern Recognition, vol. 41, 2008, pp. 1834-1844.

80. Grira N, Crucianu M, Boujemaa N, Semi-supervised fuzzy clustering with pairwise-constrained competitive agglomeration, Proc. IEEE Int. Conf. on Fuzzy Systems, 2005, pp. 867-872.

81. Guenoche A, Hansen P, Jaumard B, Efficient algorithms for divisive hierarchical clustering with the diameter criterion, Journal of Classification, vol. 8, 1991, pp. 5-30.

82. Guha S, Meyerson A, Mishra N, Motwani R, Clustering data streams: theory and practice, IEEE Transactions on Knowledge and Data Engineering, **15**(3), 2003, pp.515-528.

83. Gunter S, Bunke H. Validation indices for graph clustering. Pattern Recognition Letters, vol. 24, 2003, pp. 1107-1113.

84. Gustafson DE, Kessel WC, Fuzzy clustering with a fuzzy covariance matrix, IEEE Conf. Decision Control inc. 17th sym. Adaptive Processes, 1978, pp. 761-766.

85. Hathaway RJ, Bezdek JC, Nerf c-means: Non-Euclidean relational fuzzy clustering, Pattern Recognition, 27, 1994, pp. 429-437

86. Hathaway RJ, Davenport JW, Bezdek JC, Relational duals of the c-means clustering algorithms, Pattern Recognition, **22**(2), 1989, pp. 205-212.

87. Harms SK, Deogun J, Tadesse T, Discovering sequential association rules with constraints and time lags in multiple sequences, Proc. of the 13th Int. Sym. on Foundations of Intelligent Systems, **2366**, 2002, pp.432-441.

88. Herbrich, Learning kernel classifiers, MIT Press: Cambridge, Massachusetts, 2002.

89. Hofmann T, Bunmann JM, Active data clustering, Proc. Conf. on Advances in Neural Information Processing Systems, 1998, pp. 528-534.

90. Hofmann T, Scholkopf B, Smola AJ, Kernel methods in machine learning, The Annals of Statistics, vol. 36(3), 2008, pp. 1171-1220.

91. Hoi SCH, Jin R, Lyu MR, Batch mode active learning with applications to text categorization and image retrieval, Proc IEEE Trans on Knowledge and Data Engineering, vol. 21(9), 2009, pp. 1233-1248.

92. Horn D, Clustering via Hilbert space, Physica A, **302**, 2001, pp.70-79.

93. Horvath T, Ramon J, Efficient frequent connected subgraph mining in graphs of bounded tree-width, Theoretical Computer Science, vol. 411, 2010, pp. 2784-2797.

94. Hyndman RJ, Time Series Data Library, http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/, (Accessed on May 15, 2007).

95. Jain AK, Murty MN, Flynn PJ, Data clustering: a review, ACM Computing Surveys, **31**(3), 1999, pp. 264-323.

96. Jang JSR, ANFIS: adaptive network-based fuzzy inference system, IEEE Transactions on Systems, Man and Cybernetics, vol. 23(3), 1993, pp. 24-38.

97. Jiang D, Pei J, Zhang A, DHC: A density-based hierarchical clustering method for time series gene expression data, Proc. of the 3rd IEEE Sym. on BioInformatics and BioEngineering, 2003, pp.393-400.

98. Junkui L, Yuanzhen W, Xinping L, LB HUST: a symmetrical boundary distance for clustering time series, IEEE Int. Conf. on Information Technology, 2006, pp.203-208.

99. Kadous MW, Temporal classification: extending the classification paradigm to multivariate time series, PhD thesis, School of Computer Science & Engineering, University of New South Wales, 2002.

100. Kalpakis K, Gada D, Puttagunta V, Distance measures for effective clustering of ARIMA time-series, In Proc. Of the 2001 IEEE Int. Conf. on Data Mining, San Jose, CA, 2001, pp.273-280.

101. Kang K, Cohen S, Hess J, Novak W, Peterson AS, Feature Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute, Carnegie Mellon University, 1990, CMU/SEI-90-TR-21, ESD-90-TR-222.

102. Karray FO, de Silva C, Soft Computing and Intelligent Systems Design, Pearson Addison Wesley, 2004.

103. Kennedy J, Eberhart R, Particle swarm optimization, Proc. of IEEE Int. Conf. on Neural Networks, vol. IV, 1995, pp. 1942-1948.

104. Keogh E, Kasetty S, On the need for time series data mining benchmark: a survey and empirical demonstration, Proc. of the 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Edmonton, Alberta, 2002, pp.102-111.

105. Keogh E, Xi X, Wei L, Ratanamahatana CA, The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/~eamonn/time_series_data/, 2006, Accessed on June 13, 2007.

106. Khoo KG, Suganthan PN, Structural pattern recognition using genetic algorithms with specialized operators, IEEE Trans. On Systems, Man and Cybernetics – Part B: Cybernetics, vol. 33(1), 2003, pp.156-165.

107. Kim DW, Lee KY, Lee D, Lee KH, Evaluation of the performance of clustering algorithms kernel-induced feature space, Pattern Recognition, **38,** 2005, pp.607-611.

108. Kim HG, Berdahl E, Moreau N, Sikora T, Speaker recognition using MPEG-7 descriptors, 8th European Conf. on Speech Communication and Technology, 2003, 489-492.

109. Kim HG, Burred JJ, Sikora T, How efficient is MPEG-7 for general sound recognition?, 25th Int. AES Conf. Metadata for Audio, 2004.

110. Kim HG, Haller M, Sikora T, Comparison of MPEG-7 basis projection features and MFCC applied to robust speaker recognition, in ODYS-2004, pp. 275-278.

111. Kim HG, Moreau N, Sikora T, MPEG-7 Audio and Beyond, Audio Content Indexing and Retrieval, John Wiley and Sons: Chichester, 2005.

112. Kim HG, Sikora T, Comparison of MPEG-7 audio spectrum projection features and MFCC applied to speaker recognition, sound classification and audio segmentation, Proc. of Int. IEEE Conf. on Acoustics, Speech and Signal Processing, **5**, 2004, pp. 925-928.

113. Kienzle J, Guelfi N, Mustafiz S, Crisis management systems: a case study for aspect-oriented modeling, 2009, http://www.cs.mcgill.ca/~joerg/taosd/TAOSD/TAOSD_files/AOM_Case_Study.pdf *(Retrieved 08/03/2010)*

114. Klawonn F, Kruse R, Constructing a fuzzy controller from data, Fuzzy Sets and Systems, **85**, 1997, pp.177-193.

115. Kotti M, Moschou V, Kotropoulos C, Speaker segmentation and clustering, Signal Processing, **88**, 2008, pp.1091-1124.

116. Krishnapuram R, Kim J, A note on the Gustafson-Kessel and adaptive fuzzy clustering algorithms, IEEE Transactions on Fuzzy Systems, **7**(4), 1999, pp.453-461.

117. Kulis B, Basu S, Dhillon I, Mooney R, Semi-supervised Graph Clustering: A Kernel Approach. Proc. 22nd Int. Conf. on Machine Learning, 2005.

118. Kuramochi M, Karypis G, An efficient algorithm for discovering frequent sub-graphs, IEEE Trans. On Knowledge and Data Engineering, vol. 16(9), 2004, pp. 1038-1051.

119. Lagerholm M, Peterson C, Braccini G, Edenbrandt L, Sornmo L, Clustering ECG complexes using hermite functions and self-organizing maps, IEEE Transactions on Biomedical Engineering, **47**(7), 2000, pp.838-848.

120. Lanckriet GRG, Cristianini N, Bartlett P, Ghaoui LE, Jordan MI, Learning the kernel matrix with semidefinite programming, Journal of Machine Learning Research, vol. 5, 2004, pp. 27-72.

121. Levenshtein V, Binary codes capable of correcting spurious insertions and deletions of ones. Probl. Inf. Transmission, vol. 1, 1965, pp. 8–17.

122. Levy M, Sandler M, Structural segmentation of musical audio by constrained clustering, IEEE Transactions on Audio, Speech, and Language Processing, **16**(2), 2008, pp. 318-326.

123. Levy M, Sandler M, Casey M, Extraction of high-level musical structure from audio data and its application to thumbnail generation, Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, **5**, pp. 14-19, 2006.

124. Lewis DD, Catlett J, Heterogeneous uncertainty sampling for supervised learning, Proc. Int. Conf. on Machine Learning, 1994, pp. 148-156.

125. Li H, Zhang K, Jiang T, Minimum entropy clustering and applications to gene expression analysis, Proc. IEEE Computational Systems Bioinformatics Conference, 2004, pp. 142-151.

126. Lu L, Jiang H, Zhang HJ, A robust audio classification and segmentation method, Proc. ACM Int. Conf. Multimedia, 2001, pp. 203-211.

127. Li M, Sethi IK, Confidence-based active learning, IEE Trans on Pattern Analysis and Machine Intelligence, vol. 28(8), 2006, pp. 1251-1261.

128. Li T, Ma S, Ogihara M, Entropy-based criterion in categorical clustering, ACM International Conference Proceeding Series, Proc. 21st Int. Conf. on Machine Learning, **69**, 2004, pp.68-75.

129. Liao TW, A clustering procedure for exploratory mining of vector time series, Pattern Recognition, **40**, 2007, pp.2550-2562.

130. Liao TW, Clustering of time series data – a survey, Pattern Recognition, **38**, 2005, pp.1857-1874.

131. Lin J, Keogh E, Truppel W, Clustering of streaming time series is meaningless, 8th ACM SIGMOD Workshop on Research Issues in Data Ming and Knowledge Discovery 2003, pp.56-65.

132. Lin L, Liu X, Zhu SC, Layered graph matching with composite cluster sampling, IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 32 (8), 2010, pp. 1426-1442.

133. Lin J, Vlachos M, Keogh E, Gunopulos D, Iterative incremental clustering of time series, In Proc. Of 9th Int. Conf. on Extending Database Technology, Crete, Greece, 2004, pp.106-122.

134. Liu C, Gabor-based kernel PCA with factional power polynomial models for face recognition, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 26(5), 2004, pp. 572-581.

135. Liu H, Huang ST, Evolutionary semi-supervised fuzzy clustering, Pattern Recognition Letters, **24**, 2003, pp. 3105-3113.

136. Lu L, Li SZ, Zhang HJ, Content-based audio segmentation using support vector machines, Proc. of IEEE Int. Conf. on Multimedia and Expo, 2001, pp. 956-959.

137. Luo B, Hancock ER, Structural graph matching using the EM algorithm and singular value decomposition, IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 23 (10), 2001, pp. 1120-1136.

138. Luxburg U von, A tutorial on spectral clustering, Statistics and Computing, vol. 17 (4), 2007, pp. 395-416.

139. Mandic DP, Chambers JA, Recurrent Neural Networks for Prediction: Learning Algorithms and Architectures and Stability, John Wiley & Sons, Chichester, 2001.

140. Mansfield JR, Sowa MG, Scarth GB, Somorjai RL, Mantsch HH, Fuzzy c-means clustering and principal component analysis of time series from near-infrared imaging of forearm ischemia, Computerized Medical Imaging and Graphics, **21**(5), 1997, pp.299-308.

141. Maqbool O, Babri HA, Hierarchical clustering for software architecture recovery, IEEE Transactions on Software Engineering, **33**(11), 2007, pp. 759-780.

142. Maraziotis IA, Dragomir A, Bezerianos A, Semi supervised fuzzy clustering networks for constrained analysis of time-series gene expression data, Int. Conf. on Artificial Neural Networks, Athens, Greece, 2006, pp.818-826.

143. Masson MH, Denoeux T, RECM: relational evidential c-means algorithm, Pattern Recognition Letters, **30**, 2009, pp. 1015-1026.

144. Maulik U, Bandyopadhyay S, Performance evaluation of some clustering algorithms and validity indices, IEEE Transactions on Pattern Analysis and Machine Intelligence, **24**(12), 2002, pp. 1650-1654.

145. McCallum AK, Nigam K, Employing EM and pool-based active learning for text classification, Proc. Int. Conf. on Machine Learning (ICML), 1998, pp. 359-367.

146. Moller-Levet CS, Kalwonn F, Cho KH, Wolkenhauer O, Fuzzy clustering of short time-series and unevenly distributed sampling points, The 5th Int. Sym. On Intelligent Data Analysis, Berlin, Germany, 2003, pp.330-340.

147. Moller-Levet CS, Kalwonn F, Cho KH, Yin H, Wolkenhauer O, Clustering of unevenly sampled gene expression time-series data, Fuzzy Sets and Systems, **152**, 2005, pp.49-66.

148. Muller KR, Mika S, Ratsch G, Tsuda K, Scholkopf B, An introduction to kernel-based learning algorithms, IEEE Trans. Neural Networks, **12**(2), 2001, pp. 181-201.

149. Neuhaus M, Bunke H, Edit distance-based kernel functions for structural pattern classification, Pattern Recognition, vol. 39, 2006, pp. 1852-1863.

150. Neuhaus M, Bunke H, Self-organizing maps for learning the edit distance in graph matching, IEEE Trans. On Systems, Man and Cybernetics – Part B: Cybernetics, vol. 35(3), 2005, pp. 503-415.

151. Noland K, Sandler M, Key estimation using a hidden Markov model, Proc. Int. Society for Music Information Retrieval, Victoria, BC, 2006, pp. 121-126.

152. Oates T, Identifying distinctive subsequences in multivariate time series by clustering, Proc. of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp.322-326.

153. Oates T, Firoiu L, Cohen PR, Clustering time series with HMM and DTW, Proc. in Int. Joint Conf. in Artificial Intelligence, *Workshop* on. *Neural*, *Symbolic*, and Reinforcement Methods for Sequence Learning, 1999, pp.17-21.

154. Ong BS, Herrera P, Semantic segmentation of music audio contents, Proc. Int. Conf. Computer Music, Barcelona, 2005.

155. Papdimitriou S, Sun J, Faloutsos C, Streaming pattern discovery in multiple time-series, Proc. of the 31st Very Large Data Base Conf., Trondheim, Norway, 2005, pp.697-708.

156. Pedrycz W, A dynamic data granulation through adjustable fuzzy clustering, Pattern Recognition Letters, **29**, 2008, pp.2059-2066.

157. Pedrycz W, Algorithms of Fuzzy clustering with partial supervision, Pattern Recognition Letters, **3**, 1985, pp. 13-20.

158. Pedrycz W, Computational Intelligence: An Introduction, CRC Press LLC, 1998.

159. Pedrycz W, Conditional fuzzy c-means, Pattern Recognition Letters, **17**, pp.625-631, 1996.

160. Pedrycz W, Conditional fuzzy clustering in the design of radial basis function neural networks, IEEE Transactions on Neural Networks, **9**(4), pp. 601-612, 1998.

161. Pedrycz W, Fuzzy clustering with a knowledge-based guidance, Pattern Recognition Letters, **25**, 2004, pp. 469-480.

162. Pedrycz W, Fuzzy sets in pattern recognition: methodology and methods, Pattern Recognition, **23**(1/2), 1990, pp. 121-146.

163. Pedrycz W, Knowledge-based Clustering, J. Wiley, Hoboken, NJ, 2005.

164. Pedrycz W, Hirota K, A consensus-driven fuzzy clustering, Pattern Recognition Letters, **29**, 2008, pp. 1333-1343.

165. Pedrycz W, Loia V, Senatore S, P-FCM: a proximity – based fuzzy clustering, Fuzzy Sets and Systems, **148**, 2004, pp. 21-41.

166. Pedrycz W, Waletzky J, Fuzzy clustering with partial supervision, IEEE Trans. Systems, Man and Cybernetics, **27**(5), 1997, pp. 787-795.

167. Peeters G, Deriving musical structures from signal analysis for music audio summary generation: “sequence” and state approach, in CMMR (LNCS2771) Lecture Notes in Computer Science, New York: Springer-Verlag, 2003, pp. 142-165.

168. Popescu M, Keller JM, Mitchell JA, Fuzzy measures on the gene ontology for gene product similarity, IEEE/ACM Transactions on Computational Biology and Bioinformatics, **3**(3), 2006, pp. 263-274.

169. Puzicha J, Hofmann T, Buhmann JM, A theory of proximity based clustering: structure detection by optimization, Pattern Recognition, **33**(4), 2000, pp. 617-634.

170. Rabiner LR, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE, **77**(2), pp.257-286, 1989.

171. Rattigan MJ, Maier M, Jensen D, Graph clustering with network structure indices, Proc. Int. Conf. on Machine Learning, 2007, pp. 783-790.

172. Rho S, Han BJ, Hwang E, Kim M, MUSEMBLE: a novel music retrieval system with automatic voice query transcription and reformulation, The Journal of Systems and Software, **81**, 2008, pp. 1065-1080.

173. Riccardi G, Hakkani-Tur D, Active learning: theory and applications to automatic speech recognition, IEEE Trans on Speech and Audio Processing, vol. 13(4), 2005, pp. 504-511.

174. Riesen K, Bunke H, Approximate graph edit distance computation by means of bipartite graph matching, Image and Vision Computing, vol. 27, 2009, pp. 950-959.

175. Robles-Kelly A, Hancock ER, Graph edit distance from spectral seriation, IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 27(3), 2005, pp. 365-378.

176. Rodrigues P, Gama J, Pedroso JP, Hierarchical time-series clustering for data streams, First Int. Workshop on Knowledge Discovery in Data Streams, Pisa, Italy, 2004.

177. Roddick JF, Spiliopoulou M, A survey of temporal knowledge discovery paradigms and methods, IEEE Transactions on Knowledge and Data Engineering, **14**(4), 2002, pp.750-767.

178. Scannell JW, Blakemore C, Young MP, Analysis of connectivity in the cat cerebral cortex, The Journal of Neuroscience, vol. 15(2), 1995, pp. 1463-1483.

179. Schölkopf B, Smola AJ, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press: Cambridge, Massachusetts, 2002.

180. Scholkopf B, Smola A, Muller KR, Nonlinear component analysis as a kernel eigenvalue problem, Neural Computation, **10**, 1998, pp.1299-1319.

181. Settles B, Active learning literature survey, Computer sciences technical report 1648, University of Wsconsin-Madison, 2009.

182. Shannon CE, A mathematical theory of communication, Bell System Technical Journal, **27**, 1948, pp. 379-423 & pp. 623-656.

183. Shawe-Taylor J, Cristianini N, Kernel Methods for Pattern Analysis, University Press: Cambridge, UK, 2006.

184. Shen H, Yang J, Wang S, Liu X, Attribute weighted mercer kernel based fuzzy clustering algorithm for general non-spherical datasets, Soft Computing, **10**(11), 2006, pp.1061-1073.

185. Shenoi S, Melton A, Proximity relations in the fuzzy relational database model, Fuzzy Sets and Systems, vol. 31, 1989, pp. 285-296.

186. Singhal A, Seborh DE, Clustering multivariate time-series data, Journal of Chemometrics, **19**, 2005, pp.427-438.

187. Smolders A, Martino FD, Staeren N, Scheunders P, Sijbers J, Goebel R, Formisano E, Dissecting cognitive stages with time-resolved fMRI data: a comparison of fuzzy clustering and independent component analysis, Magnetic Resonance Imaging, 2007, doi:10.1016/j.mri.2007.02.018

188. Staiano A, Tagliaferri R, Pedrycz W, Improving RBF networks performance in regression tasks by means of a supervised fuzzy clustering, Neurocomputing, **69**, pp. 1570-1581, 2006.

189. Stitson MO, Gammerman A, Vapnik V, Vovk V, Watkins C, Weston J, Support vector regression with ANOVA decomposition kernels, University of London, Department of Computing Science, Technical Report CSD_RE-97-22, 1997.

190. Storn R, Price K, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization, **11**, 1997, pp. 341-359.

191. Strehl A, Ghosh J, Cluster ensembles - a knowledge reuse framework for combining multiple partitions, Journal of Machine Learning Research, vol. 3, 2002, pp. 583-617.

192. Sun Spot Archives http://www.spaceweather.com/java/archive.html (Accessed March 19, 2007)

193. Tang W, Lu Z, Dhillon IS, Clustering with multiple graphs, Proc. IEEE Int. Conf. Data Mining, 2009, pp. 1016-1021.

194. Telsca L, Identifying time-clustering structures in the sequence of solar flare hard X-ray bursts, Physica A, 2007, doi:10:1016/j.physa.2007.05.041

195. Timm H, Klawonn F, Kruse R, An extension of partially supervised fuzzy cluster analysis, Proc. Fuzzy Information Processing Society, 2002, pp. 63-68.

196. Togneria R, deSilva CJS, Fundamentals of Information Theory and Coding Design, Chapman & Hall / CRC Press, 2002.

197. Tomanek K, Hahn U, Semi-supervised active learning for sequence labeling, Proc. Association for Computational Linguistics (ACL), 2009, pp. 1039-1047.

198. Truong TK, Lin CC, Chen SH, Segmentation of specific speech signals from multi-dialog environment using SVM and wavelet, Pattern Recognition Letters, **28**, 2007, pp. 1307-1313.

199. Tsumoto S, Hirano S, A comparative study of clustering methods for long time-series medical databases, Lecture Notes in Computing Science, **3131**, 2004, pp.260-272.

200. Tumer K, Agogino AK, Ensemble clustering with voting active clusters, Pattern Recognition Letters, vol. 29, 2008, pp. 1947-1953.

201. Tur G, Hakkani-Tur D, Schapire, RE, Combining active and semi-supervised learning for spoken language understanding, Speech Communication, vol. 45, 2005, pp. 171-186.

202. Tushir M, Srivastava S, A new kernelized hybrid c-mean clustering model with optimized parameters, Applied Soft Computing, **10**(2), 2010, pp.381-389.

203. Wang W, Zhang Y. On fuzzy clustering validity indices. Fuzzy Sets and Systems, vol. 158, 2007, pp. 2095-2117.

204. Wang X, Smith K, Hyndman R, Characteristic-based clustering for time series data, Data Mining and Knowledge Discovery, **13**(3), 2006, pp.335-364.

205. Weeks ER, Chaotic Time Series Analysis http://www.physics.emory.edu/~weeks/research/tseries1.html (Accessed March 19, 2007)

206. Weeks JR, The Shape of Space (Pure and Applied Mathematics), Taylor & Francis e-Library / CRC Press, 2005, pp. 137-147.

207. Weston N, Chitchyan R, Rashid A, A framework for constructing semantically composable feature models from natural language requirements, Proc. ACM Int. Conf. Software Product Line, vol. 446, 2009, pp. 211-220.

208. Williams P, Li S, Feng J, Wu S, Scaling the kernel function to improve performance of the support vector machine, 2nd Int. Symposium on Neural Networks 2005, **3496**, 2005, pp.831-836.

209. Wilson RC, Hancock ER, Luo B, Pattern vectors from algebraic graph theory, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27 (7), 2005, pp. 1112-1124.

210. Wu CH, Hsieh CH, Multiple change-point audio segmentation and classification using an MDL-based Gaussian model, IEEE Transactions on Speech and Audio Processing, **4**(2), 2005, pp. 647-657.

211. Xiao B, Torsello A, Hancock ER, Isotree: tree clustering via metric embedding, Neurocomputing, vol. 71, 2008, pp. 2029-2036.

212. Xing EP, Ng A, Jordan MI, Russel S, Distance metric learning with application to clustering with side-information, Advances in Neural Information Processing Systems, **15**, 2003.

213. Xion H, Swamy MNS, Ahmad MO, Optimizing the kernel in the empirical feature space, IEEE Trans on Neural Networks, vol. 16(2), 2005, pp. 460-474.

214. Xiong Y, Yeung DY, Time series clustering with ARMA mixtures, Pattern Recognition, **37**, 2004, pp.1675-1689.

215. Xiong Z, Radhakrishnan R, Divakaran A, Huang TS, Comparing MFCC and MPEG-7 audio features for feature extraction, maximum likelihood HMM and entropic prior HMM for sports audio classification, Proc. of Int. Conf. on Multimedia and Expo, **3**, 2003, pp. 397-400.

216. Xu H, Wang X, Liao Y, Zheng C, An uncertainty sampling-based active learning approach for support vector machines, Proc. IEEE Int. Conf. on Artificial Intelligence and Computational Intelligence, 2009, pp. 208-213.

217. Yan B, Domeniconi C, An adaptive kernel method for semi-supervised clustering, Proc 17th European Conf. on Meachine Learning, 2006.

218. Yan B, Domeniconi C, Exploration of different constraints and query methods with kernel-based semi-supervised clustering, Proc. IEEE Int. Conf. Systems, Man, and Cybernetics, 2006, pp. 829-834.

219. Yang ZR, Probabilistic mercer kernel clusters, Proc. of the 2005 IEEE Int. Conf. on Neural Networks and Brain, **3**, 2005, pp.1885-1890.

220. Yin X, Chen S, Hu E, Zhang D, Semi-supervised clustering with metric learning: an adaptive kernel-method, Pattern Recognition, **43**(4), 2010, pp. 1320-1333.

221. Zeyu L, Shiwei T, Jing X, Jun J, Modified FCM clustering based on kernel mapping, Proc. of Int. Society for Optical Engineering, **4554**, 2001, pp.241-245.

222. Zhang DQ, Chen SC, Clustering incomplete data using kernel-based fuzzy c-means Algorithm, Neural Processing Letters, **18**(3), 2003, pp.155-162.

223. Zhang D, Chen S, Fuzzy clustering using kernel method, Proc. Int. Conf. Control and Automation, 2002, pp.123-127.

224. Zhang L, Zhou C, Ma M, Liu X, Li C, Sun C, Liu M, Fuzzy kernel clustering based on particle swarm optimization, Proc. of the 2006 IEEE Int. Conf. on Granular Computing, 2006, pp.428-430.

225. Zhang H, Ho TB, Lin MS, An evolutionary k-means algorithm for clustering time series data, Proc. of the 3rd Int. Conf. on Machine Learning Cynernetics, Shanghai, 2004, pp.1282-1287.

226. Zhang H, Lu J, Semi-supervised fuzzy clustering: a kernel-based approach, Knowledge-Based Systems, **22**, 2009, pp. 477-481.

227. Zhou S, Gan J, Mercer kernel fuzzy c-means algorithm and prototypes of clusters, Proc. Conf. on Int. Data Engineering and Automated Learning, 2004, pp.613-618.

228. Zou Z, Li J, Gao H, S. Zhang, Mining frequent subgraph patterns from uncertain graph data, IEEE Trans. On Knowledge and Data Engineering, vol. 22(9), 2010, pp. 1203-1218.

APPENDICES

APPENDIX A: INCORPORATING SEMANTIC PROXIMITY OF NODES IN THE PROXIMITY FUNCTION

When the nodes are not perfect matches but exhibit proximity, a combined semantic and structural proximity measure can be derived. We define two nodes to be similar when the concepts they represent are similar. We assume the function $Match(f_1,f_2)$ exists, which gives the semantic proximity of the concepts of nodes f1 and f2 expressed on the domain [0..1]. The realization of the function depends on the elements that are used to determine the semantic similarity of the nodes. While this goes beyond the scope of this thesis, in our approach we have used Latent Semantic Analysis (LSA) [1022] for this purpose, a natural language processing technique that can determine the similarity of descriptions in natural language that are attached to individual nodes in the variability models.

A.1 MATCHING OF NODES

We define $H_{S\text{-}T}$ to be aet of tuples (n_a,n_b) that represent the matchings of nodes n_a in criterion S to nodes n_b in variability model T. Moreover, we define the quality L of match $H_{S\text{-}T}$ to be:

$$L\left(H_{S-T}\right)=\frac{\sum_{(n_a,n_b)\in H_{S-T}} Match\left(n_a,n_b\right)}{\left|S\right|} \qquad \textbf{(0-1)}$$

We define the match $H^o_{S\text{-}T}$ to be the optimal match for S and T, which means there is no match for S and T for which the quality is larger than $L(H^o_{S\text{-}T})$.

A.2 ACCOMMODATING SEMANTIC PROXIMITY IN THE PROXIMITY FUNCTION

To accommodate the semantic proximity in the function when evaluating T and W with respect to criterion S, we first define the function Ω:

$$\begin{array}{ll} \Omega\left(n_1,n_2\right)=1 & if\ \exists x\in S\ such\ that\left(x,n_1\right)\in H^0_{S-T}\wedge\left(x,n_2\right)\in H^0_{S-W} \\ \Omega\left(n_1,n_2\right)=Match\left(n_1,n_2\right) & if\ \forall x\in S\ such\ that\left(x,n_1\right)\notin H^0_{S-T}\wedge\left(x,n_2\right)\notin H^0_{S-W}\ . \\ \Omega\left(n_1,n_2\right)=0 & \text{otherwise} \end{array} \qquad \textbf{(0-2)}$$

The semantic proximity of nodes is included be redefining the fuzzy set $P(x)$ for $x\in S$ to be the following:

$$P\left(x\right)=1-\Omega\left(pt_T\left(x\right),pt_W\left(x\right)\right) \qquad \textbf{(0-3)}$$

In this definition the function pt_T returns the parent node of its argument in variability model T. The membership of elements in this set corresponds to how semantically similar the respective parents are in T and W. With this new definition, the combined semantic/structural distance function is defined as:

$$d_S\left(T,W\right)=w_E\left|E\right|+w_P\sum_{x\in S}P\left(x\right)+w_R\left|R\right|+w_C\sum_{x\in S}C\left(x\right) \qquad \textbf{(0-4)}$$

APPENDIX B: VARIABILITY MODELS

The variability models for data set 1 and 2 are described in the following tables.

Table 45. Variability Models of Data Set 1

Name	Description
M1	Model derived from the standard text using the Arborcraft tool with standard settings
M2	Model derived from the standard text using the Arborcraft tool with modified settings
M3	Model derived from the standard text using the Arborcraft tool with modified settings
M4	Model derived from the standard text using the Arborcraft tool with modified settings
M5	Model derived from the standard text using the Arborcraft tool with modified settings
M6	Model derived from the standard text using the Arborcraft tool with modified settings
M7	Model derived from the standard text using the Arborcraft tool with modified settings
M8	Model derived from the standard text by an expert software designer
M9	Model derived from the standard text by an expert software designer
M10	Model derived from the standard text by an expert software designer
M11	Model derived from the specialized text using the Arborcraft tool with standard settings
M12	Model derived from the specialized text by an expert software designer
M13	Model derived from the specialized text by an expert software designer
M14	Model derived from the specialized text by an expert software designer
M15	Model derived from the specialized text by an expert software designer

Table 46. Variability Models of Data Set 2

Name	Description
M16	Model derived from the standard text using the Arborcraft tool with modified settings
M17	Model derived from the standard text using the Arborcraft tool with modified settings
M18	Model derived from the standard text by an expert software designer
M19	Model derived from the standard text by an expert software designer
M20	Model derived from the standard text using a modified version of the Arborcraft
M21	Model derived from the standard text using a modified version of the Arborcraft with modified parameters
M22	Model derived from the standard text using a modified version of the Arborcraft with modified parameters
M23	Model derived from the standard text using a modified version of the Arborcraft with modified parameters
M24	Model derived from the standard text using a modified version of the Arborcraft with modified parameters
M25	Model derived from the standard text using a modified version of the Arborcraft with modified parameters
M26	Model derived from the standard text by an expert software designer
M27	Model derived from the specialized text using the Arborcraft tool with standard settings
M28	Model derived from the specialized text by an expert software designer
M29	Model derived from the specialized text by an expert software designer

APPENDIX C: MUSICAL SEGMENTATION LABELS

The "ground truth" labels used in evaluating the musical segmentation results are provided in this section for reference. The time for each song starts at 0s. The time given in each entry in the tables is the time at the end of the segment, i.e. boundary or the point in time when the segment transitions into the next segment. The last entry displays the length of the song in seconds. The number of segments (denoted by "*S*") is given in the table caption. The number of boundaries in each song is *B*=*S*-1. The hope size, i.e. number of seconds per sample/beat, is denoted by hopSize. The labels assigned are shown in the following tables.

Table 47. Structure of the song "Yellow" (*S*=10, hopSize=0.690s)

	Intro	Chorus	Verse	Pre	Chorus	Verse	Pre	Chorus	Chorus2	Outro
Time (s)	11.16	33.61	88.76	110.80	133.39	171.71	193.75	215.74	249.26	265.59

Table 48. Structure of the song "A Message" (*S*=8, hopSize=0.627s)

	Verse	Verse2	Chorus	Verse	Chorus	Bridge	Chrous	Outro
Time (s)	32.91	73.77	106.73	147.88	188.79	206.85	246.85	282.29

Table 49. Structure of the song "Fix You" (*S*=10, hopSize=0.572s)

	Intro	Verse	Verse2	Chorus	Instru	Verse	Chorus	Bridge	Bridge2	Outro
Time (s)	13.63	41.39	70.35	91.69	105.12	133.73	154.88	182.19	264.51	192.92

Table 50. Structure of the song "Swallowed in the Sea" (*S*=8, hopSize=0.423s)

	Verse	Chorus	Verse	Chorus	Bridge	Chorus	Chorus2	Outro
Time (s)	38.37	74.95	102.11	132.72	159.63	186.63	217.08	232.71

Table 51. Structure of the song "Talk" (*S*=9, hopSize=0.560s)

	Intro	Verse	Chorus	Verse	Chorus	Instru	Instru2	Chorus	Outro
Time (s)	36.17	79.20	119.76	163.43	207.59	239.31	255.54	291.61	305.09

Table 52. Structure of the song "A Rush of Blood to the Head" (*S*=11, hopSize=0.566)

	Verse	Instru	Verse	Chorus	Instru Verse	Verse	Verse2	Chorus	Instru Verse	Verse	Outro
Time (s)	58.47	73.48	104.38	146.96	173.83	201.72	233.33	275.52	301.04	328.30	347.03

Table 53. Structure of the song "In My Place" (*S*=10, hopSize=0.622s)

	Intro	Instru	Verse	Chorus	Instru	Verse	Chorus	Instru	Instru2	Outro
Time (s)	6.85	34.13	59.82	86.35	100.12	125.98	152.60	179.29	206.04	225.23

Table 54. Structure of the song "Politik" (*S*=9, hopSize=0.708s)

	Intro	Chorus	Verse	Chorus	Verse	Chorus	Instru	Instru2	Outro
Time (s)	3.15	25.64	70.53	104.43	147.71	171.95	244.65	289.91	317.05

Table 55. Structure of the song "God Put a Smile Upon Your Face" (*S*=11, hopSize=0.716s)

	Intro	Verse	Instru	Verse	Chorus	Instru	Verse	Chorus	Bridge	Chorus	Outro
Time (s)	16.61	45.92	62.03	91.34	122.01	138.12	167.26	198.34	227.85	259.44	289.18

Table 56. Structure of the song "The Scientist" (*S*=9, hopSize=0.822s)

	Intro	Verse	Chorus	Instru	Verse	Chorus	Instru	Instru2	Outro
Time (s)	26.27	79.52	113.12	137.61	190.28	223.60	249.16	288.66	307.10

*Note: Instru = “Instrumental”