FUZZY CLUSTERING IN DOCUMENT CLASSIFICATION
(2002)

by

Frank Wißbrock

Knowledge based Systems Group

Faculty of Computer Science and Mathematics

University of Paderborn

Submitted in partial fulfillment of the requirements
for the degree of

Diplom Wirtschaftsinformatiker

Correctors: Prof. Dr. Hans Kleine Büning, Knowledge based Systems Group
Dr.-habil. Benno Stein, Graduate School of Dynamic Intelligent Systems

Paderborn, 30.07.2002

ABSTRACT

Information retrieval is concerned with the structure, analysis, organization, storage, searching, and dissemination of information. Because of the increasing amount of data in the Internet and in Intranets the retrieval of relevant information is becoming more important and also more difficult. Because most of the data is stored in text documents, new methods to structure text documents are needed to cope with the problem. One approach is to classify the documents. This imposes a structure on the documents so that the retrieval performance may be improved. A sub-problem of classification is how to create the classification schema for the documents. Clustering methods proved to be useful to fulfill this task. Especially fuzzy clustering methods are attractive because they allow to create non-exclusive clusters and degrees of memberships for the document to the clusters. This leads to a more advanced document structure. This work presents the theoretical background of document clustering in general and especially document fuzzy-clustering. Therefore the k-means and k-medoid algorithms and their fuzzy versions are discussed. Experiments with the algorithms showed that the quality of the partitions created by the fuzzy-clustering algorithms are not superior to the quality of the partitions created by the crisp clustering algorithms. However the work provides a theoretical background that the fuzzy clustering algorithms have the potential to outperform the crisp clustering algorithms if one desires multiple class assignments of documents and degrees of membership and therefore provides a justification for further research.

TABLE OF CONTENTS

FIGURES AND TABLES

INDEX OF SYMBOLS

$\mathbf{A}$	Matrix A
$\mathbf{A}^{\mathbf{T}}$	Transpose of matrix A
C_j	Class j
$\cos(\mathbf{o_1}, \mathbf{o_2})$	Cosine value between object vector $\mathbf{o_1}$ and object vector $\mathbf{o_2}$
D^L	Collection of class labeled documents
$d(\mathbf{o_1}, \mathbf{o_2})$	Dissimilarity between object vector $\mathbf{o_1}$ and object vector $\mathbf{o_2}$
$\text{docfreq}(t_j)$	Document frequency of term j in a document collection
E	Set of edges
$e(m_j)$	Representation error of medoid m_j
$e_{Fuzzy}(m_j)$	Representation error of medoid m_j under fuzzy conditions
e(P)	Representation error of partition P with optimal medoids
$e_{Fuzzy}(P)$	Representation error of fuzzy-partition P with optimal medoids
$e(P,M_P)$	Representation error of partition P given the medoids M_P
F(i, j)	F-measure value for cluster i compared to class j
G=(V,E)	Graph with vertices set V and edges set E
I(S)	Index term vocabulary for S
$\text{idf}(t_j)$	Inverse document frequency of term j in a document collection
IR	Information Retrieval
m	fuzzifier
m_j	Medoid of cluster C_j
n_j	Number of documents in Cluster C_j
η_j	Clustering parameter for cluster C_j
$\mathbf{o_i}$	Feature vector of object i
$p(C_j)$	a priori probability of class C_j
$p(\mathbf{o} \mid C_j)$	Class conditional probability of object o
$p(C_j \mid \mathbf{o})$	a posteriori probability of class C_j, given that $\mathbf{o}$ is observed
$\wp(V)$	Set of all partitions of V
$\wp_k(V)$	Set of all k-partitions of V

$\mathbb{R}^p$	p dimensional space of real values
S	Set of objects
$s_{Jaccard}(\mathbf{o_1}, \mathbf{o_2})$	Jaccard coefficient between object vector $\mathbf{o_1}$ and object vector $\mathbf{o_2}$
$s(\mathbf{o_1}, \mathbf{o_2})$	Similarity between object vector $\mathbf{o_1}$ and object vector $\mathbf{o_2}$
SVD	Singular value decomposition
t_i	Index term i
$tf(t_j)$	Term frequency of term j in a document collection
$tf_{o_i}(t_j)$	Term frequency of term j in document o_1
$u_{j,i}$	Membership value of object i in cluster j
$u_j(x)$	Membership value of object x in cluster j
V	set of vertices
$\mathbf{v_i}$	feature vector of the object that is represented by the vertex v_i
$\overline{\mathbf{v}}_{C_j}$	Mean vector of cluster C_j
v_i	vertex i
$w_{i,j}$	Weight component j of object vector $\mathbf{o_i}$

ACKNOWLEDGEMENTS

The author thanks the chair of knowldege based systems at the university of Paderborn, which supported this work and made available a hardware infrastructure to realize the programs. The author also thanks the research associates Dr. Benno Stein and Dipl. Inf. Sven Meyer zu Eissen for helpful discussions about theoretic issues and their help by the implementation of parts of the programs.

Chapter 1

INTRODUCTION

1.1 System definition

"*Information Retrieval (IR)* is concerned with the structure, analysis, organization, storage, searching, and dissemination of information. An *IR system* is designed to make a given stored collection of information items available to a user population" [Sal00]. Fig. 1.1 shows a standard model for an IR system. In the figure rectangles mark elements of the system or its environment and ellipses mark the input or output of an element. There are two interfaces between the IR system and the environment, one to the authors and one to the users. Authors create documents that are stored in a collection and form the information base of the IR system. Users have an information need that should be satisfied by the IR system. Therefore they type a query containing certain keywords into the system that, in their own perception, most likely represent their information need. The system handles the request by searching in the collection for relevant documents. A matching procedure decides whether a document is relevant to a query or not. After relevant documents are identified they are presented to the user in list form.

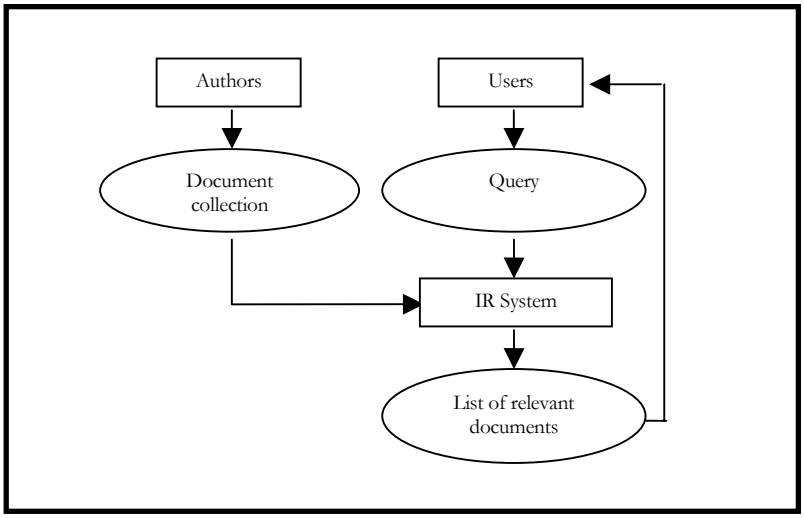

Figure 1.1: Standard Model of Information Retrieval

1.2 Problem description

Current IR systems still have potential for improvements because the generated results are only of moderate quality. Internet search engines are good examples for this. Additional over the last years the amount of digital stored text documents in sources like the Internet, digital libraries, news sources, or company-wide Intranets increased. Due to [FR95] this development will continue and make text data to one of the predominant data types stored online. This recent development challenges IR systems even more and demands for improvements because searching for information in document collections is increasingly difficult and time consuming.

One approach to cope with the increasing amount of information is to structure it. This increases the performance of the retrieval process and opens possibilities for other search processes than the classical search request. To classify documents within IR systems so that documents dealing with similar topics are grouped together is one possible structure. Document classification can be embedded at two positions into the standard IR model (Fig. 2). At position 1 all documents of the collection are classified. This has three advantages. First, the computation time for search requests is lower because only documents in classes that most likely match the search request must be analyzed in more detail. Second, users may navigate in the document collections and browse for information. Third, the classification represents the structure of the document collection and may be used for further processing like summarization or as knowledge base for reasoning systems. At position 2 all documents that are returned by a query get classified. Without a classification the documents are presented in list-form to the user. But despite all documents in that list contain the topic represented by the information request, usually only some of them cover the actually needed information. A classification fine-grains the results while identifying additional differences of the matching documents. This allows the user to browse the classes to find the needed information faster.

Creating a classification is very difficult because it bases on an understanding of the document contents in the context of the collection. In some cases, like in libraries, humans perform this task. However this has some shortcomings that reduce the quality of the resulting classification or, with an increasing number of documents, make it impossible to highlight any collection structure at all. First, an accurate classification has to represent the topics contained in the document collection. Therefore a person has to read all documents and understand their content before grouping them. This is time consuming and not possible for many practical applications. Additional the increasing number of documents in the collections require more than one person to perform this task. Therefore an increasing number of individuals is needed. This in turn leads to a cost explosion of the process. Next, some contents focus on special fields. Because expert knowledge is required to

understand them, this constrains the class of people that are able to understand the contents and may make the classification process impossible due to a lack of appropriate experts available. Last, the classification of the documents is subject to individual interpretation of the contents. Different persons tend to describe the same document differently and even the same person describes a document differently over time.[1] Hence different persons create different classifications and some of them may not be accurate to represent the structure.

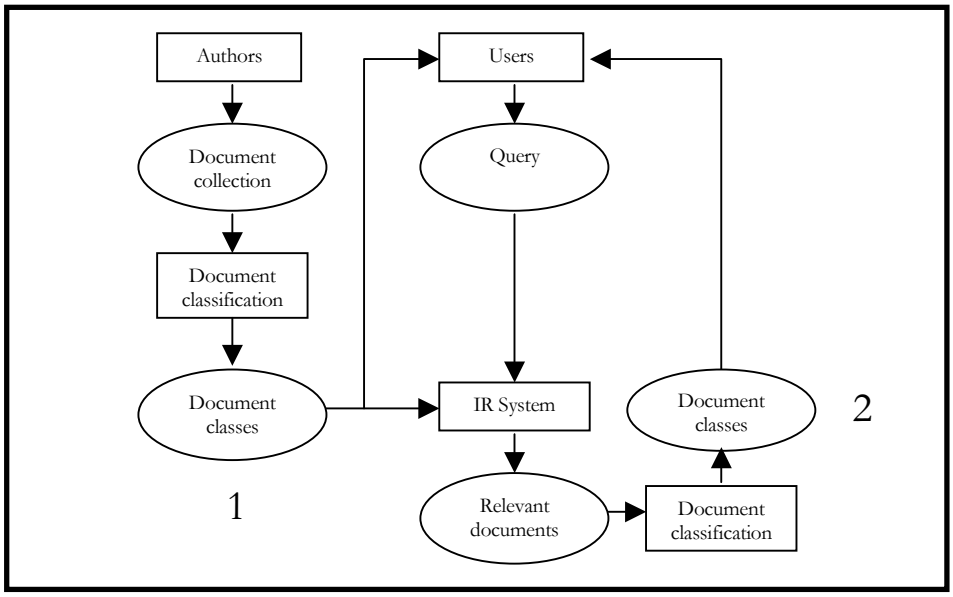

Figure 1.2: Extended Standard Model of Information Retrieval

In the document classification process human thinking processes are needed that are hard to automate. For most of the older document classification systems this was the main reason to be fully manual or semi-automatic. But due to the discussed shortcomings fully automatic document classification systems may have advantages and due to the increasing number of digital documents they may even be necessary in the future to perform the classification process with an accurate quality. Therefore the research work in developing models and methods for automatic document classification is justified.

1.3 Goal of this work

The goal of this work is to discuss fully automated document classification. A sub problem of classification is to discover a classification schema in the data. The task of discovering a classification schema in data is called clustering. Document clustering methods create classes so that documents in one class are more similar to each other than to documents in other classes. The theoretical foundation that document clustering may improve retrieval performance was formulated

[1] See [LP89, page 104]

in 1979.[2] This was followed by research about different clustering methods and their impact on the retrieval performance. Most of these methods create exclusive clusters of documents so that each document of the collection belongs to exactly one class. But the content of documents is usually very ambiguous so that the covered topics may be accurately described by more than one class. More recently fuzzy-clustering methods were applied to overcome this problem and to create more appropriate classifications. They have the advantage that each document of the collection may belong to more than one class. Additional these methods quantify the membership degree of each document to every class. Because this approach to document classification is relatively new, additional research is needed in this area. Therefore this work focuses on the evaluation of fuzzy-clustering methods in document classification.

1.4 Structure of this work

Chapter two gives a general introduction to document classification. It discusses the rationale behind document classification, introduces some class structures, and gives a brief overview about different approaches to classification. Chapter three formalizes the document clustering process. It introduces a formal model for document representation and formalizes the output of the clustering process. Chapter four discusses fuzzy-clustering methods. The basics of the algorithms are introduced and the algorithms are formally presented. Chapter five describes some experiments that were performed with the algorithms described in chapter four. Therefore it starts to describe evaluation methods for document clustering algorithms. This is followed by a description of the experimental details. The chapter closes with a conclusion of the observed results. Chapter six summarizes this work and gives an outlook for further research.

[2] See [Rij79]

Chapter 2

DOCUMENT CLASSIFICATION

The main problem with large collections of objects is that one can lose track of things due to an oversupply of data. Because every object has characteristics that make it unique, searching for objects with certain features is very difficult. However usually in every collection there are groups of objects that share certain characteristics and may be considered similar. Instead of representing these objects by multiple sets of features they can be represented by the group. This imposes a structure on the objects that bears the potential to increase efficiency and effectiveness of a search process or enables one to browse in the collection. The groups are called *classes*. Every class has a *class definition* that describes the contained objects. A set of classes that is intended to structure a certain collection of objects is called a *classification schema*.

The intention of this chapter is to explain document classification in the context of IR systems. Therefore the next section highlights the advantages of classified documents in IR systems. The second section describes the document classification process in more detail.

2.1 Advantages of classified documents in IR systems.

Using collections that are structured by a classification schema one hopes to increase the retrieval performance of an IR system. Therefore measurements that access the performance are needed. The next subsection discusses such measures. After that subsection 2.1.2 discusses different types of classification schemes and their potential to increase the performance of an IR system.

2.1.1 Performance measures for IR systems

The performance of an IR system is determined by the execution efficiency and the retrieval effectiveness of the system.

2.1.1.1 Execution efficiency

Definition (Execution efficiency)

The execution efficiency is measured by the time it takes the system, or part of the system, to perform a computation. [3] ■

[3] See [FB92, page10]

It is very important to have a high execution efficiency because most retrieval operations are interactive with the user. E.g. if a user enters a search query he expects the system to return the desired information within a couple of seconds.

2.1.1.2 Retrieval Effectiveness

The retrieval effectiveness of an IR system depends on the relevance of the returned information to the information need of the user. The disadvantage of relevance judgments is that they are subjective. Different judges will assign different relevance values to the same information.[4] However, the relevance judgment is commonly used in IR systems to access the effectiveness.[5] In [SM87] relevance measurement is discussed in more detail.

Definition (Recall and Precision)[6]

Let D be a document collection. Given a search request s of a user, let $n(s)$ be the number of documents found by s, let $n_f(s)$ be the number of relevant documents found by s, and let $n_r(s)$ be the number of all relevant documents in the collection. The *recall of s* is

$$R(s) = \frac{n_f(s)}{n_r(s)}.$$

The *precision of s* is

$$P(s) = \frac{n_f(s)}{n(s)}.$$

■

In other words, recall measures the ability of a system to find the relevant documents, while precision measures the ability to avoid non-relevant documents. It can be shown that within an IR system recall and precision are inversely related.[7] Therefore if the precision is increased, the recall is decreased and vice-versa. Fig. 2.1a shows an example of a recall-precision plot that visualizes this relationship.[8] Such a plot is created by evaluating the recall and precision values of several search requests.[9] The effectiveness of an IR system is increased if one can modify the system so that the recall-precision plot for the new system is somewhere on the upper right side of the old plot (Fig. 2.1b).

---

4 See [FB92, page 10]

5 See [FB92, page 10]

6 See [SM87, page175]

7 See [FB92, page 11]

8 The example plot is taken from [FB92, page 11]

9 See [SM87, chapter 5, subsection C]

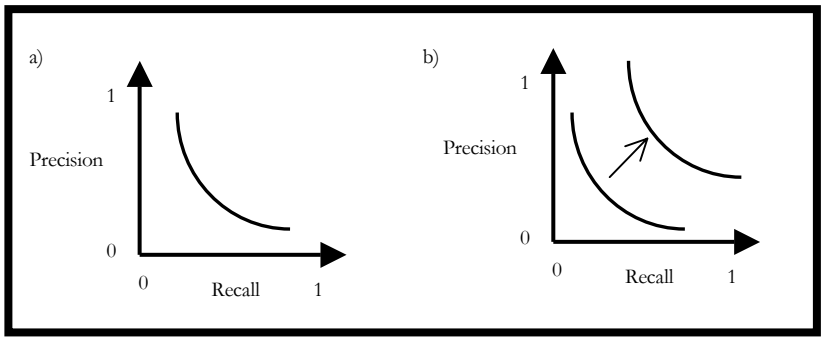

Figure 2.1: a) Example of a recall-precision plot. Precision and recall are inversely related. b) The recall-precision plot moves to the upper right. The effectiveness of the system is increased. A reason could be that parts of the system were changed.

Other properties of IR systems may also influence the performance. E.g. in history the storage efficiency was measured, too. However, this measure was important in times where document databases only were able to hold parts of every document, like the title, keywords, and sometimes an abstract. Today memory is not the bottleneck anymore and the two performance measures mentioned above are the important ones.

2.1.2 Types of classification schemes

In the following hierarchical and non-hierarchical classification schemes and their impact on the retrieval performance are reviewed.

2.1.2.1 Non-hierarchical classification schema

Fig. 2.2 visualizes a non-hierarchical classification schema. The classes are not related in any way. The class definition of every class can be matched to new documents. The document is assigned to the best matching class and is represented by this class in the future. This structure increases the execution efficiency of information retrieval systems. Given a search query the system only compares the query to the class definitions. The most appropriate class is returned completely or the search is continued within this class. In both cases computation time is saved because a huge part of the collection need not to be compared to the query. However it was shown that this class structure reduces the effectiveness of an IR system.[10]

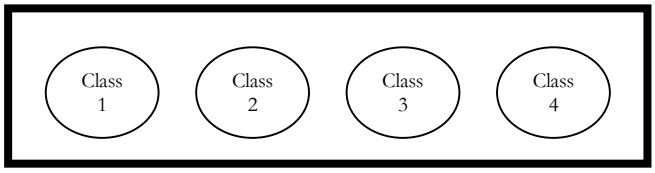

Figure 2.2: Nonhierarchical classification schema

[10] See [FB92, page 439]

2.1.2.2 Hierarchical classification schema

In a hierarchical classification schema the classes are arranged in a tree. Fig 2.3a shows an example of five documents. The tree has the height four. Every node represents a class. Every class is included in its parent. The root is the class that contains all documents of the collection and the leaves are classes consisting of a single document. Every layer of the tree represents a non-hierarchical class structure. E.g. the nodes of the bottom layer, the leaves, represent the non-hierarchical classification schema where every document forms its own class. The layer above the bottom layer represents the non-hierarchical classification schema with d_1 and d_2 forming one class and each of the remaining documents forming its own class. The difference between two layers is that the two most similar classes in the lower layer form one class in the upper layer. A tree with these properties is called a *dendrogram*. A class in a dendrogram has only two children that represent the two most dissimilar topics of the parent class.

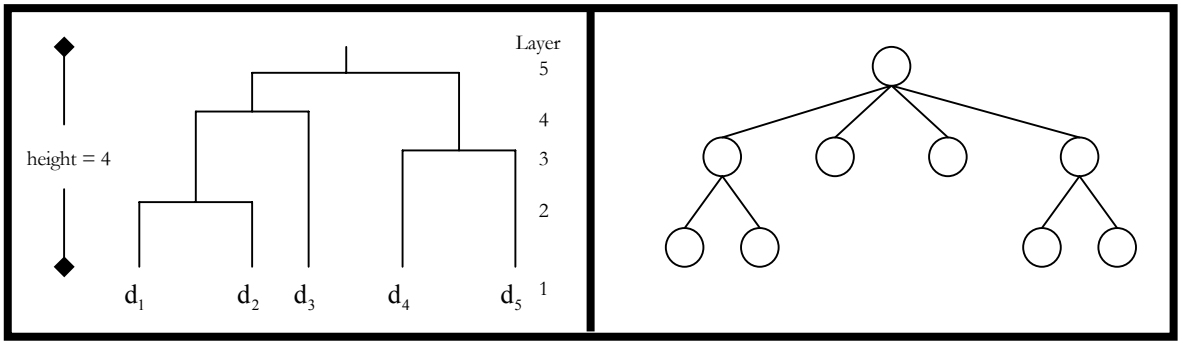

Figure 2.3: Hierarchical classification schemes: a) Dendrogram with height four. Every layer represents a non-hierarchical classification schema. Classes are included in their parents. b) A classification schema that supports browsing. Every class represents the main topic of the documents contained. Every child of a class represents a sub topic of the parent class.

[Rij79] stated that closely associated documents tend to be relevant to the same requests. This statement is known as the cluster hypothesis.[11] Based on the cluster hypothesis one hopes to increase the effectiveness of an IR system, if the documents are organized in a hierarchical classification schema.[12] The idea is that if one retrieves a relevant document from a classification schema other relevant documents are retrieved at the same time, thus leading to higher recall and precision measures. Given a search query the system may start at the root or at the leaves and move through the tree until a certain criterion is satisfied. The criterion could be the maximal or minimal number of documents in a class or a certain degree of similarity between search query and the class definition of a particular node. Such a search strategy leads to very high precision values but may

[11] See [Rij79]

[12] See [FB92, page 439] or [SM87, page 229]

decrease the recall values.[13] To overcome this problem the system should perform parallel searches in more than one path of the tree. Usually this increases the recall values.[14]

The classification schemes discussed until now are used to improve the classical search request. This means a user enters his or her information need and the system searches automatically in the classification schema for appropriate documents. Another type of searching is to browse in the collection. This means a user starts at one point in the classification schema and continues to fine-grain the results by moving to smaller, more specific classes. Fig. 2.3b shows a hierarchical structure that enables browsing in the collection. The difference to Fig.2.3a is that the children of a class represent sub topics of the parent class. E.g. a topic "Computer" may have the sub topics "Software", "Hardware", "Periphery". The browsing search method may increase the retrieval effectiveness of an IR system. E.g. a user starts with entering a search request. The system returns a class trying to reach a high recall. In the following the user browses from this point to other classes that more accurately represent his or her information need. While doing so the precision is increased.

2.2 The classification process

Definition (Classification)

Classification is the process of finding a model that describes and distinguishes data classes, for the purpose of being able to use the model to predict the class of objects with unknown class labels. A model for a particular application that satisfies this definition is referred to as *classifier*. ■

Fig. 2.1 shows the document classification process. The classification process requires a set of already class labeled documents that acts as training set. Based on the training set a classifier is derived which can be used to assign unlabeled documents to classes. Usually a test set of labeled documents exists that is different from the training set. The test set is used to evaluate the classifier and to adjust it if necessary. The so trained and evaluated classifier can be used to classify unlabeled documents. The classification process has two difficult parts. First one has to find a way to derive the classifier from the training set. This is discussed in the next subsection. Another difficulty may arise when there is no training set. In this case a classification schema with sample documents has to be created first. This problem is discussed in section 2.2.2.

[13] See [SM87, page 236]

[14] See [SM87, page 236]

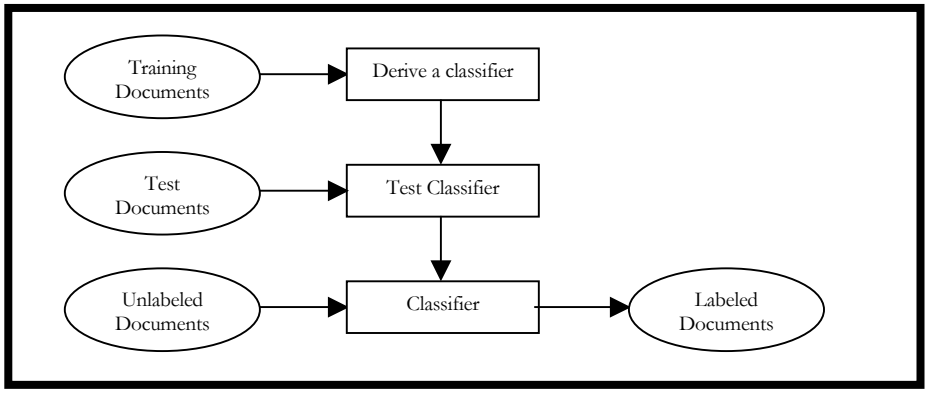

Figure 2.4: Document classification process

2.2.1 Approaches to derive a classifier

Usually it is assumed that every document is characterized by a couple of features. The features can be represented by a vector in the $\mathbb{R}^p$, which is called the feature space.[15] The models are similar in that they describe an approach to partition the feature space into disjoint regions. After this, new unlabeled documents can be classified by examine their features. Depending on in what region of the feature space the vectors take value, the documents are classified to the corresponding class.

The specific parameters of the classifier models are unknown at the beginning. They have to be learned from a given training set of class labeled documents D^L. D^L contains knowledge about the number of classes, the class definitions, and the typical content of the classes. In the following two approaches to derive the classifier are briefly discussed.

2.2.1.1 Bayes approach

This approach assumes that a feature vector $\mathbf{o} \in \mathbb{R}^p$ of a document is a random variable taking value in the feature space. A decision rule that assigns a document to a class C_j, $j=1,2,...,k$ is designed such that a given criteria is minimized or maximized.

In Bayes decision theory the conditional probability that $\mathbf{o}$ will occur, given that the corresponding document is a member of class C_j is $p = (\mathbf{o} \mid C_j)$. The probability that an arbitrary document belongs to class C_j is $p(C_j)$ and is called the a priori probability of C_j. The Bayes decision rule chooses the class C_a, $a \in \{1,2,...,k\}$ of an unlabeled document d so that

$$p(C_a \mid \mathbf{o}) = \max_{j \in \{1,2,...,k\}} \frac{p(\mathbf{o} \mid C_j) * p(C_j)}{\sum_{l=1}^{k} p(\mathbf{o} \mid C_l) * p(C_l)} \tag{2.1}$$

[15] For now the assumption is made that a document can be represented by a vector. In the next chapter the logic behind this representation is discussed

In other words C_a is the class with the maximum a posteriori probability, given that $\mathbf{o}$ is observed. Because the assignment to class C_a is done under uncertainty, there is always the possibility that the real class of $\mathbf{o}$ is not C_a. The probability of this mistake is called probability of error. It is the criteria for the Bayes classification model and can be calculated by

$$p^{error}(C_a \mid \mathbf{o}) = \sum_{\substack{j=1 \\ j \neq a}}^{k} p(C_j \mid \mathbf{o}).$$

It can be shown that the Bayes decision rule (2.1) minimizes the probability of error.[16] Because of this the Bayes decision rule (2.1) finds the best possible class C_a for a given document $\mathbf{o}$. Therefore it is said that the rule represents the limit of excellence beyond which it is impossible to go.[17] However in practical document classification the a priori probabilities $p(C_j)$ and the class conditional probabilities $p(\mathbf{o} \mid C_j)$ are not known.. They have to be learned from the training set.[18]

Usually the functional form of the class conditional probability density is also not known and nonparametric estimation methods have to be used. In this case statistics provide estimation methods for one dimensional random variables. But the feature vector of a document is n-dimensional. Therefore a simple estimation is only possible if the components of the feature vectors are independent. This assumption is usually done in Bayes document classification and the resulting classifier is called Naive Bayes. However the assumption does not hold in practice, but it can be shown that Naïve Bayes document classifiers still perform moderate. In [DP96] a theoretical explanation is given for this phenomenon.

2.2.1.2 Support vector machines

This approach assumes the feature vector $\mathbf{o}$ to be a point in the feature space. The goal is to spatial divide regions in the feature space. Fig 2.2 visualizes the idea of SVM in the two-dimensional space for a two class problem. The bold line divides the two classes and is usually called decision line. Points on one side of this line belong to class C_1 and points on the other side belong to class C_2. The normal lines show how much the decision line can be moved before a misclassification of the training data occurs. The distance between the two normal lines is called margin. Depending on the position of the line in the space the size of the margin is different. The idea of SVM is to fit the line into the feature space so that the margin is maximized.

---

16 See [The89, Chapter 3]

17 See [DK82, page 26]

18 See [NMTM98] for a procedure to estimate the parameters from the sample set.

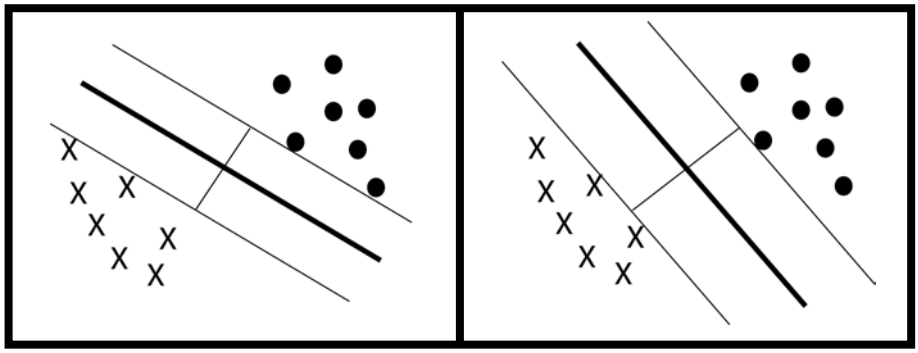

Figure 2.5: Visualization of the SVM idea

The SVM approach was applied to document classification and there is theoretical and empirical evidence that it is well suited for this.[19] SVM has also been examined in comparison to other document classifiers and showed a very good performance.[20]

2.2.2 Approaches to create a classification schema

The above document classification approaches require a training set and therefore also an a priori classification schema. But in some practical situations there is no given classification schema. E.g. dynamic document collections like internet search results make existing class definitions obsolete and new class definitions are needed with every new search request. Additional is it not possible to define classes a priori that cover every topic of large document databases or the Internet.

To create a classification schema a set of unlabeled documents is needed. If such a collection is available, clustering methods can be applied to create the classifications schema. The so derived classes and labeled documents can be used to train a classifier that classifies the remaining documents or new documents in the future.

2.3 Summary

This chapter discussed document classification. Using performance measures it highlighted the potential of a classification structure on document collections. It discussed two commonly used classifier approaches that enable an IR system to fit new documents into existing classification schemes. The chapter closed by mentioning the problem that a classification schema may not be available a priori. In this case clustering methods have to be applied to identify an accurate structure in the documents so that this structure can be used as classification schema. The following chapter will discuss document clustering in more detail.

---

[19] See [Joa98]

[20] See [YL99]

Chapter 3

DOCUMENT CLUSTERING

Definition (Clustering)

Clustering is the process of grouping data such that objects within a single group are similar, while objects of different groups are dissimilar according to some measure of similarity. The groups are called clusters. In the case of *document clustering* the data is a collection of documents. ■

The intention of this chapter is to formalize the document clustering process. The first section discusses some basic considerations, the second section presents the document model used in this work, and the third section formalizes the result of the clustering process.

3.1 Basic considerations

The specific process of clustering depends on the nature of the objects to be clustered. The notion of similarity between objects has to be adapted to a particular clustering problem. Fig. 3.1 visualizes this problem. A collection of objects with triangle and rectangle like shapes are given. Clearly, one would identify two classes in this collection, one class containing the triangles and the other class containing the rectangles. What seems to be an easy process is hard to explain formally. There is no obvious formal similarity measure that explains this manual clustering behavior. One could state that the number of sides differentiates the two clusters. But first it is not obvious why the number of sides is a more important attribute than, for example, the circumference and second the number of sides is not accurately defined for every triangle in the example. One of the triangles has an arched side, another one has four sides but seems to be more likely a triangle than a rectangle. There has to be a principle, not necessarily a mathematical one, that unites the members of the triangle class. A class based on such a principle is called a *concept*. Another example would be the concept of an apple. Two apples may differ in form, size, color, and even taste, but presented to a human observer he or she would be able to recognize them as apples despite the huge amount of possible variations. The discussion shows that it is very hard to define a certain concept. Therefore in research one uses a more restrictive definition to capture a concept formally:[21]

[21] See [Mic92]

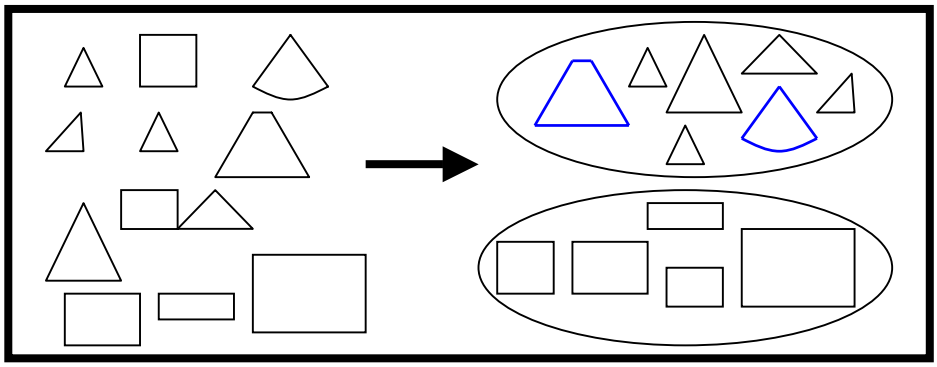

Figure 3.1: A collection of triangles and rectangles. A human observer would identify two groups: the objects with triangle like shapes and the objects with rectangle like shapes.

Definition (Concept)

A *concept* is a class of entities, which are united by some principle. A concept can be comprehensible described by no more than a small set of statements. The members of a concept are called *instances* and are equally representative.[22] ■

Definition (index term)

An *index term* represents a concept in a vocabulary. An index term in a natural language may be a single word, a number of words used alternatively, a phrase or phrases, or a text devoted to defining a concept. ■

Definition (Index term vocabulary)

Let S be a collection of objects. An index term vocabulary $I(S) = \{t_1, t_2, ..., t_p\}$, where $t_i, i = 1, 2, ..., p$ is an index term, is the set of all index terms that can be used to represent a concept in S. ■

3.2 Vector-space model

Usually a reader is interested in the content of a document. Therefore a similarity measure is needed that enables one to compare the contents of documents. The content of a document can be seen as a set of concepts (Fig.3.2a). Therefore the formal representation of a document should capture the concepts. One could state that the content itself best represents the concepts. But this representation is very complex and not formal. Therefore it is desirable to describe the concepts by a set of index terms.

[22] See [Mic92]

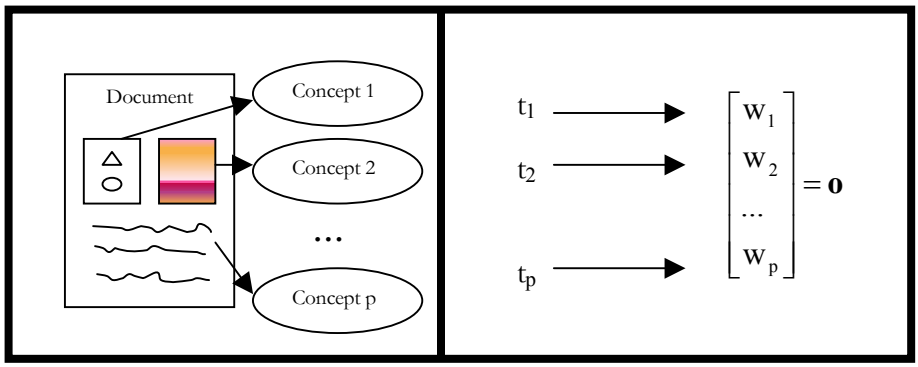

Figure 3.2: a) Document model: A document consists of a set of concepts. b) Vector-space model: The document model is transferred into the vector-space model.

The vector-space model is a document representation model that bases on index terms. Fig. 3.2b visualizes this model. Let $S = \{o_1, o_2, ..., o_n\}$ be a document collection. Given an index term vocabulary $I(S) = \{t_1, t_2, ..., t_p\}$, each document of the collection can be represented by a vector $\mathbf{o_i} = (w_{i,1}, w_{i,2}, ..., w_{i,p}), i = 1, 2, ..., n$. Each component of the vector represents an index term from the index term vocabulary and its value quantifies the importance of the index term for the particular document. If all components of the document vector are real values, then the document is represented by a point in the $\mathbb{R}^p$. This opens possibilities to impose a similarity or dissimilarity measure on the collection. In the following subsection these possibilities are discussed.

3.2.1 Similarity/Dissimilarity measures

Let S be a document collection. A *dissimilarity measure* $\mathrm{d}: \mathrm{S} \times S \to \mathbb{R}_{\geq 0}$ should have the following conditions:

1. $d(\mathbf{o_1}, \mathbf{o_2}) \geq 0$ for all $\mathbf{o_1}, \mathbf{o_2} \in S$
2. $d(\mathbf{o_1}, \mathbf{o_1}) = 0$ for all $\mathbf{o_1} \in S$
3. $d(\mathbf{o_1}, \mathbf{o_2}) = d(\mathbf{o_2}, \mathbf{o_1})$ for all $\mathbf{o_1}, \mathbf{o_2} \in S$

A dissimilarity measure that satisfies the above conditions is the Euclidean distance between two points in the n-dimensional space.

Definition (Euclidian distance between documents)

Let $\mathbf{o_1} = (w_{1,1}, w_{1,2}, ..., w_{1,p}), \mathbf{o_2} = (w_{2,1}, w_{2,2}, ..., w_{2,p})$ be two document vectors. The *Euclidian distance between* $\mathbf{o_1}$ *and* $\mathbf{o_2}$ is

$$d(\mathbf{o_1}, \mathbf{o_2}) = \sqrt{\sum_{j=1}^{p} (w_{1,j} - w_{2,j})^2}$$

■

The Euclidean distance has a shortcoming. Usually a document collection consists of more than only a few concepts. Therefore the index term vocabulary has to be large to capture all concepts. This results in a high dimensional document vector. But because a document only contains a small subset of the concepts discussed in the collection most components of the document vector are zero values. Vectors that have this property are called sparse vectors. The Euclidean distance does not work very well on sparse vectors in high dimensional spaces.[23] The reason is that the presence of a concept is as important as the absence of a concept in this measure. Fig. 3.3 visualizes this problem. The documents that are represented by the two vectors are completely different because they do not share any concepts. A similarity/dissimilarity measure should recognize this. But some measures, like the Euclidean distance, recognize a high similarity between the two vectors because most components are equal, namely zero. Such a measure assumes that the absence of a concept in two documents implies a similarity of the two documents to a certain degree. But this assumption is not satisfied. E.g. a document may solely contain mathematical concepts and another document may solely contain sports concepts. Both documents would score zero on economics concepts. But this similar property should not imply any similarity between the documents. Summarizing, for sparse vectors in high dimensional spaces the presence of a concept should be more important than the absence.[24] In the following a similarity measure is introduced that only takes into consideration components that are greater than zero for both vectors.

Let S be a set of objects. A *similarity measure* $s : S \times S \rightarrow [0,1]$ should have the following conditions:

1. $s(\mathbf{o}, \mathbf{o}) = 1$ for $\mathbf{o} \in S$
2. $s(\mathbf{o_1}, \mathbf{o_2}) = s(\mathbf{o_2}, \mathbf{o_1})$ for $\mathbf{o_1}, \mathbf{o_2} \in S$

The restriction on the interval [0,1] is necessary. Without this restriction equal objects cannot be compared by this measure. One measure that satisfies the above conditions is the cosine measure.

---

[23] See [SEK00]

[24] See [SEK00]

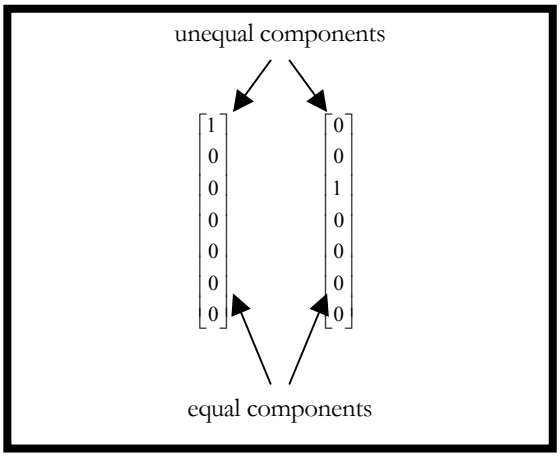

Figure 3.3: Sparse vectors are similar because they have a lot of equal components. For documents this property is not desired.

Definition (Cosine similarity between documents)

Let $\mathbf{o_1} = (w_{1,1}, w_{1,2}, ..., w_{1,p}), \mathbf{o_2} = (w_{2,1}, w_{2,2}, ..., w_{2,p})$ be two document vectors. The *cosine similarity between* $\mathbf{o_1}$ *and* $\mathbf{o_2}$ is

$$\cos(\mathbf{o_1}, \mathbf{o_2}) = \frac{\mathbf{o_1} \bullet \mathbf{o_2}}{\|\mathbf{o_1}\| * \|\mathbf{o_2}\|} = \frac{\sum_{j=1}^{p} (w_{1,j} * w_{2,j})}{\sqrt{\sum_{j=1}^{p} (w_{1,j})^2} * \sqrt{\sum_{j=1}^{p} (w_{2,j})^2}}$$

■

Fig. 3.4 visualizes the cosine measure for an example in the two dimensional space. The two documents are represented by the two vectors $\mathbf{o_1}$ and $\mathbf{o_2}$ in this space. The cosine measure returns the cosine value of the angle α between the two vectors. When the components of all vectors are greater or equal than zero, the cosine measure returns values in the interval [0,1]. If two documents are equal, then they have equal document vectors. In this case the angle between the vectors is zero and the cosine measure returns 1. In contrast the highest possible angle between two vectors is 90. In this case the measure returns 0.

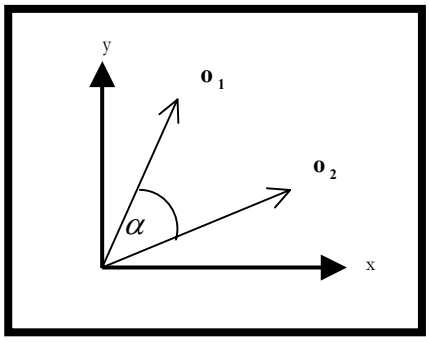

Figure 3.4: The cosine measure returns the cosine value of the angle α between the two vectors.

It is useful to normalize all document vectors to unit length $\mathbf{o_i} := \mathbf{o_i}/\|\mathbf{o_i}\|$. Now the cosine similarity can be calculated by $\cos(\mathbf{o_1}, \mathbf{o_2}) = \mathbf{o_1} \bullet \mathbf{o_2}$. This reduces the computation time in later processing steps. Another approach to access the similarity between two vectors is the Jaccard coefficient.

Definition (Jaccard coefficient)

Let $\mathbf{o_1}, \mathbf{o_2}$ be two document vectors. Let a be the number of components that are greater than zero for both vectors. Let b be the number of components that are greater than zero for only one of the two vectors. The Jaccard coefficient is

$$s_{\text{Jaccard}}(\mathbf{o_1}, \mathbf{o_2}) = \frac{a}{a+b}$$

■

The Jaccard coefficient is a binary similarity measure. This means it takes into account the occurrence of an index term in a document but not its weight. Two components are equal as soon as they both are greater than zero. Two components are unequal if one and only one of them is zero. Components that are zero for both vectors are not considered in the formula. Therefore the Jaccard coefficient does not have problems with sparse vectors.

The vector-space model is attractive because it opens possibilities to define a similarity measure on the documents. But until now the text implicitly assumed that it is possible to transfer a document into a vector in the $\mathbb{R}^p$. Therefore one has to answer the questions how to choose the index term vocabulary and how to assign the weights to the document vectors. The following subsection provides answers to these questions.

3.2.2 Document indexing

Definition (Indexing)

Indexing is the process of

1. the identification and selection of the concepts representing the document content and
2. assigning weighted index terms to the concepts that accurately represent them. ■

The concepts described by a document base on three sources. The first source are the data types forming the document. A document may contain several data types like text, figures, videos or sound. The combination of these data types and their content are intended to collectively capture certain concepts. The second source is the context of a document in the collection. A certain document discusses topics that are associated with certain concepts. But in the context with other documents the topics may be seen in another light, thus associating other concepts. E.g. some documents refer to or are continuations of other documents and require the reader to read the predecessor documents first. The third source is the experience of the reader. Documents written for a certain group of people, like engineers, can assume a certain degree of a priori knowledge. Without this knowledge a reader could be unable to identify any concepts at all.

The experience of the reader is hard to formalize because it requires one to make assumptions about him or her. This is not part of this work and will not be discussed further. Therefore the content of all documents in the collection is left for analysis. For nearly every data type there are methods that extract the represented concepts.[25] But in the following this work focuses on the text data type because text data is one of the most widely used data types and usually within a document text contains most of the important information.[26]

The structure of text consists of two parts, the words of the language that is used to write the text and the grammar that defines the meaning of combinations of words and phrases. Indexing methods that base on knowledge about the text structure of a language are called linguistic. Indexing methods that base on occurrences and distributions of words in documents are called statistic. Linguistic methods are closer to how humans deal with text. But linguistic methods have to cope with some problems. First the words used in common natural languages are not a closed set. In contrast, it is a very dynamic set. New words evolve and old ones disappear. Variations of words can be formed that are not written down in a wordbook but still are meaningful. Second the grammar intends to formally describe the combinations of words. However most common natural

[25] See [SZR99] or [C01] for picture indexing, [G98] for video motion indexing. See also [CSBB97].

[26] See [FR95]

languages do not have a closed set of rules so that exceptions are possible.[27] These problems make it difficult to formalize a natural language. In contrast statistic methods try to approximate the linguistic methods. They assume that the concepts are best represented by the distribution of the words in the text.[28]

3.2.2.1 Identification of concepts in document collections

Using the vector-space model one has to determine the index term vocabulary that captures all important concepts of the collection. A good point to start is to select all words of the collection S to be the index term vocabulary $I(S) = \{t_1, t_2, ..., t_p\}$. Cleary, an index term vocabulary chosen in this way has the shortcoming that it also contains index terms that refer to unimportant concepts or do not refer to any concepts at all. Therefore these index terms have to be eliminated from the index term vocabulary.

Looking at the properties of text, it is fact that words in natural languages are not equally distributed. [29] An author uses words in dependence on what concepts he intends to communicate. Therefore the frequency of words contains information about the concepts.

Definition (Term frequency)

Let $\{t_1, t_2, ..., t_p\}$ be the set of all words that occur in a document collection $S = \{o_1, o_2, ..., o_n\}$. The *term frequency of* $t_j, j = 1, 2, ..., p$ *in document* $o_i, i = 1, 2, ..., n$, $tf_{o_i}(t_j)$, is the amount of occurrence of term t_j in o_i. The *term frequency of* t_j *in the collection* is

$$tf(t_j) = \sum_{o \in S} tf_o(t_i)$$ ■

It can be shown that the terms with a high term frequency $tf_S(t_j)$ in the collection are the non-informative ones.[30] These are words like "the", "of", "and". They should not be used as index terms for any document. Therefore they are eliminated from the index term vocabulary either by using a stop-word list or by using a threshold. A stop-word list contains words that are definitely unimportant as index terms. It can be matched to the index term vocabulary to eliminate the unimportant words. A stop-word list has the disadvantages that it needs a lot of work to create it and that it is restricted to a certain language. A threshold is a value. Every term with a term frequency above this value is eliminated form the index term vocabulary. The disadvantage of a

---

[27] See [C86, chapter 2]

[28] See [Kar00]

[29] See [SM87, page 65]

[30] See [SM87, page 66]

threshold is the difficulty in choosing the threshold so that only the unimportant words are eliminated from the index term vocabulary. It can also be shown that the words with a low $tf_o(t_i)$ are unimportant.[31] Thus they are also eliminated from the index term vocabulary. Summarizing the words with a medium frequency are the important ones. Within this class of words the more frequent the words are the more likely they are in capturing an important concept from the collection.

3.2.2.2 Discriminating power of index terms

The term frequency is a measure for the importance of a concept for a particular document. But this measure has a shortcoming. It does not capture how good an index term is in discriminating documents in the collection. E.g. if a document collection is entirely about soccer topics, the word "soccer" is likely to be present in every document. Therefore it would be an important concept for every document. But because it characterizes every document it cannot be used to discriminate a certain group of documents from the rest of the collection. In contrast, concepts that are only part of some documents in the collection have a higher discriminating power. Concepts that discriminate documents are important for clustering tasks because the goal is to identify similarities and dissimilarities between documents. The following definition shows how one can measure the discriminating power of an index term.

Definition (Inverse document frequency)

Let $I(S) = \{t_1, t_2, ..., t_p\}$ be an index term vocabulary for the document collection $S = \{o_1, o_2, ..., o_n\}$. Let $docfreq(t_j)$ be the number of documents from S that contain the word t_j. The *inverse document frequency of* $t_j, j = 1, 2, ..., p$ *in S* is

$$idf(t_j) = \log_2 \frac{n}{docfreq(t_j)} + 1. \quad (3.1)$$

■

Recall that the index term vocabulary is chosen from the collection. Therefore $docfreq(t_j)$ cannot be zero. The higher the value of the inverse document frequency for a certain index term, the more important is that index term in discriminating documents in the collection. The inverse document frequency has the lowest value for index terms that occur in every document of the collection. In this case it equals one. With decreasing $docfreq(t_j)$ the $idf(t_j)$ increases. The logarithm function is used to smooth the measure.

[31] See [SM87, page 67]

3.2.2.3 Weight of index terms

Using the vector-space model a document is represented by a vector $\mathbf{o_i} = \{w_{i,1}, w_{i,2}, ..., w_{i,p}\}, i = 1, 2, ..., n$. After the index term vocabulary is chosen, the weights for every document vector have to be determined. Because the vectors are used to impose a similarity measure on them, the weights should include the importance of an index term as well as its discriminating power. Therefore a combination of term frequency and inverse document frequency is used to determine the weights:[32]

$$w_{i,j} = tf_{o_i}(t_j) * [\log_2 \frac{n}{docfreq(t_j)} + 1]. \quad (3.2)$$

If an index term refers to an important concept in the document, $tf_{o_i}(t_j)$ has a high value. Is the concept very common in the collection, the inverse term frequency has a value near one, thus the weight nearly equals $tf_{o_i}(t_j)$. However, is the concept very uncommon for the collection the $idf(t_j)$ causes the weight to increase. Additional one can see why the logarithm is used in (3.1) to smooth the measure. This reduces the effect of the collection size on the weights.

3.2.2.4 Dimensionality reduction

One property of natural languages is their ambiguousness. This means that different words may refer to the same concept. E.g. the words "boat" and "ship" are referring to the same concept. Therefore documents that talk about "boats" should be similar to documents that talk about "ships". This requires that both types of documents are assigned the index term that refers to the concept of a "boat"/"ship". But because the index term vocabulary is constructed from the words of the collection, it is likely that it does not contain a unique index term for this concept. Probably both words, "boat" and "ship", are in the index term vocabulary. This redundancy has to be eliminated so that every concept in the collection is represented by only one index term in the index term vocabulary. The reduction of redundant index terms is called conflation. In the following some conflation methods are briefly discussed.

Morphological conflation

Morphology is the science of word form changes. E.g. the word "document" has the plural form "documents". The word "to write" has the conjugation "he writes". Morphological conflation aims at identifying different forms of a word and reducing them to their stem. The stem is used as index term. E.g. the words "analysis", "analyzing", "analyzer", and "analyzed" have the stem "analy".[33]

---

[32] See [SM87, page 69]

[33] The example was taken from [SM87, page 78]

The part that distinguishes a word from its stem is called suffix. The suffix of "analysis" is "-sis". Algorithm that reduce words to their stem are called stemming algorithms. Usually this algorithms base on a list of possible suffixes. The list depends on the language the algorithm is applied to. Usually every common language has words that have an uncommon stem, the so called exceptions. Therefore the algorithm needs exception rules to cope with these words. A stemming algorithm for the English language is described by [Por80].

Semantic conflation

The „ship"/"boat" example from the beginning of this subsection is a good example for semantically related words. "Boat" is not a form of "ship". It is a different word but refers to the same concept. There are linguistic as well as statistic methods that identify semantically related words. A linguistic method bases on a list of words with associations to semantically similar words. Such a list is called thesaurus. Statistic methods base on distributions of words in the document collection. A statistic method, called latent semantic indexing (LSI) that copes with semantic conflation is described by [DDFL90]. In the following the method is briefly described. LSI requires that the document vectors are arranged in a terms by document matrix **A** . It can be shown that the matrix **A** can be decomposed into three other matrices **T**, **S**, and **D** (Fig 3.5). **T** is the matrix of the left singular vectors, **D** is the matrix of the right singular vectors and **S** is a diagonal matrix that contains the singular values. The singular values are arranged in descending order. The decomposition is called singular value decomposition (SVD).

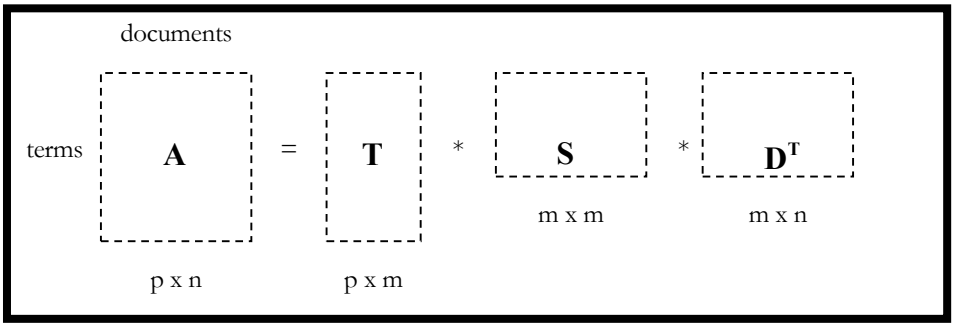

Figure 3.5: Visualization of the singular value decomposition

The goal of SVD is to map the term vectors and the document vectors into a space of uncorrelated factors. Therefore the new document vectors are represented by the matrix **D**. After the SVD is calculated there are two possibilities to proceed. First, the Matrix **D** with the new document vectors can be used as the new document representation. This step is related to factor analysis and results in document vectors that have a lower dimensionality than the original document vectors. The components of the new vectors are not the original index terms anymore. Moreover concepts that

are semantically related are represented by one component in the new document vectors. E.g. consider the "boat"/"ship" example. In the new document representation both index terms are represented by one component. The second possibility is to recalculate the original document vectors so that the weight of semantically related index terms for a particular document are all greater zero if one of the index terms is used in the document. Therefore the k greatest singular values are chosen to reduce **S** to $\mathbf{S_1}$ (Fig. 3.6). Additional the matrices $\mathbf{T}/\mathbf{D}^T$ are reduced to $\mathbf{T_1}/\mathbf{D_1^T}$ by choosing the first k columns/rows. The product of the matrices $\mathbf{T_1}$, $\mathbf{S_1}$, $\mathbf{D_1^T}$ create $\mathbf{A_1}$. The new matrix $\mathbf{A_1}$ contains the new document vectors. They have the same dimensionality than the original vectors. The difference is that some zero values of the original document vectors are greater than zero in the new document vectors. E.g. in the "boat"/"ship" example a document that only talks about boats does also have a weight greater than zero for the ship component in the vector. $\mathbf{A_1}$ is the more similar to A the closer k is chosen to m

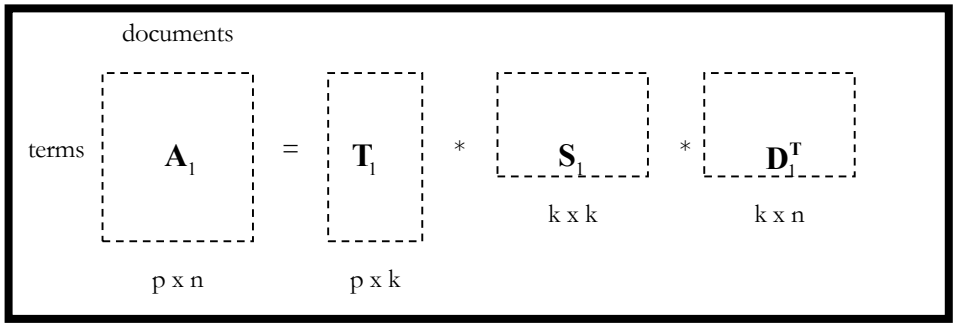

Figure 3.6: Visualization of the reduced singular value decomposition

Summarizing latent semantic indexing does not reduce the dimensionality of the document vectors like factor analysis, but it identifies semantically related index terms. The advantage of the vectors created by latent semantic indexing is that they can directly be matched to search queries. While there are authors that use different words to say the same things, there are also users that enter different search queries but have the same information need. Such different queries can be matched to the document vectors and will return useful results. In the case that the document vectors are reduced to certain factors, the queries need to be pre-processed and mapped to the same factor space to return useful results. For document clustering both approaches are useful. The reduced document vectors usually lead to an improvement of the runtime when the dissimilarity between two vectors is calculated.

3.2.3 Graph based representation

Let G=(V,E) be an undirected, edge weighted graph. $V = \{v_1, v_2, ..., v_n\}$ is a set of vertices and $E \subset V \times V$ is a set of edges. For every $(x, y) \in E$ let $w_{x,y}$ be the weight of the edge between vertex x and vertex y.

The vector-space model can be extended so that the documents and their relations are represented by a graph. In the graph model each document of the collection is represented by a vertex. An edge between two vertices represents the similarity/dissimilarity between the two corresponding documents. Therefore $w_{x,y} = d(x, y)$ for a dissimilarity graph and $w_{x,y} = s(x, y)$ for a similarity graph. Let $w_{x,y} = 0$ for $(x, y) \notin E$ in a similarity graph and let $w_{x,y} = \infty$ for $(x, y) \notin E$ in a dissimilarity graph.

For computation purposes a graph is usually represented as a vertex by vertex matrix (Fig. 3.7). The content of the matrix contains the weights of the edges.

$$\mathbf{D} = \begin{bmatrix} d(v_1, v_1) & d(v_1, v_2) & \dots & d(v_1, v_2) \\ d(v_2, v_1) & & & \\ \dots & & & \dots \\ d(v_n, v_1) & & & d(v_n, v_n) \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} s(v_1, v_1) & s(v_1, v_2) & \dots & s(v_1, v_2) \\ s(v_2, v_1) & & & \\ \dots & & & \dots \\ s(v_n, v_1) & & & s(v_n, v_n) \end{bmatrix}$$

Figure 3.7: Dissimilarity matrix **D** and similarity matrix **S**

3.2.4 Experimental results

The above discussed indexing methods were implemented in Java.[34] In the following the program and its results are discussed.

3.2.4.1 Document collection

The algorithm was adapted to run on the "Reuters-21578, Distribution 1.0".[35] The collection was made available for research purposes by Reuters Ltd. and Carnegie Group, Inc. in 1990 and represents the documents that appeared on the Reuters newswire in 1987. It consists of 22 text files (reut2-000.sgm to reut2-021.sgm) that together contain 21578 documents. The collection was

[34] Java is an object-oriented programming language. The language can be obtained at http://www.sun.com.

[35] The collection was taken from http://www.research.att.com/~lewis

adapted to serve as a test collection for several IR tasks. Therefore a subset of documents has class assignments that can be used for evaluation of clustering algorithms.

Fig. 3.8 shows an example of a common document from the collection. Tags are used to mark certain regions of the document. The beginning of a document is marked by "<REUTERS" while the end is marked by "</REUTERS>". The tags "<TOPICS>" and "</TOPICS>" enclose the names of the classes a document belongs to, "<TEXT>" and "</TEXT> marks the information content, "<TITLE>" and "</TITLE>" enclose the title , and "<BODY>" and "</BODY>" enclose the body of the text. Within the collection there are several other tags containing additional information. However in the program only the information enclosed by the above described tags are used.

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5552" NEWID="9">
<DATE>26-FEB-1987 15:17:11.20</DATE>
<TOPICS><D>earn</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;F
&#22;&#22;&#1;f0762&#31;reute
r f BC-CHAMPION-PRODUCTS-&lt;CH   02-26 0067</UNKNOWN>
<TEXT>&#2;
<TITLE>CHAMPION PRODUCTS &lt;CH> APPROVES STOCK SPLIT</TITLE>
<DATELINE>    ROCHESTER, N.Y., Feb 26 - </DATELINE><BODY>Champion Products Inc said its
board of directors approved a two-for-one stock split of its
common shares for shareholders of record as of April 1, 1987.
The company also said its board voted to recommend to
shareholders at the annual meeting April 23 an increase in the
authorized capital stock from five mln to 25 mln shares.
Reuter
&#3;</BODY></TEXT>
</REUTERS>
```

Figure 3.8: A common document from the Ruters collection.

From the set of all documents some text files with subsets of the original collection were created. Table 3.1 gives an overview about the files. The files Multicat*.txt contain documents that are assigned to at least one class. The number of all documents in the Reuters collection that satisfy this constraint is 11367. The files Singlecat*.txt contain documents that are assigned to exactly one class. In the Reuters collection are 8913 documents of this type.

Files	Number of documents	Number of categories
Multicat500.txt	500	73
Multicat1000.txt	1000	77
Multicat1500.txt	1500	87
Multicat2000.txt	2000	93
Multicat5000.txt	5000	105
Multicat11367.txt	11367	120
Singlecat500.txt	500	33
Singlecat1000.txt	1000	40
Singlecat1500.txt	1500	45
Singlecat2000.txt	2000	45
Singlecat5000.txt	5000	55
Singlecat8913.txt	8913	66

Table 3.1: Files that contain subsets of the Reuters collection

3.2.4.2 Indexing program[36]

The program starts by determining the term frequency of every word in the collection. Therefore the words are saved with their frequency in a hash-set. Before a term is entered into the hash-set it is reduced to its stem using Porter's stemming algorithm.[37] After the hash-set is created words that are similar to a given stop-word list are eliminated from the set.[38] Finally all terms that have a term frequency of one are eliminated from the set. The result is the index term vocabulary for the collection. The index term vocabulary is used to create the documents by terms matrix. The components of the matrix are calculated by the combination of term frequency and inverse document frequency (3.2). Optionally the weights can be adjusted by applying latent semantic indexing to the document vectors.[39] This increases the runtime significantly. E.g. for 500 documents the runtime of the program on a Pentium II 220Mhz is below 30 seconds without latent semantic indexing and around 15 minutes with latent semantic indexing. Table 3.2 shows some results of the algorithm. The number of index terms before reduction includes all different words that are reduced to their stem. The number of index terms after reduction includes all terms without the ones in the stop-word list and without the terms that have a term frequency of one. The reduction ratio shows how much the index term vocabulary was reduced by the algorithm. It is an interesting observation that the reduction ratio is relatively constant over all document sets. It would be of interest to see if this observation can be repeated with other document collections. But this is not part of this work and will not be discussed further.

---

[36] The user manual of the indexing program is in appendix D

[37] The source code of the stemming algorithm was taken from http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/

[38] The stop-word list was taken from http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/. It was adapted by the author. The stop-word list is shown in appendix E

[39] The Singular Value Decomposition procedure that is necessary for latent semantic indexing was taken from the National Institute of Standards and Technology at http://math.nist.gov/javanumerics/jama/.

File	Words	Index terms (before reduction)	Index terms (after reduction)	Reduction ratio [%]
Multicat500.txt	57,942	4,273	2,432	57
Multicat1000.txt	112,364	6,093	3,515	58
Multicat1500.txt	171,175	7,761	4,468	58
Multicat2000.txt	236,664	9,175	5,409	59
Multicat5000.txt	599,971	14,593	8,879	61
Multicat11367.txt	1,370,960	22,403	13,878	62
Singlecat500.txt	53,373	4,090	2,264	55
Singlecat1000.txt	109,656	6,245	3,465	55
Singlecat1500.txt	168,368	7,924	4,536	57
Singlecat2000.txt	223,785	9,182	5,298	58
Singlecat5000.txt	572,444	14,946	8,934	60
Singlecat8913.txt	1,012,400	20,203	12,364	61

Table 3.2: Results from the indexer program

Some of the indexed collections were used to create a dissimilarity matrix. Tables 3.3 and 3.4 show the distribution of the values in the dissimilarity matrix for the cosine measure. The cosine measure was transformed into a dissimilarity measure by applying the formula $1-\cos(\mathbf{o}_1,\mathbf{o}_2)$. In this case the dissimilarities take value in the interval [0,1], whereas 1 means that two documents are dissimilar and 0 means that they are similar. It is obvious that most documents are very dissimilar. In all cases around 85% of all dissimilarity values are in the interval [0.9, 1.0[.

	Singlecat500.txt		**Singlecat1000.txt**		**Singlecat1500.txt**		**Singlecat2000.txt**	
	Dissimilarity	[%]	Dissimilarity	[%]	Dissimilarity	[%]	Dissimilarity	[%]
[0, 0.1[	532	0,21	1,070	0,11	1,690	0,08	2,258	0,06
[0.1, 0.2[	158	0,06	404	0,04	780	0,03	1,114	0,03
[0.2, 0.3[	396	0,16	1,624	0,16	2,672	0,12	4,156	0,10
[0.3, 0.4[	1,002	0,40	4,300	0,43	6,928	0,31	11,326	0,28
[0.4, 0.5[	1,848	0,74	8,036	0,80	13,700	0,61	23,578	0,59
[0.5, 0.6[	2,662	1,06	12,706	1,27	22,194	0,99	41,040	1,03
[0.6, 0.7[	3,890	1,55	18,262	1,83	31,528	1,40	61,162	1,53
[0.7, 0.8[	5,928	2,36	27,304	2,73	47,618	2,12	91,126	2,28
[0.8, 0.9[	15,032	5,99	55,786	5,58	104,928	4,66	192,948	4,82
[0.9, 1.0[	212,220	84,55	849,288	84,93	1949,650	86,65	3,429,642	85,74
[1.0]	7,332	2,92	21,220	2,12	68,312	3,04	141,650	3,54

Table 3.3: Dissimilarity distribution with the cosine measure. Most document pairs are very dissimilar.

	Multicat500.txt		Multicat1000.txt		Multicat1500.txt		Multicat2000.txt	
	Documents	[%]	Documents	[%]	Documents	[%]	Documents	[%]
[0, 0.1[	546	0,22	1,078	0,11	1,680	0,07	2,272	0,06
[0.1, 0.2[	136	0,05	318	0,03	794	0,04	1,042	0,03
[0.2, 0.3[	318	0,13	1,154	0,12	2,572	0,11	3,394	0,08
[0.3, 0.4[	756	0,30	3,026	0,30	6,634	0,29	8,568	0,21
[0.4, 0.5[	1,280	0,51	5,904	0,59	12,946	0,58	16,856	0,42
[0.5, 0.6[	1,686	0,67	9,262	0,93	20,608	0,92	26,940	0,67
[0.6, 0.7[	2,484	0,99	13,496	1,35	29,370	1,31	39,308	0,98
[0.7, 0.8[	3,938	1,58	20,654	2,07	45,662	2,03	62,880	1,57
[0.8, 0.9[	12,906	5,16	49,862	4,99	108,394	4,82	171,834	4,30
[0.9, 1.0[	211,514	84,61	840,066	84,01	1856,206	82,5	3,323,340	83,08
[1.0]	14,436	5,77	55,180	5,52	165,134	7,34	343,566	8,59

Table 3.4: Dissimilarity distribution with the cosine measure. Most document pairs are very dissimilar.

Such a high ratio of dissimilar document pairs to similar document pairs is not desired. One way to overcome this problem could be to use another dissimilarity measure. Therefore a second run was performed. This time the Jaccard coefficient was used to calculate the dissimilarities. Tables 3.5 and 3.6 are similar to the above tables except that they show the distribution created by the Jaccard coeffecient. The Jaccard coefficient was tranformed into a dissimilarity measure by the formula $1 - s_{\text{Jaccard}}(\mathbf{o_1}, \mathbf{o_2})$. Similar to the cosine measure most document pairs are very dissimilar. Therefore a different dissimilarity measure did not produce better results.

	Singlecat500.txt		Singlecat1000.txt		Singlecat1500.txt		Singlecat2000.txt	
	Documents	[%]	Documents	[%]	Documents	[%]	Documents	[%]
[0, 0.1[	522	0,21	1,038	0,10	1,564	0,07	2106	0,05
[0.1, 0.2[	50	0,02	84	0,01	132	0,01	172	0
[0.2, 0.3[	72	0,03	194	0,02	246	0,01	408	0,01
[0.3, 0.4[	272	0,11	936	0,09	1,416	0,06	2344	0,06
[0.4, 0.5[	392	0,16	1,530	0,15	2,464	0,11	4572	0,11
[0.5, 0.6[	1,444	0,58	6,564	0,66	10,736	0,48	19744	0,49
[0.6, 0.7[	3,516	1,41	17,906	1,79	28,960	1,29	54606	1,37
[0.7, 0.8[	6,876	2,75	35,052	3,51	57,686	2,56	109128	2,73
[0.8, 0.9[	20,192	8,08	76,044	7,60	148,148	6,58	260002	6,5
[0.9, 1.0[	209,332	83,73	839,432	83,94	1,930,336	85,79	3405268	85,13
[1.0]	7,332	2,93	21,220	2,12	68,312	3,04	141650	3,54

Table 3.5: Dissimilarity distribution with the Jaccard coefficient. Most document pairs are very dissimilar.

	Multicat500.txt		Multicat1000.txt		Multicat1500.txt		Multicat2000.txt	
	Documents	[%]	Documents	[%]	Documents	[%]	Documents	[%]
[0, 0.1[	522	0,21	1,036	0,10	1,556	0,07	2,114	0,05
[0.1, 0.2[	50	0,02	72	0,01	118	0,01	136	0
[0.2, 0.3[	68	0,03	142	0,01	270	0,01	288	0,01
[0.3, 0.4[	228	0,09	644	0,06	1,366	0,06	1,502	0,04
[0.4, 0.5[	292	0,12	1,032	0,10	2,276	0,10	2,734	0,07
[0.5, 0.6[	958	0,38	4,456	0,45	9,750	0,43	11,856	0,3
[0.6, 0.7[	2,086	0,83	12,096	1,21	26,398	1,17	32,332	0,81
[0.7, 0.8[	3,962	1,58	23,764	2,38	53,550	2,38	66,138	1,65
[0.8, 0.9[	14,698	5,88	58,898	5,89	127,904	5,68	193,410	4,84
[0.9, 1.0[	212,700	85,08	842,680	84,27	1,861,678	82,74	3,345,924	83,65
[1.0]	14,436	5,77	55,180	5,52	165,134	7,34	343,566	8,59

Table 3.6: Dissimilarity distribution with the Jaccard coefficient. Most document pairs are very dissimilar.

Because of the poor dissimilarity distribution the above files are not good for the clustering tests. Therefore new files were created that should better be suited for clustering test runs. The file "Singlecat8913.txt" was analyzed and the 10 topics that represent the most documents were extracted. These topics were "acq", "coffee", "crude", "earn", "interest", "money-fx", "money-supply", "ship", "sugar", "trade".[40]

Based on the topics two new files were created: "Documents400.txt" and "Documents800.txt". For the file Documents400.txt forty documents were taken from each of the ten classes. This resulted in a collection size of 400 documents. Similar for the file Documents800.txt eighty documents were taken from each of the ten classes. This resulted in a collection size of 800 documents. Therefore the final files contain documents that are uniformly distributed over 10 topics. This input configuration should lead to well distinguished clusters. Tables 3.7 and 3.8 show the dissimilarity distribution of this configuration. The first row shows the distribution that was calculated like in the previous experiments. Again the number of document pairs that are very dissimilar dominates but this time the dominance is lower than in the previous collections.

The resulting rows show the distribution of the collections in the case that latent semantic indexing was performed. K refers to how many singular values were used for the LSI method. Using LSI some document pairs have dissimilarity values above 1.0. This means that the angle between these documents is greater than 90 degrees. In other words after LSI is performed on the document

[40] The collections where created in cooperation with the Chair of Knowledge based Systems at the University of Paderborn

vectors some components of some vectors may have negative values. At all the dissimilarity distribution of all document pairs is more equally distributed when LSI was used than without LSI.

	Documents400.txt Indexing without LSI		Documents400.txt Indexing with LSI					
			K=40		K=20		K=10	
	Documents	[%]	Documents	[%]	Documents	[%]	Documents	[%]
[0, 0.1[	430	0,27	762	0,48	1120	0,7	2892	1,81
[0.1, 0.2[	58	0,04	494	0,31	1476	0,92	3554	2,22
[0.2, 0.3[	174	0,11	760	0,48	2364	1,48	4694	2,93
[0.3, 0.4[	252	0,16	1264	0,79	3154	1,97	5904	3,69
[0.4, 0.5[	670	0,42	1796	1,12	4372	2,73	6746	4,22
[0.5, 0.6[	1416	0,89	2898	1,81	6320	3,95	8166	5,1
[0.6, 0.7[	3774	2,36	5006	3,13	8900	5,56	10138	6,34
[0.7, 0.8[	9928	6,21	10300	6,44	13018	8,14	12744	7,97
[0.8, 0.9[	31346	19,59	21592	13,5	19764	12,35	16592	10,37
[0.9, 1.0[	101748	63,59	42452	26,53	29582	18,49	21922	13,7
[1.0, 1.1[	10204	6,38	49686	31,05	36500	22,81	30594	19,12
[1.1, 1.2[			18334	11,46	20808	13,01	20196	12,62
[1.2, 1.3[			4016	2,51	9486	5,93	10414	6,51
[1.3, 1.4[			596	0,37	2580	1,61	4020	2,51
[1.4, 1.5[			42	0,03	490	0,31	1184	0,74
[1.5, 1.6[			2	0	66	0,04	214	0,13
[1.6, 1.7[							24	0,02
[1.7, 1.8[							2	0

Table 3.7: Dissimilarity distribution of Documents400.txt. The dissimilarities were calculated by 1-cosine between two documents.

Some historical remarks should be mentioned. Indexing documents is a very old discipline and was probably performed since the first libraries exist. Clearly, in the first time this process was performed manually. The first approaches to automatic text analysis were made in the 1950s leading to the term frequency measure.[41] The inverse document frequency measure was introduced in the 1970s.[42] More recently linguistic methods are developed and still subject to current research. Because of the many rules that are necessary to describe natural languages they base on huge databases that map semantic relationships between words.[43]

[41] See [Luh58]

[42] See [Spa72]

[43] See for example http://www.cogsci.princeton.edu/~wn/

A question that remains is if the results of automatic indexing methods are equal to the results of manual indexing methods. Besides the fact that automatic indexing is faster and cheaper it was shown that automatic indexing programs produce results that are at least as good as manual indexing results.[44]

	Documents800.txt Indexing without LSI		Documents800.txt Indexing with LSI					
			K=40		K=20		K=10	
	Documents	[%]	Documents	[%]	Documents	[%]	Documents	[%]
[0, 0.1[	937	0,15	2002	0,31	3302	0,52	12958	2,02
[0.1, 0.2[	184	0,03	1854	0,29	5866	0,92	15376	2,4
[0.2, 0.3[	616	0,1	3002	0,47	7910	1,24	18502	2,89
[0.3, 0.4[	1026	0,16	4488	0,7	11472	1,79	20622	3,22
[0.4, 0.5[	2684	0,42	7248	1,13	15358	2,4	25082	3,92
[0.5, 0.6[	6008	0,94	12194	1,91	22678	3,54	30934	4,83
[0.6, 0.7[	14814	2,31	21102	3,3	32930	5,15	39006	6,09
[0.7, 0.8[	40678	6,36	42058	6,57	51082	7,98	49370	7,71
[0.8, 0.9[	129042	20,16	89258	13,95	81878	12,79	65686	10,26
[0.9, 1.0[	405056	63,29	168682	26,36	121460	18,98	88842	13,88
[1.0, 1.1[	38955	6,09	188766	29,49	144236	22,54	115296	18,02
[1.1, 1.2[			76438	11,94	86774	13,56	79028	12,35
[1.2, 1.3[			19686	3,08	39670	6,2	45616	7,13
[1.3, 1.4[			2930	0,46	12638	1,97	21244	3,32
[1.4, 1.5[			280	0,04	2348	0,37	8508	1,33
[1.5, 1.6[			10	0	354	0,06	2730	0,43
[1.6, 1.7[			2	0	42	0,01	848	0,13
[1.7, 1.8[					2	0	198	0,03
[1.8, 1.9[							132	0,02
[1.9, 2.0[							22	0

Table 3.8: Dissimilarity distribution for Documents800.txt. The dissimilarities were calculated by 1-cosine between two documents.

[44] See [SM87, page 65]

3.3 Formalization of the output of the document clustering process

Now the output of the clustering process is formalized. Recall from chapter two that there are two possible classification schemes to cluster documents, the hierarchical and the non-hierarchical.

3.3.1 Nonhierarchical clustering

Definition (Partition)

Let V be a set. A set P of subsets is called a *partition* of V if

1. $C_1 \cap C_2 = \varnothing$ for all $C_1, C_2 \in P$
2. $\bigcup_{C \in P} = V$

■

Definition (Set of all partitions)

Let $\wp(\mathrm{V})$ be the *set of all partitions* of V.

■

Fig. 3.9 a shows a data set with a partition on it. Every point is assigned to exactly one cluster. In this case a partition of the data can easily be determined because the clusters are well distinguished and a partition on the data does very well represent the structure. In contrast Fig. 3.9b shows another example. The point x between the two clusters has to be assigned either to the top cluster or to the bottom cluster. But a decision for only one of the two clusters would not be accurate. The resulting partition would not highlight the data structure very well. This is a situation that occurs in document clustering very often. Usually a document cannot be assigned to exactly one class because its concepts may be described by multiple overlapping topics. The problem can be solved by relaxing the constraint that the clusters have to be disjoint, which leads to a non-exclusive partition. However, this does not solve the situation of point y. It has nearly the same distance to the top cluster than to the bottom cluster. Thus it is in the same situation than point x. But point y is far away from both clusters. Therefore a full membership in one or both of them would not be accurate. Such a situation is also very common in document clustering. Usually the topics in a document are not equally important, some are discussed exhaustive, others are only shortly mentioned. Take for example this text. In chapter two a short introduction and examples of classifiers were given. It was not an exhaustive discussion, just a brief introduction. Therefore this text should be part of a cluster that represents documents about classifiers but not to the same degree than documents that provide in depth knowledge about them. Thus besides non-exclusive clusters it is desirable to quantify the degree of membership of a document in a certain cluster. One approach to do this is to apply fuzzy logic.[45]

[45] See appendix A for a brief introduction to fuzzy logic

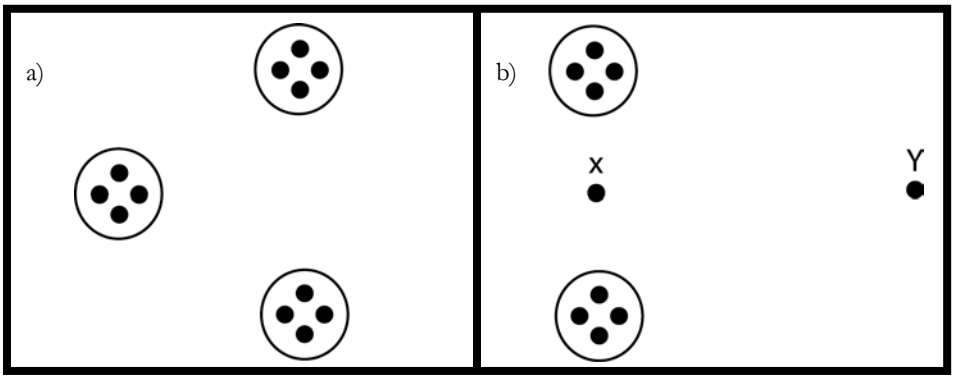

Figure 3.9: a) A partition on a well distinguished data set. b) The point x is in the middle of the two clusters and may belong to the upper cluster as well as to the bottom cluster. It would be accurate to assign it to both clusters. The point y may also belong to both clusters but not with the same degree than the other points.

Definition (Fuzzy-partition)

Let V be a set and $P = \{C_1, C_2, ..., C_k\}$ be a set of fuzzy sets and $u_j, j = 1, 2, ..., k$ be the membership function of C_j. P is called a *fuzzy partition* of V, if

1. $u_j(v_i) \in [0,1]$ for all j = 1, 2, …, k; i = 1, 2, …, n
2. $\sum_{i=1}^{n} u_j(v_i) > 0$ for all j = 1, 2, …, k
3. $\bigcup T(C_j) = V$ for all j = 1, 2, …, k

■

The first condition of the above definition constrains the membership values of the membership function to be in the interval [0,1]. The second condition guarantees that there is no empty fuzzy set. The third condition constrains the members of the different fuzzy sets to be from V. A fuzzy partition with the above properties is also called possibilistic fuzzy-partition. The term possibilistic is used because the membership values of the elements represent degrees of possibility, sometimes also called degrees of typicality.

Definition (Set of all fuzzy partitions)

Let $\wp_{Fuzzy}(V)$ be the set of all fuzzy-partitions of V. ■

3.3.2 Hierarchical clustering

Definition (Fuzzy-dendrogram)

Let $p : \mathbb{N}_0 \to \wp_{Fuzzy}(V)$ be a function. p is called a *fuzzy-dendrogram* if

1. $h \le h' \Rightarrow \exists C_1, C_2 \in p(h) : C_1 \neq C_2 \wedge (C_1 \cup C_2) \in p(h')$ for $h, h' \in \mathbb{N}_0$
2. $\exists h \in \mathbb{N} : p(h) = \{\{V\}\}$ ■

Informally a fuzzy-dendrogram is a hierarchy of fuzzy-partitions. It has the form of a tree, whereas the root is a single cluster that contains all elements. Every child of a node is a cluster nested in the parent cluster.

3.4 Summary

This chapter introduced the basics of document clustering. The main part dealt with the preparation of the documents prior the clustering process. The commonly used vector-space model was introduced so that possibilities to impose a similarity measure on the documents were opened. After that the desired output of a document clustering process was discussed and formalized. Fuzzy logic was introduced to cope with the properties of document clusters. In the next chapter some clustering algorithms are discussed that work on the above introduced formal structures.

Chapter 4

FUZZY-CLUSTERING ALGORITHMS

There are many approaches to cluster data but there is no common taxonomy of clustering algorithms. Usually clustering algorithms can be distinguished by a couple of properties that may overlap. In the following some of these properties, which are important for the algorithms described in this chapter, are shortly described:

- *hierarchical clustering vs. non-hierarchical clustering:* This division was made and formally described in the previous two chapters. However, an algorithm that was created for non-hierarchical clustering can also be used to create hierarchies. Therefore the algorithm starts to partition the original data. After that the algorithm performs the same steps within each cluster found in the previous step. This is proceeded until an appropriate depth of the tree is reached. Clearly, such an algorithm has exponential runtime behavior and would not be useful. Thus only data of "interesting clusters" should be fine-grained. Therefore "interesting clusters" have to be identified in some way. One possibility is to delegate the task to the user. The user are confronted with the non-hierarchical cluster structure. In the following he or she decides which cluster is interesting for him or her and the algorithm performs a second clustering in the selected cluster. It is also possible to create a non-hierarchical clustering structure from a dendrogram. Recall from the previous chapters that a dendrogram is nothing else than a hierarchy of partitions. Therefore one chooses one layer of the tree to be the final non-hierarchical clustering. The difficulty is to define a criteria that decides what layer is the best cluster structure. The algorithms described in this chapter are non-hierarchical.

- *objective function based clustering vs. non objective function based clustering:* Objective function based clustering algorithms optimize an objective function that describes the structure of the given data. For example the structure may be described by the average distance between all objects of a cluster. In this case the algorithm minimizes the average distance. Usually the objective functions cannot be optimally resolved. In this case clustering algorithms search for a local optimum of the objective function. Non-objective function based clustering algorithms do not optimize any function. The algorithms described in this chapter are belonging to the objective function based clustering algorithms.

- *exclusive clustering vs. non-exclusive clustering:* Exclusive clustering algorithm assign every object to exactly one cluster while non-exclusive clustering algorithm allow an object to be member of more than one cluster. Like described in the previous chapter, in the case of document clustering it is desired to have non-exclusive clusters. In this chapter algorithms of both types are described.

In the following this chapter focuses on the k-prototype clustering algorithm and its fuzzy versions. Therefore this chapter starts by describing the underlying ideas of the k-prototype algorithm. After that the fuzzy version are introduced.

Remark

In the following of this chapter let V be a set of objects. If $v \in V$ is an object, then $\mathbf{v}$ denotes the corresponding vector to v in the $\mathbb{R}^n$. Let $\mathbf{v} \in V$ denote that the corresponding object v to $\mathbf{v}$ belongs to V. ■

4.1 Basic considerations

Definition (set of all k-partitions)

Let V be a set of objects. $\wp_k(V)$ is the set of all k-partitions of V. ■

The idea of k-prototype clustering is to search for a good k-partition in the set of all possible k-partitions of an object set. Therefore a quality measure is imposed on $\wp_k(V)$ that enables the comparison of two k-partitions. The quality measure enables the definition of an objective function that has to be maximized by the algorithm. The basic functionality of all k-prototype variations is similar. Starting from an initial k-partition the algorithm moves successive from one k-partition to a new higher quality k-partition until a local optimum is found.

Given a k-partition $P = \{C_1, C_2, ..., C_k\} \in \wp_k(V)$ on the object collection V, the problem is how to chose the prototype of each cluster so that it best represents the other objects. Usually the cluster is either represented by the mean of all cluster members or by a typical object of all cluster members.

4.1.1 Representation by the mean

One approach is to represent a cluster by the mean of all cluster members. This mean is also called the *centroid* of cluster $C_j, j = 1, 2, ..., k$ and is calculated by

$$\overline{\mathbf{v}}_{C_j} = \frac{1}{|C_j|} \sum_{v \in C_j} \mathbf{v} .$$

The runtime complexity of calculating the mean of all members of a particular cluster is $O(n_j)$, where n_j is the number of members of cluster C_j. Therefore the complexity of calculating the mean of all clusters of a k-partition is $O(k*|C_{max}|)$, where $|C_{max}|$ is the cluster with the maximal number of members.

The quality of a cluster that is represented by a centroid can be calculated by the standard derivation:

$$\sigma(\bar{\mathbf{v}}_{C_j}) = \frac{1}{|C_j|}\sum_{\mathbf{v}\in C_j} d(\mathbf{v}, \bar{\mathbf{v}}_{C_j})$$

The lower the standard derivation the less different are the members of the cluster from the mean and thus the higher is the quality of the cluster representation. In other words if the standard derivation is low the majority of the cluster members are very close to the mean. Therefore they have only slightly differing feature values compared to the mean.

The main advantage of this cluster representation approach is the low runtime complexity for the mean calculation. The main disadvantage is that the feature vectors of the objects are needed to calculate $\bar{\mathbf{v}}_{C_j}$ and that the features have to be interval-scaled values.[46]

4.1.2 Representation by the medoid

If the feature vectors of the objects are unknown the mean cannot be calculated. In this case the mean cannot be used as cluster representation. Another approach that overcomes this problem is to represent a cluster by a typical object from its members. Such an object is called *medoid* m_j of cluster C_j.

The difficulty of this approach is to decide what object is typical for the cluster. Not every object within a single cluster represents its cluster with the same quality. Fig. 4.1 shows two examples in the two dimensional space. The cluster of Fig. 4.1a consists of three points. The medoid is the left most point. It has a moderate distance to the middle point and a high distance to the right point. This means that the features of the right point are considerable different from the features of the medoid. Therefore the medoid does not represent the right point very well. Fig. 4.1b shows the same cluster but with the middle point as medoid. In this case the medoid is more centrally located and has a moderate distance to the left as well as to the right point. The features between the points

[46] See [KR90, page 112]

and the medoid are less different than in Fig. 4.1a. Thus this representation should be more accurate than the previous one.

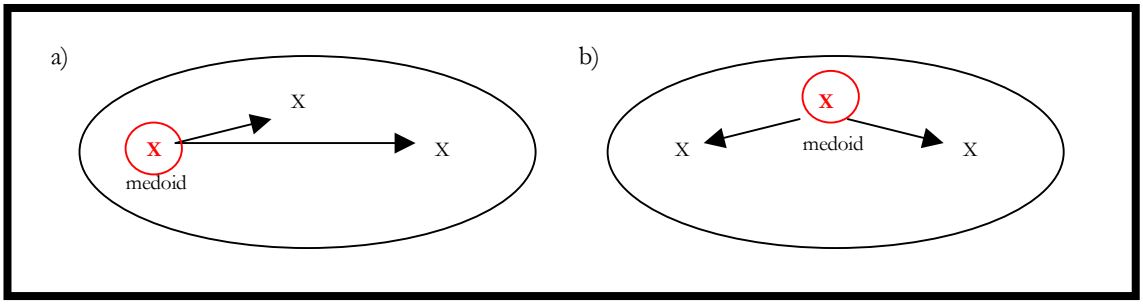

Figure 4.1: A cluster with different medoids. a) The left most point is the medoid. It has a moderate distance to the middle point and high distance to the right point. b) The middle point is the medoid. It has a moderate distance to the other points.

Because every object in the cluster is associated with the medoid, the dissimilarity between a point and its medoid can be regarded as an error of the representation. Let m_j be the medoid of cluster C_j, $j=1,...,k$. Thus the representation error of m_j is calculated by

$$e(m_j) = \sum_{v \in C_j} d(v, m_j), \tag{4.1}$$

the sum of the dissimilarities between the medoid and the members of the cluster. Within a single cluster the object m_j^* that minimizes the above sum has the lowest representation error and can best represent the cluster. Therefore it should be the medoid.

Let n_j be the number of objects in cluster C_j. The complexity of calculating the optimal medoid m_j^* of a single cluster is $O(n_j^2)$ because every object in the cluster has to be compared with all the other objects in the cluster. The complexity of calculating the optimal medoids for all clusters of a partition is $O(k^*|C_{max}|^2)$, where C_{max} is the cluster with the maximal amount of objects. Therefore it is possible to calculate the minimal representation error for all clusters of a partition in an acceptable time. However, compared to the mean calculation the algorithms are slower. In the following let $e(m_j^*)$ be the minimal possible representation error of cluster C_j.

K-prototype clustering algorithms that use the mean as prototype are usually called k-means clustering. K-prototype clustering algorithms that use a medoid for the cluster representation are called k-medoid clustering. Sometimes the letter c is used instead of k to refer to the algorithms, i.e. c-means clustering. The letter k or c shows the number of prototypes at the beginning of the clustering process.

4.2 K-Medoid clustering algorithm

4.2.1 Objective function

Let $M_P=\{m_1,m_2,...,m_k\}$ be a set of medoids of the k-partition P. The notion of the representation error of a single cluster can be generalized to the representation error of the partition P:

$$e(P,M_P)=\sum_{j=1}^{k} e(m_j)=\sum_{j=1}^{k}\sum_{v\in C_j} d(v,m_j), \text{ and}$$

$$e(P)=\sum_{j=1}^{k} e(m^*{}_j)=\sum_{j=1}^{k}\sum_{v\in C_j} d(v,m_j^*), \text{ for all } P\in\wp_k(V). \tag{4.2}$$

The value of e(P) depends only on P because for a given k-partition the optimal medoids can be calculated, which was shown in the previous subsection. e(P) defines an objective function that assigns every partition $P\in\wp_k(V)$ a value that quantifies its quality. The lower the value of e the higher the quality of the particular partition. The goal is to find the partition that minimizes e. However, a complete enumeration of all possible k-partitions is not a good idea due to the size of the search space, which can be calculated by $k^N/k!$.[47] E.g. for 50 objects the possible 5-partitions are already approximately $7.4*10^{32}$. Applied to document clustering one sees that even for 50 documents the enumeration cannot be performed in an acceptable time and in practical document clustering applications the amount of documents usually exceeds the number of 50 by far.

Because the enumeration of the search space is not possible it is necessary to use a greedy approach that finds a partition with a low, but not optimal e. The approach starts with a given k-partition $P_1\in\wp_k(V)$ with optimal medoids and searches a partition $P_2\in\wp_k(V)$ in the neighborhood of P_1 that has a lower representation error. The neighborhood of P_1 consists of all k-partitions with the same set of medoids. Notice that despite the medoids are optimal for P_1 they need not to be optimal for the neighborhood partitions. The next lemma helps to identify such a partition P_2.

Lemma

Let $P_1=\{C_1,C_2,...,C_k\}\in\wp_k(V)$ be a k-partition with the optimal medoids $M_{P_1}^*=\{m_1^*,m_2^*,...,m_k^*\}$. Let $v\in C_h, h=1,2,...,k$. If there exists a cluster $C_j\in P_1, j=1,...k, j\neq h$, so that $d(v,m_j)<d(v,m_h)$, then moving v from cluster C_h to C_j creates a k-partition P_2 with $e(P_2,M_{P_1}^*)<e^*(P_1)$. ■

[47] See [KR90, page 115]

Proof

Let $d(v,m_j) < d(v,m_h)$. If v is moved from C_h to C_j, then $e(P_2, M^*_{P_1}) = e(P_1) - d(v,m_h) + d(v,m_j) < e^*(P_1)$. ■

Let M be a set of medoids. From the lemma follows that if every object is assigned to the cluster of the closest medoid the resulting k-partition P_{opt} has the lowest $e(P_{opt}, M_{P_1})$. The complexity of assigning every object to the closest medoid is $O(k*n)$ because every object has to be compared to every medoid.

4.2.2 K-Medoid algorithm

Based on the ideas discussed until now a greedy algorithm can be formulated that finds a partition so that $e^*(P)$ is a local minimum.[48]

1. The algorithm starts with choosing a partition $P_{t=1}$ of V and k medoids $\{m_1, m_2, ..., m_k\}$, k<|V| randomly. The value of t is the iteration number.
2. Every object is assigned to the cluster of the closest medoid. This creates a partition P_{t+1} with $e(P_{t+1})$ lower or equal to $e(P_t)$ for a given set of medoids (see Lemma).
3. Because in step two a new partition was created, the actual medoids may not be the best choice for representing the clusters anymore. Therefore for every cluster C_j the optimal medoid is calculated. If the new medoids are identical with the old ones the algorithm stops. Else it continues with step 2.

The question that remains about the above algorithm is if it always converges. It is easy to see that it always converges. In every iteration step a new higher quality partition-medoid combination is found. Therefore a once abandoned combination cannot occur a second time. Because the set of possible medoids and the set of possible k-partition is finite, the set of all k-partition-medoid combinations is also finite. Therefore the algorithm converges.

4.3 K-Means clustering algorithm

Similar to the k-medoid clustering algorithm the objective function of the k-means algorithm is defined as

$$\sigma(P) = \sum_{j=1}^{k} \sigma(\bar{\mathbf{v}}_{C_j}) = \sum_{j=1}^{k} \sum_{\mathbf{v} \in C_j} d(\mathbf{v}, \bar{\mathbf{v}}_{C_j})$$

[48] See [KJY99]

The k-means clustering algorithm is similar to the k-medoid algorithm in that the mean can be considered as an optimal medoid. Therefore the ideas and the algorithm described in the previous subsection apply also to the k-means algorithm.

4.4 Fuzzy K-medoid

The above algorithm works on an objective function that assumes that an object belongs to exactly one cluster. Before a fuzzy algorithm can be formulated the objective function has to be adapted to work on fuzzy-partitions. In the following two possible fuzzy objective functions are discussed.

4.4.1 First objective function

The objective function can be extended by weighting the dissimilarities between an object and its medoid by the degree of membership the object belongs to the cluster. In the following let u_j be the membership function of cluster C_j and $u_{j,i}$ denote the membership value of v_i in cluster C_j, then the fuzzy objective function is

$$e_{Fuzzy}(P) = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{j,i}^{m} * d(v_i, m_j). \tag{4.3}$$

The goal is to minimize the fuzzy objective function. If one chooses the membership values of all objects to be zero the function equals zero and therefore reaches its minimum. However this trivial solution is not desired. To overcome this problem the restriction is added that the sum of all membership values of a particular object has to equal one:[49]

$$\sum_{j=1}^{k} u_j(x_i) = 1, \text{ for all i = 1, …, n} \tag{4.4}$$

To understand the logic behind this restriction a closer look at the definition of a crisp partition has to be made. From the definition of a classical partition the restriction follows directly. The definition forces an object to be a member of exactly one cluster. Therefore the particular object has a membership value of one in the cluster it belongs to and membership values of zero in all other clusters. Thus the sum of the membership values equals one and the restriction holds. Because of this the restriction is added to the definition of a fuzzy-partition.[50] Clustering methods that create fuzzy-partitions with this restriction are also called probabilistic fuzzy-clustering methods. The name shows that the membership values can be interpreted as probabilities or degrees of sharing. Consider, the membership values are not absolute. This means that if an object

[49] See [KK93]

[50] See [KK93]

has to be split over many clusters the membership values will usually be low compared to an object that has to be split over a few clusters.

The exponent $m \in \mathbb{R}_{>1}$ in (4.3) is called fuzzifier. The greater one chooses m the less developed will be local minima of e_{Fuzzy}.[51] The right choice of m depends on the estimation of how well the data can be divided into clusters. If the clusters are easy distinguishable, one should chose m close to one.[52] In contrast, if the clusters are hardly distinguishable, one should chose a large m.[53] In practice it is common to use m=2.[54]

It can be shown that the following membership function leads to the optimal fuzzy-partition for a given set of medoids:[55]

$$u_{j,i} = \frac{(\frac{1}{d(v_i, m_j)})^{1/(m-1)}}{\sum_{l=1}^{k} (\frac{1}{d(v_i, m_l)})^{1/(m-1)}} \tag{4.5}$$

The function can be generated from the objective function by calculating the first derivation and setting it to zero. It calculates the degree of membership for the object v_i in the cluster C_j. The value of the numerator determines the membership value of object v_i in cluster C_j. It depends on the dissimilarity between an object v_i and the medoid m_j. The more dissimilar v_i and m_j are, the lower is the value of the numerator and therefore the lower is the membership of object v_i in cluster C_j. The denominator is used to normalize the membership values so that they satisfy the equation $\sum_{j=1}^{k} u_{i,j} = 1$, for all $i = 1,...,n$. For the case that an object is similar to one of the medoids, namely $d(v_i, m_j) = 0$, the point is assigned with membership value one to the cluster of this medoid and with membership value zero to the remaining clusters. Compared to the crisp version the runtime complexity of the task that assigns every object to the clusters stays $O(k*n)$ because for every object the membership value in every cluster has to be calculated.

The representation error of every medoid can be calculated by

---

51 See [Fuzzy cluster analysis, page 21]

52 See [Fuzzy cluster analysis, page 21]

53 See [Fuzzy cluster analysis, page 21]

54 See [Fuzzy cluster analysis, page 22]

55 See [KJY99]

$$e_{Fuzzy}(m_j) = \sum_{v \in V} u_{i,j}^m * d(v, m_j).$$

The dissimilarities between the medoid and the objects is weighted by the membership value of the object. The runtime complexity of calculating the optimal medoid of a cluster C_j increases to $O(n^2)$ because every object belongs to a certain degree to C_j and therefore influences the representation error. The runtime complexity of calculating all medoids increases to $O(k*n^2)$.

The following algorithm finds a local minimum of (4.3):[56]

```
Fix the number of clusters k;
Randomly pick the initial set of medoids:
  M = {m_1, m_2, ..., m_k}
Repeat
  for j=1 to k
    for i=1 to n
      Compute u_{j,i} by using (4.5);
    endfor
  endfor
  Store the current medoids: M^old = M ;
  Compute the new medoids:
  for j=1 to k
    q = argmin_{1<h<n} Σ_{i=1}^{n} u_{j,i}^m * d(v_i, m_h) ;
    m_j = v_q ;
  endfor
Until (V^old = V)
```

The above discussion showed that the computation of the membership values has cost $O(k*n)$ and the calculation of the new medoids $O(k*n^2)$. Therefore the algorithm has a runtime complexity of $O(n^2)$ in each iteration step. There is no mathematical proof about the speed of convergence known to the author. But in practical experiments the algorithm converged relatively fast, usually in five or six iterations.[57]

The membership values calculated by (4.5) are relative values. This leads to some problems that are visualized by Fig. 4.2.[58] Fig 4.2a shows a data set with two natural clusters. The point x should have the same membership value in cluster one as point y has because they are equally distant from the

[56] See [KJY99]

[57] See [KJY99]

[58] Examples are taken from [KK93]

center of cluster one. But because the membership values are interpreted as degrees of sharing point x has a high membership value in cluster one and a low membership value in cluster two while point y has a moderate membership value in both clusters. Fig 4.2b visualizes another problem. Clearly, point x is far away from the two clusters and therefore should not be associated with one of them. However it has a membership value of one half in both clusters.

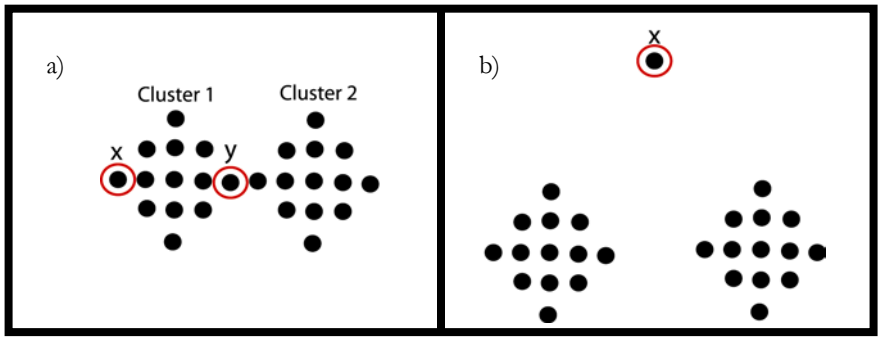

Figure 4.2: a) The points x and y should have the same membership value in cluster 1 but point y has a lower membership value than point x b) The point x should have a low membership value in both clusters but has membership values of one half in each.

4.4.2 Second fuzzy objective function

The above algorithm uses fuzzy logic to avoid a crisp decision in every iteration step. But after the algorithm terminates the relative membership values have to be interpreted in some way. Usually they are used to create a crisp partition.[59] E.g. a possibility to defuzzify the fuzzy-partition is to assign every object to the cluster with the highest degree of membership. For some applications, like pattern classification and image segmentation, this is appropriate. For this kind of applications the above algorithm is very successful.[60] But in the previous chapter it was stated that the result of a document clustering process should not be a crisp partition. Moreover it is desired to have absolute membership values to quantify the typicality of a document in a certain cluster. Thus in document clustering the restriction (4.4) is not appropriate.

The restriction cannot be eliminated because this leads to the trivial solution. Therefore another objective function is needed that is somehow similar to the above one but does not need the restriction. [KK93] suggested the following objective function:

$$e_{Fuzzy}(\mathrm{P}) = \sum_{j=1}^{k}\sum_{i=1}^{n}(u_{j,i})^m * d(v_i, m_j) + \sum_{j=1}^{k}\eta_j * \sum_{i=1}^{n}(1-u_{j,i})^m$$

[59] See [KK93]

[60] See [KK93]

The first part of the function is equal to the previous one. It forces the dissimilarities to be as small as possible. The second part forces the membership values to be as large as possible. This avoids the trivial solution. It can be shown that for a given set of medoids the objective function reaches its minimum if the membership values are chosen to be[61]

$$u_{j,i} = \frac{1}{1 + \left(\frac{d(v_i, m_j)}{\eta_j} \right)^{1/(m-1)}} \tag{4.6}$$

The parameter η_j determines the dissimilarity at which the membership value of an object in a cluster becomes 0.5.[62] The value can be different for every cluster. It is recommendable that η_j relates to the overall size and shape of a cluster. [KK93] describes the following formula to work well:

$$\eta_j = \frac{\sum_{i=1}^{n} u_{j,i}^m * d(v_i, m_j)}{\sum_{i=1}^{n} u_{j,i}}. \tag{4.7}$$

It calculates the average *intra-cluster dissimilarity*. Another possible formula is:

$$\eta_j = \frac{\sum_{v_i \in (C_j)_\alpha} d(v_i, m_j)}{|(C_j)_\alpha|} \tag{4.8}$$

If $(C_j)_\alpha$ is an appropriate α-cut, the formula calculates the average intra-cluster dissimilarity for all good objects. It is not necessary to update the η_i in every iteration step because the overall result is very robust against variations of η_i.[63] Therefore the η_i should be calculated for the initial partition and stay constant until the algorithm terminates. For an improvement of the final result [KK93] suggests a combination of both formulas. In this case the algorithm is run two times. In the first run η_i is estimated for the initial partition by (4.7). After that the algorithm is run using the estimated values for η_i. After termination the η_i are estimated again using formula (4.8). With the new values the algorithm is run a second time. The second run is a fine-tuning process and should be performed if the shapes of the clusters are important.[64]

---

[61] See [KK93]

[62] See [KK93]

[63] See [KK93]

[64] See [KK93]

Because the second part of the objective function is independent of the dissimilarity values the conditions on the medoids for optimization stay the same.[65] Therefore the same algorithm than for the first objective function can be used for optimization. The only difference is that the membership values are calculated by (4.6). Because the membership values are absolute, the algorithm is also called possibilistic fuzzy k-medoid.

Coincident clusters

It was criticized that the possibilistic fuzzy k-medoid algorithm may lead to coincident clusters.[66] The reason is that the membership functions of the single clusters are independent of each other. Therefore the clusters move independently of each other to dense regions in the feature space while the algorithm runs. If two clusters are in the same dense region at the beginning they will move to the same point. On the other side this effect can also be seen as an advantage.[67] In most clustering problems the number k of clusters is not known in advance. Therefore the algorithm may start with a k that is definitely larger than the real number of clusters. After the algorithm terminates, coincident clusters are merged. This leads to a more accurate fuzzy-partition. However, because the clusters move independent of each other the algorithm depends on a good choice of the initial medoids. Fig. 3.3 visualizes this problem. In the data are three cluster. Assume the initial medoids are x, y, and z. Because x and y are initially in cluster two both will move to the center of cluster two. Merging both clusters after the algorithm terminates suggests to the user that there are only two clusters, while there are three. Therefore the initial medoids have to be chosen so that every dense region contains at least one of them. A solution to this problem is to use the probabilistic fuzzy k-medoid algorithm for initialization.[68]

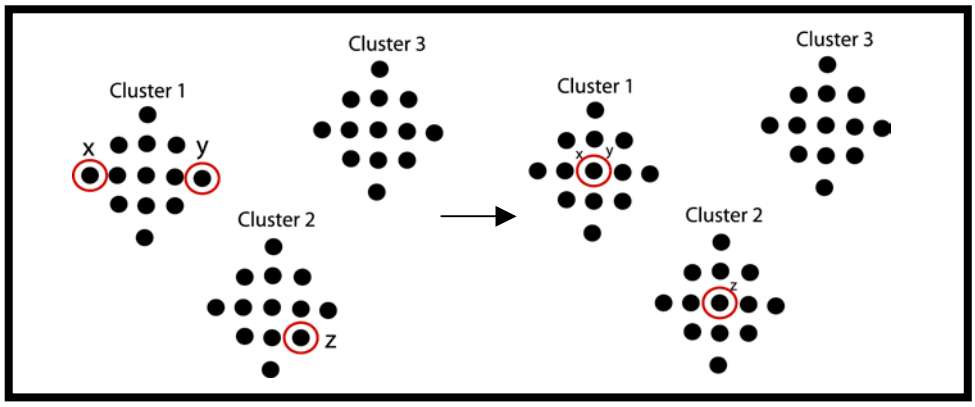

Figure 4.3: In the case of possibilistic fuzzy clustering the medoids are independent of each other. Therefore they are moving to the center of the next dense region.

[65] See [KK93]

[66] See [BCM96]

[67] See [KK96]

[68] See [KK96]

4.4.3 Runtime considerations

As discussed above the runtime complexity of the fuzzy k-medoid is $O(n^2)$ for each iteration step. This may lead to problems if large collections of objects have to be clustered. To overcome this problem a combination of clustering and classification can be applied. Therefore one starts to cluster a subset of the original data to create a classification schema. The remaining objects are assigned to the clusters using a fuzzy classifier, for example, the fuzzy nearest prototype algorithm.[69] This classifier assigns each of the remaining objects to the clusters using formula (4.5) for probabilistic fuzzy clustering or formula (4.6) for possibilistic clustering. This very simple classifier produces useful results.[70] The main advantage is the linear runtime complexity because every object has only to be compared to the cluster prototypes.

4.5 Fuzzy k-means

The above ideas apply also to the probabilistic fuzzy k-means and the possibilistic fuzzy k-means version. The are two differences to the fuzzy k-medoid algorithms: the calculation of the cluster prototypes and the criterion for termination of the algorithm.

First, instead of a medoid the mean is used for representation. Because the calculation of the mean requires lower runtime the overall runtime complexity of the algorithms is lower than for the fuzzy k-medoid versions. The calculation of the mean has cost O(k*n). To calculate the new memberships the algorithm needs O(k*n). Therefore the overall runtime complexity of the fuzzy k-means is $O(k*n)$ in every iteration step.

Second, the termination criterion for the algorithm is different. The algorithm terminates if $|U(t+1)-U(t)|<\varepsilon$, whereas ε is a small positive constant and $|\bullet|$ is a matrix norm.[71]

4.6 Summary

This chapter introduced the k-medoid and the k-means algorithms and their fuzzy versions. The algorithms are similar in that they consist of two steps. First for every cluster the new prototype is calculated and second the objects are assigned to the cluster using formula (4.5) or (4.6). The two steps are repeated until a certain criterion is reached. Fig. 4.5 summarizes the runtime complexity of the four discussed fuzzy algorithms.

---

[69] See [KGG85]

[70] See [KGG85]

[71] See [MF98, pp.376-378]

		Cluster representation	
		Mean	**object**
Result	**probabilistic**	O(k*n)	$O(k*n^2)$
	possibilistic	O(k*n)	$O(k*n^2)$

Figure 4.4: Runtime complexity of the fuzzy k-prototype algorithms. n is the number of objects that are clustered and k is the number of clusters.

Some historical remarks should be mentioned. Historically the probabilistic fuzzy k-means algorithm was introduced first followed by the probabilistic fuzzy k-medoid. The algorithms were developed in the 1970s. [Bez81] gives an overview about these developments and other algorithms that base on fuzzy objective functions. The possibilistic versions were introduced in 1993.[72]

[72] See [KK93]

Chapter 5

EXPERIMENTAL RESULTS

This chapter discusses some experiments that base on the algorithms described in the previous chapter and their results. Therefore this chapter starts to describe some performance measures for clustering algorithms. After that the implementation details of the program are presented. Then, the experiments and the results are described. The chapter closes with a discussion about the results.

5.1 Performance measures

Besides runtime considerations that were discussed in the previous chapter the quality of the produced clusters is important. Measures that access the quality of clusters are split into internal and external methods.[73] Internal measures do not use more knowledge about the clustered data than the output of the algorithm to evaluate the quality. External measures use additional knowledge for the evaluation. E.g. if the best possible cluster structure of a data set is known the measure can use this knowledge to access the quality of the clusters created by the algorithm in relation to the optimal clusters. In the following two quality measures for cluster structures are introduced.

5.1.1 F-measure

The F-measure is an external quality measure. It bases on the idea of the precision and recall measures that are commonly used in information retrieval to evaluate the quality of query responses.[74] The measure matches known optimal classes to the created clusters. A single cluster $C_{cluster,\,j}$ is seen as the result of a query and a class $C_{class,\,i}$ as the set of relevant documents to the query. There are four possible cases if a cluster is matched to a class (Fig. 5.1). A document may be in the cluster and the class (case a), only in the cluster but not in the class (case b), only in the class but not in the cluster (case c), and a document may not be in the cluster and the class (case d).

[73] See [SKK00]

[74] See Chapter 2

	in cluster	**not in cluster**
in class	a	b
not in class	c	d

Figure 5.1: The four possible cases for a particular document if a cluster is compared to a class

Let a be the number of documents of case a, b the number of document of case b, c the number of documents of case c, and d the number of documents of case d. The recall and precision measures from chapter two can be adapted as follows:

$$\text{Recall} = \frac{a}{a+b}$$

$$\text{Precision} = \frac{a}{a+c}$$

Using only one of these measures for evaluation would not be accurate because a high value in precision usually causes a low value in recall and vice versa. Therefore a combination of both measures is used for evaluation, the so called F-measure:[75]

$$F(C_{class,\,i}, C_{cluster,\,j}) = \frac{(1+p^2) * \text{Precision}\,(C_{class,\,i}, C_{cluster,\,j}) * \text{Recall}\,(C_{class,\,i}, C_{cluster,\,j})}{p^2 * \text{Precision}(C_{class,\,i}, C_{cluster,\,j}) + \text{Recall}(C_{class,\,i}, C_{cluster,\,j})}$$

The F-measure takes value in the interval [0,1]. The higher the value of F the better corresponds class $C_{class,\,i}$ to cluster $C_{cluster,\,j}$. A value of one means a perfect correspondence between class $C_{class,\,i}$ and cluster $C_{cluster,\,j}$. The parameter p determines the importance of precision in relation to recall. The parameter is used to model varying preferences of different users. E.g. a value of 0.5 for p indicates that a user is twice as interested in precision as in recall. A value of 2 indicates that a user was twice as interested in recall as precision.[76]

In this work the measure is applied to a created partition in the following way: First, for every "optimal" class the F value is calculated for every found cluster. The lowest F value is used for the

[75] See [FB92, page 11]

[76] See [FB92, page 11]

class. Second, the mean of all F-measure values is used to access the overall quality of the partition. Clearly, the higher the F-measure mean the better the quality of the partition. Formally this means:

$$F := \frac{1}{|Classes|} \sum_{i=1}^{|Classes|} \max_{C_{cluster,\,j}} F(C_{class,\,i},\, C_{cluster,\,j})$$

The advantage of the F-measure is that it can also access the quality of hierarchical cluster structures. Let n be the number of all documents, then:[77]

$$F_{hierarchy} = \sum_{i} \frac{a_i + b_i}{n} \max\left\{F(C_{class,\,i},\, C_{cluster,\,j})\right\}$$

The maximum is taken over all clusters at all levels of the tree.

5.1.2 Cluster cohesiveness

The cluster cohesiveness is an internal quality measure. The cohesiveness of cluster C is defined as

$$Q = \frac{1}{|C|} \sum_{o \in C} \sum_{o' \in C} d(o, o')$$

In other words the cohesiveness measures how compact a partition is. The lower the cohesiveness the better is the quality of the cluster. The measure is useful to compare partitions created by different algorithms.

5.1.3 Subjective judgement

The F-measure requires the existence of relevance judgements. But the relevance judgements are usually assigned to a test set of documents by scientists that try to classify the documents as accurate as possible. But while the relevance of documents differs from user to user, there is no unique cluster structure that is optimal for every user at the same time. Therefore the relevance judgements are optimal for the people that assigned them to the documents but not for all other users. A measurement that overcomes this problem is a subjective judgement of the found cluster structure. Therefore for every cluster the 10 terms with the highest average weight are chosen to be representative for the cluster. Based on the terms it must be possible to identify the content of the cluster. In other words the 10 terms have to be considered consistent by many observers.

5.2 Cluster program

The clustering algorithms discussed in the previous chapter were implemented in Java. Fig. 2.3 shows the UML diagram of the class structure of the program. In the following the functionality of the classes are shortly described.

---

[77] See [SKK00]

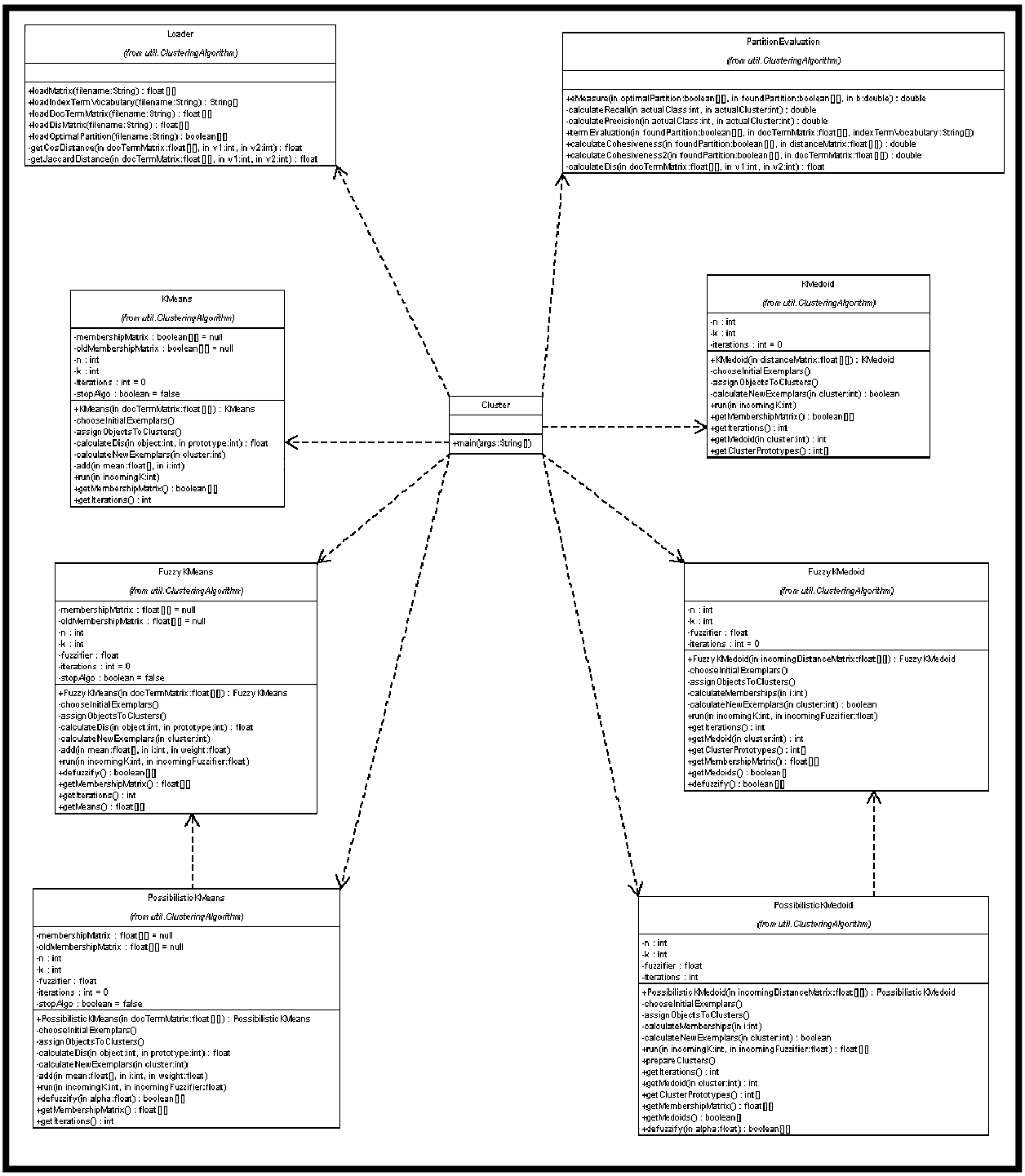

Figure 5.2: UML diagram of the Java classes

The class "Loader" is used to load files that are written in the vector format.[78] It provides functions that return a documents by terms matrix ("loadDisMatrix (filename, parameter)") or a documents by documents matrix ("loadDocTermMatrix (filename, parameter)"). In the latter case the matrix contains the dissimilarity values between all documents. The dissimilarity values are calculated by the cosine measure if the parameter is "1" and by the Jaccard coefficient if the parameter is "2". All returned matrices are standard java matrices that contain float values. The documents are

[78] See appendix …

represented by the matrix so that the first document in the loaded file has the position 0 in the matrix, the second document has the position 1, and so on. The first index term in the loaded file has the position 0 in the matrix, the second term has the position 1, and the on (Fig. 4.2). Additional the index term vocabulary can be loaded ("loadIndexTermVocabulary (filename)"). This is necessary for the subjective evaluation of the found partitions.

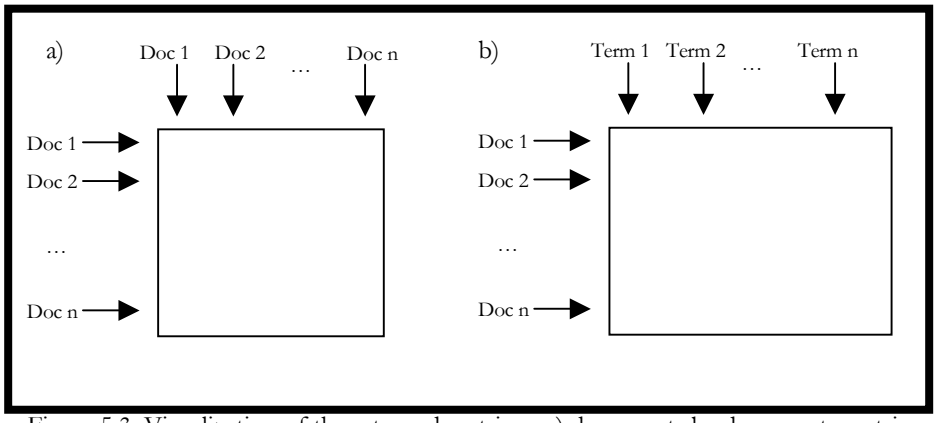

Figure 5.3: Visualization of the returned matrices. a) documents by documents matrix b) documents by term matrix

Every clustering algorithm is implemented as a class. The k-medoid variations require a documents by documents matrix for the constructor. The values have to be dissimilarity values. An input matrix that contains similarity values does not lead to correct results. Therefore it has to be transformed into a dissimilarity matrix before it can be used as input to the algorithms. The k-means variations require a documents by terms matrix for the constructor. All clustering algorithm can be started by calling the function "run(k, fuzzifier)". The crisp versions of the algorithms only need the parameter k, which is the required number of clusters. The fuzzy versions also need the parameter fuzzifier, which represents the value of the fuzzifier. Additional the fuzzy versions have a function "defuzzify()". It returns a crisp partition of the found cluster structure. In the case of probabilistic fuzzy clustering this function assigns the documents with full membership to the cluster in that they have the highest membership value and with zero membership to all other clusters. In the case of possibilistic fuzzy clustering the function requires an α-cut as input. Every document is assigned to the class or classes in that it has a membership value equal or above the α-cut.

The class "StatisticsLoader" is used to load data from the statistics files.[79]. The statistics file contains information about the number of classes used to classify the documents and the class assignments of the documents. The only purpose of the class is to load the optimal cluster structure, which was

[79] See appendix C for a description of the statistics file format

created by scientists, from the file so that it can be matched to the found cluster structure in the evaluation procedures. More specific, it returns a documents by classes matrix consisting of Boolean values.

The class PartitionEvaluation contains functions that access the quality of the found cluster structure. They are equal to the evaluation methods described in the previous subsection. The class requires the input partitions to be crisp. A crisp evaluation is applied because the Reuters documents that are used to test the clustering algorithms do not support the evaluation of fuzzy partitions. Therefore the found fuzzy cluster structures have to be defuzzified before they can be evaluated. The input to the different function is a documents by classes matrix. The matrix is a standard Java matrix that contains Boolean values. A value of "true" means that the corresponding document is member of the corresponding class.

The class "Cluster" is the main class that combines the other classes. The class can be activated from a shell.[80]

5.3 Experiments

The cluster program was used to make some experiments on the document collections described in chapter 3.

5.3.1 Goals

The experiments had two goals:

- Determine the quality of the fuzzy clustering algorithms, especially in relation to their crisp versions.
- Determine if the clustering results are consistent over different document collections.

5.3.2 Experimental structure

The specially prepared collections of documents were clustered (Documents400*.txt and Documents800*.txt). Every collection was clustered for k=2, 4, 8, 16, 32. Because the algorithms depend heavily on the initial chosen partition, for every collection 30 runs with randomly chosen initial partitions were performed and the best result was taken. This is appropriate because in real world applications one could perform the same strategy to cluster data. In this case the algorithms are run several times, every run is evaluated, and the best partition is returned to the user. The

[80] See appendix D for the manual of the program

number of runs that are performed is only constrained by the desired runtime of the algorithm. The fuzzifier value for all algorithms was set to two for all runs.

The final partitions were defuzzified. An evaluation of the fuzzy partition was not possible because the Reuters data does not contain enough information for this. For every algorithm the F-measure and the cohesiveness were calculated. The F-measure was taken to determine which run was the best of the 30 runs.

5.3.3 Observation

The tables 5.1 to 5.3 show the results of the experiment. Table 5.1 shows the results of the k-medoid and probabilistic fuzzy k-medoid algorithm. Table 5.2 shows the result of the possibilistic fuzzy k-medoid algorithm for different values of alpha. Table 5.3 show the equivalent results for the k-means variants.

Filename	k-Medoid		Probabilsitic Fuzzy k-Medoid	
Documents.txt	F-Measure	Cohesiveness	F-Measure	Cohesiveness
Without LSI	k=16: 0.67	0.70	k=16: 0.58	0.75
LSI, dim 40	k=16: 0.65	0.63	k=16: 0.58	0.67
LSI, dim 20	k=16: 0.65	0.48	k=16: 0.60	0.48
LSI, dim 10	k=16: 0.65	0.36	k=16: 0.60	0.33
Without LSI	k=16: 0.65	0.69	k=16: 0.55	0.72
LSI, dim 40	k=16: 0.66	0.69	k=16: 0.58	0.64
LSI, dim 20	k=16: 0.66	0.47	k=16: 0.56	0.51
LSI, dim 10	k=16: 0.65	0.24	k=16: 0.57	0.30

Table 5.1: Results of the k-medoid variations

Filename	Probabilistic k-Medoid $\alpha = 0.3$		Possibilistic Fuzzy k-Medoid $\alpha = 0.4$		Probabilistic Fuzzy k-Medoid $\alpha = 0.5$	
Documents.txt	F-Measure	Cohesiveness	F-Measure	Cohesiveness	F-Measure	Cohesiveness
Without LSI	k=32: 0.25	0.82	k=32: 0.24	0.85	k=8: 0.17	0.80
LSI, dim 40	k=16: 0.25	0.62	k=32: 0.25	0.42	k=32: 0.19	0.32
LSI, dim 20	k=8: 0.26	0.79	k=32: 0.37	0.41	k=8: 0.19	0.62
LSI, dim 10	k=32: 0.27	0.11	k=8: 0.23	0.35	k=4: 0.20	0.55
Without LSI	k=2: 0.18	0.91	k=8: 0.24	0.74	k=8: 0.17	0.51
LSI, dim 40	k=16: 0.24	0.93	k=16: 0.24	0.72	k=16: 0.19	0.67
LSI, dim 20	k=32: 0.28	0.70	k=8: 0.25	0.78	k=2: 0.19	0.75
LSI, dim 10	k=16: 0.34	0.63	k=8: 0.30	0.60	k=4: 0.27	0.54

Table 5.2: Results of the possibilistic k-medoid clustering for different a-values

Filename	k-Means		Probabilsitic Fuzzy k-Means		Possibilistic Fuzzy k-Means α = 0.4	
	F-Measure	Cohesiveness	F-Measure	Cohesiveness	F-Measure	Cohesiveness
Without LSI	k=16: 0.66	0.74	k=8: 0.20	0.86	k=2: 0.18	0.91
LSI, dim 40	k=16: 0.65	0.59	k=16: 0.28	0.63	k=2: 0.19	0.95
LSI, dim 20	k=16: 0.66	0.49	k=16: 0.25	0.15	k=2: 0.18	0.91
LSI, dim 10	k=16: 0.62	0.30	k=16: 0.33	0.37	k=2: 0.18	0.87
Without LSI	k=16: 0.67	0.75	k=16: 0.19	0.85	k=2: 0.18	0.97
LSI, dim 40	k=16: 0.70	0.62	k=8: 0.30	0.60	k=2: 0.19	0.95
LSI, dim 20	k=16: 0.65	0.46	k=8: 0.31	0.18	k=2: 0.18	0.92
LSI, dim 10	k=16: 0.61	0.28	k=8: 0.27	0.33	k=2: 0.18	0.88

Table 5.3: Results of the k-means variations

5.3.4 Discussion of the observation:

The tables show some interesting results:

- In most cases the cohesiveness is lower the higher the impact of the latent semantic indexing was in the indexing process. This means that latent semantic indexing leads to more compact clusters.
- The latent semantic indexing does not have an impact on the F-measure. In most cases the F-measure of a certain algorithm is relatively equal over a particular document collection, independent of if latent semantic indexing was applied to prepare the input data or not.
- While k-medoid and k-means perform similar on the different collections, there is a difference in the performance between the fuzzy versions. The fuzzy k-medoid algorithms perform better than the fuzzy k-means algorithms.
- The fuzzy algorithms do not perform better in the F-measure compared to their crisp versions. Moreover the fuzzy algorithms have little less F-measure values than the crisp versions on the same collection. This does not support the thesis discussed in the previous chapters that fuzzy logic would better fit to document clustering tasks than crisp logic. One reason could be that the collection were prepared artificially. Recall that the collection files consist documents from ten classes and that the documents are uniformly distributed over the ten classes. Because this leads to ten very distinguished classes this may support non-fuzzy clustering algorithms.

Chapter 6

SUMMARY AND OUTLOOK

This work examined the usefulness of fuzzy clustering algorithms in the context of information retrieval systems. Therefore in chapter two the work described performance measures of IR systems and why a classification schema may improve the performance. It described some common methods how to classify documents into an existing classification schema. At the end the chapter mentioned the problem that a classification schema may not be available every time and that clustering may overcome this problem.

In chapter three the input and output of the document clustering process was formalized. It highlighted the difficulties of text data and its formalization. The chapter introduced the vector-space model and presented some common methods to index documents so that they can be represented in the model. Such a document representation allows to define a dissimilarity measure on the collection. The cosine measure proved to be an accurate dissimilarity measure. At the end the chapter discussed formally why fuzzy logic may improve the clustering process of documents.

Chapter four continued to described the k-prototype algorithms and their fuzzy versions. The basic ideas of the algorithms as well as the algorithms itself were described and formally presented.

Chapter five described the experiments that were made with the algorithms and their results. The algorithms performed good on the test collections. However the fuzzy algorithms could not prove to be superior over their crisp versions. But for the case that a user desires non-exclusive clusters the possibilistic fuzzy k-means or the possibilistic fuzzy k-medoid may be necessary because the other algorithms are not able to create non-exclusive clusters. Additional research is needed to check if the results can be repeated on many other document collections. It is also necessary to test the clustering algorithms in running IR systems over a longer time period. This would allow to collect relevance measures of the retrieved documents which in turn can be compared to relevance measures of IR systems that do not retrieve from clustered data or from data that was clustered using other clustering algorithms. Another shortcoming is that most test collections are prepared for classical cluster algorithms. In the experiments it was necessary to deffuzify the fuzzy-partitions prior evaluation. But the deffuzification eliminates important information of the classification that would be necessary for a superior quality of the classification schema. Therefore the creation of

document collections is necessary that support the evaluation of fuzzy-partitions. This would lead to more accurate performance measures of fuzzy-clustering algorithms that may provide better support for their usefulness.

APPENDIX

A Fuzzy Logic

Fuzzy logic was developed by L.A. Zadeh in the year 1965.[81] One premise of fuzzy logic is the contradiction of high complexity and high precision. Fuzzy logic is a theory that is meant to cope with this contradiction. This opens possibilities to model systems that could not be captured by classical set theory because some parts or parameters of the systems are not precisely defined. Take, for example, an automatic heating system. The system is considered to maintain an accurate temperature in a room. Therefore the system has to turn on the heater if the temperature is too cold and turn off the heater if the temperature is accurate. But the terms "too cold" and "accurate" are not precisely defined. Moreover they are more or less true for a wide range of temperature values. Fuzzy logic allows to model fuzzy terms like "too cold" and "accurate". In the case of clustering, fuzzy logic can be used to model the notion of "membership". In the following the basics of fuzzy logic are described.

In mathematics a set is a collection of unique objects, also called elements. Therefore an element does either belong to a set or not. Let X be a universe and x be an element from X. The set A can be described by a *characteristic function*:

$$u_A : X \to \{0,1\},\ \mathrm{u}_A(x) = \begin{cases} 1, & \text{for x} \in \text{A} \\ 0, & \text{else} \end{cases}$$

Fuzzy logic is a generalization of set theory. The basic idea is that the membership of an element in a set should not be complete but fuzzy. This means an element is only to a certain degree a member of a particular set. Therefore the basic element of fuzzy logic is the fuzzy set.

Definition (Fuzzy set)

Let X be a universe. A *fuzzy set* is a set $A = \{(x, u_A(x)) \mid x \in X\}$. $u_A(x)$ is called *membership function* and is defined as:

$$\begin{aligned} u_A : \quad & X \to [0,1] \\ & x \to u_A(x) \end{aligned}$$

■

[81] See [Kle00]

To cope with the imprecision of natural language terms fuzzy logic provides the concept of the linguistic variable. The values of a linguistic variable are terms of a natural language. They are represented by a fuzzy set. The fuzzy set maps the linguistic value scale on a numeric value scale. For example, the term "member of cluster 1" is a linguistic variable. The linguistic value scale contains terms like "not member of cluster 1", "more or less member of cluster 1", "definitely member of cluster 1". This values can be mapped on the numerical value scale [0,1], whereas 0 represents "not member of cluster 1", 1 represents "definitely member of cluster 1", and the other linguistic values are mapped somewhere between these extremes.

A fuzzy set can be transformed into a crisp set. This process is called *defuzzzification*. The following two definition provide tools for defuzzification.

Definition (Carrier T)

Let A be a fuzzy set. A crisp set T(A) is called carrier if

$$\mathrm{T(A)} = \{x \mid x \in X, u_A(x) > 0\}$$ ■

Definition (α-cut)

Let A be a fuzzy set. A crisp set A_α is called α-cut if

$$\mathrm{A}_\alpha = \{x \mid x \in X, u_A(x) \geq \alpha\}$$ ■

B Vector Format

The vector format was developed by the author of this text. The goal was to create an interface between the indexer program and the clustering algorithms described in this text. The format described how document vectors are represented in a file. Therefore a file in vector format starts with the line "%Vector Format". In the following the format consists of three blocks.

The first block starts with the number of documents described by the format. It is followed by a line that contains the number of terms.

The next block consists of one line and contains all indexing terms. Each index term is separated by a space. The index terms are ordered so that the first index term corresponds to the first component of the document vector, the second index term corresponds to the second component of the document vector, and so on.

The last block contains the document vectors. Therefore the first line of the block represents the document vector of document one, the second line represents the document vector of document two, and so on. Every line contains the weights of the corresponding document vector. The single weights are separated by an empty space. The first weight represents the value of the first component, the second weight represents the value of the second component, and so on.

The format ends with the line "EOF". Formally the format is as follows:

```
%Vector Format
<n = number of documents>
<p = number of terms>
<index term1> <index term 2> … <index term p>
<component 1> <component 2> … <component p>
…
<component 1> <component 2> … <component p>
EOF
```

C Format of the statistics file

The statistic files were created to contain information about the optimal class assignments of a document collection. The format is completely written in text. The first line contains the words "Amount topics: " followed by the number of topics in the collection. In the next line the word "Topics:" is written. In the following lines the topics are written. Therefore every line corresponds to exactly one topic. Such a line starts with the name of the topic, followed by the number of documents that are assigned to the topic. After that the id of the documents follow. It is assumed that the id's of the documents are 0, 1, 2, …, n-1, whereas the first document of the collection has the id 0, the second document has the id 1, and so on. Within every line the documents are ordered by their id in ascending order. The format models also documents that are assigned to more than one topic. Formally the format is as follows:

```
Amount topics: <p=number of topics>
Topics:
<name of topic 1> <number of documents> <id1> <id2> … <idt_1>
…
<name of topic p> <number of documents> <id1> <id2> … <idt_p>
```

D Manual of the indexer program

The indexer program is a single class programmed in Java. Therefore the Java Virtual Machine has to be installed on the computer to run the program. The input of the program is a text file containing the documents and the output is a text file in the vector format that contains the document vectors.

The path of the program has to be set in the Java class path. The file with the stop word list has to be in the same directory. The stop word list file has to be text file. Every line of the file contains exactly one stop word. Additional there has to be a symbol file in the directory. The symbol file is also a text file and contains symbols like "?", ".", "-". Corresponding symbols in the text are ignored by the indexer program and therefore not used as index terms. Like in the stop word list each line of the symbol file contains exactly one symbol. From a shell of the operating system the program is started by

```
java TextAnalyzer –k  Sourcefile Destinationfile
```

- *-k :* is followed by number that is greater or equal to zero. The number is the parameter for the latent semantic indexing procedure. It determines how many singular values are used for the procedure. E.g. –k100 means that the first 100 singular values are used in the LSI process. –k0 means that no LSI is performed.

- *Sourcefile:* The path and the name of the source file that contains the text to be indexed. The text has to be in the Reuters format described in chapter 3.2.4.1 of this work.

- *Destinationfile:* The path and the name of the file, where the created document vectors have to be saved in. The file will be saved in the vector format.[82]

E Manual of the cluster program

The cluster program is programmed in Java. Therefore the Java Virtual Machine has to be installed on the computer to run the program. The input of the program is a text file in the vector format that contains the document vectors.[83] The path of the program has to be set in the Java class path. The program can be started from a shell by:

```
java cluster –type –k –m -Sourcefile -a
```

- -type: Decides which algorithm is run:
    - 1: k-medoid
    - 2: probabilistic k-medoid
    - 3: possibilistic k-medoid
    - 4: k-means

[82] See appendix B

[83] See appendix B for a description of the vector format

  - 5: probabilistic k-means
  - 6: possibilistic k-means
- -k: the number of clusters
- -m: the fuzzifier value
- -a: the α -cut for defuzzification. This value is only needed for the possibilistic versions.
- -Sourcefile: The file that contains the document vectors. The file has to be in vector format.

Take for example the following call of the program:

```
java cluster –type2 –k5 –m2 –c:\Documents.txt
```

The program clusters the document from Documents.txt with the probabilistic fuzzy k-medoid. The desired number of clusters is 5 and the fuzzifier is two.

After the program finished, the fuzzy-partition is defuzzified. For the crisp decision the F-measure and the cohesiveness is calculated and shown to the user.

F Stop-Word list

a
about
above
according
across
after
against
albeit
all
almost
alone
along
already
also
although
always
among
amongst
an
and
another
any
anybody
anyhow
anyone
anything
anyway
anywhere
apart
are
around
as
at
author
av
available
be
became
because
become
becomes
becoming
been
before
beforehand
behind
being
below
beside
besides
between
beyond
both
but
by
can
cannot
canst
certain
cf
cfrd
choose
conducted
considered
contrariwise
cos
could
crd
ct
cuday
described
describes
designed
determine
determined
different
discussed
do
does
doesn't
doing
dost
doth
double
down
dual
due
during
each
either
else
elsewhere
enough
et
etc
even
ever
every
everybody
everyone
everything
everywhere
except
excepted
excepting
exception
exclude
excluding
exclusive
far
farther
farthest
few
ff
first
for
formerly
forth
forward
found
from
front
further
furthermore
furthest
generalgiven
get
go
had
halves
hardly
has
hast
hath
have
he
hence
henceforth
her
here
hereabouts
hereafter
hereby
herein
hereto
hereupon
hers
herself
him
himself
hindmost
his
hither
hitherto
how
however
howsoever
I
ie
if
in
inasmuch
inc
include
included
including
indeed
indoors
inside
insomuch
instead
into
investigated
inward
inwards
is
it
its
itself
just
kind
kg
km
last
latter
latterly

less
lest
let
like
little
ltd
made
many
may
maybe
me
meantime
meanwhile
might
more
moreover
most
mostly
more
mr
mrs
ms
much
must
my
my
self
namely
need
neither
never
nevertheless
next
no
nobody
none
nonetheless
noone
nope
nor
not
nothing
notwithstanding
now
nowadays
nowhere
obtained
of
off
often
ok
on
once
one
only
onto
or
other
others
otherwise
ought
our
ours
ourselves
out
outside
over
own
per
performance
performed
perhaps
plenty
possible
present
presented
presents
provide
provided
provides
quite
rather
really
related
report
required
results
round
said
sake
same
sang
save
saw
see
seeing
seem
seemed
seeming
seems
seen
seldom
selected
selves
sent
several
sfrd
shalt
she
should
shown
sideways
significant
since
slept
slew
slung
slunk
smote
so
some
somebody
somehow
someone
something
sometime
sometimes
somewhat
somewhere
spake
spat
spoke
spoken
sprang
sprung
srd
stave
staves
still
studies
such
supposing
tested
than
that
the
thee
their
them
themselves
then
thence
thenceforth
there
thereabout
thereabouts
thereafter
thereby
therefore
therein
thereof
thereon
thereto
thereupon
these
they
this
those
thou
though
thrice
through
throughout
thru
thus
thy
thyself
till
to
together
too
toward
types
towards
unable
under
underneath
unless
unlike

```
until
up
upon
upward
upwards
us
use
used
using
various
very
via
vs
want
was
we
week
well
were
what
what
ever
what
soever
when
whence
whenever
whensoever
where
whereabouts
whereafter
whereas
whereat
whereby
wherefore
wherefrom
wherein
whereinto
whereof
whereon
wheresoever
whereto
whereunto
whereupon
wherever
wherewith
whether
whew
which
whichever
whichsoevr
while
whilst
whither
who
whoa
whoever
whole
whom
whomever
whom
soever
whose
whosoever
why
will
wilt
with
within
without
worse
worst
would
wow
ye
yet
year
yippee
you
your
yours
yourself
yourselves
```

G Symbol list

```
""
" "
-
<
>
&lt;
&#3;
&#2;
..
--
?
.
:
,
!
'
(
)
/
\
+
%
"
""
<<
```

REFERENCES

[Bez81] Bezdek, J. C.; „Pattern Recognition with Fuzzy Objective Function Algorithms“, Plenum Press, New York, 1981

[BCM96] Barni, M.; Capellini, V.; Mecocci, A.; “Comments on ‘A Possibilistic Approach to Clustering’”; IEEE Trans. Fuzzy Syst., vol. 4, Aug. 1996pp. 393-396

[C01] Chitkara, V.; “Color-Based Image Retrieval Using Compact Binary Signatures”; University of Alberta, Technical Report TR 01-08, May 2001

[C86] Chomsky, N.; “Knowledge of Language. Its Nature, Origin, and Use.”; Greenwood Press, Praeger Publishers, New York, 1986

[CSBB97] Chang, S.; Smith, J.; Beigi, M.; Benitez, A.; “Visual Information Retrieval from Large Distributed On-line Repositories”; Columbia University, NY, 1997

[DDH90] Deerwester, S.; Dumais, S.; Harshman, R.; „Indexing by Latent Semantic Analysis“;

[DK82] Devijer, P.; Kittler, J.; „Pattern Recognition: A Statistical Approach”; Prentice Hall, London, 1982

[DP96] Domingos, P.; Pazzani, M.; “Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier”; Machine Learnin: Proccedings of the 13th International Conference, Morgan Kaufman, pp. 105-112

[FR95] Forrester Research; “Coping with complex data”; The Forrester Report, April 1995

[FB92] Frakes, W. B.; Baeza-Yates, R.; „Information Retrieval – Data Structures & Algorithms“, Prentice Hall, New Jersey, 1992

[G98] Gavrila, D. ; “The Visual Analysis of Human Movement: A Survey”; Daimler Benz Research, 1998

[Joa98] Joachims, T.; “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”; Machine Learning, ECML-98, Tenth European

Conference on Machine Learning, pp. 137-142

[Kar00] Karlgren, J.; "The Basics of Information Retrieval: Statistics and Linguistics"; Swedish Institute of Computer Science, Human Machine Interaction and Language Engineering Laboratory, 2000

[KMG85] Keller, J.; Gray, M.; Givens, J.; „A Fuzzy K-Nearest Neighbor Algorithm"; IEEE Trans. Syst., Man, Cybern., vol.SMC-15, No. 4, July/August 1985, pp. 580-585

[KJY99] Krishnapuram, R.; Annupam, J.; Yi, L.; "A Fuzzy Relative of the k-Medoids Algorithm with Application to Web Document and Snippet Clustering"; Proceedings of IEEE Intl. Conference Fuzzy Systems – FUZZIEEE 99, Korea, 1999

[KK93] Krishnapuram, R.; Keller, M.; „A Possibilistic Approach to Clustering"; IEEE Transactions on Fuzzy Systems, Vol. 1, No. 2, May 1993, pp.98-110

[KK96] Krishnapuram, R.; Keller, M.; „A Possibilistic C-Means Algorithm: Insights and Recommendations"; IEEE Transactions on Fuzzy Systems, vol. 4, No. 3, August 96, pp. 385-393

[LP89] Lee Pao, M.; "Concepts of Information Retrieval"; Libraries Unlimited, Inc., Englewood, 1989

[Luh58] Luhn, H.P.; "The Automatic Creation of Literature Abstracts", IBM Journal of Research and Development, Vol. 2, No. 2, 1958, pp. 159-165

[MF98] Michalewicz, Z.; Fogel, D.B.; "How to Solve It: Modern Heuristics"; Springer-Verlag, Berlin, 1998

[Mic92] Michalski, R. S.; "Concept Learning" in Encyclopedia of Artificial Intelligence" 2nd edition, Volume 1, A-L, pp. 249-259

[NMTM98] Nigam, K.; McCallum, A., Thrun, S.; Mitchell, T.; "Learning to Classify Text from Labeled and Unlabeled Documents"; 15th National Conference on Artificial Intelligence (AAAI-98), 1998

[Por80] Porter, M. F.; "An algorithm for suffix stripping"; Program, 14, pp. 130-137, 1980

[Rij79] van Rijsbergen, C. J.; „Information Retrieval", 2nd edition, Butterworths, London, 1979

[Sal00] Salton, G.; "Information Retrieval" in Encyclopedia of Computer Science 4th edition, Nature Publishing Group, London, 2000, pp. 858-863

[SEK00] Steinbach, M.; Ertöz, L.; Kumar, V.; „A new shared nearest neighbor clustering algorithm and its applications"; University of Minnesota, 2000

[SKK00] Steinbach, M.; Karypis, G.; Kumar, V.; „A Comparison of Document Clustering Techniques"; KDD Workshop on Text Mining, 2000

[SZR99] Srihari, R.; Zang, Z.; Rao A.; „Intelligent Indexing and Semantic Retrieval of Multimodal Documents"; State University of New York at Buffalo, November 1999

[SM87] Salton, G., McGill, M.; "Information Retrieval - Grundlagen für Informationswissenschaftler"; McGraw Hill, Birkenau, 1987

[SN99] Stein, B.; Niggemann, O.; „On the Nature of Structure and its Identification"; Dept. of Mathematics and Computer Science, D-33095 Paderborn, Germany

[Spa72] Sparck Jones, K.; "A statistical interpretation of term specificity and its application in retrieval", Journal of Documentation, 28: 11-20

[The89] Therrien, C.; "Decision Estimation and Classification"; John Wiley & Sons, 1989

[YL99] Yang, Y.; Liu, X.; „A re-examination of text categorization methods"; SIGIR-99, 1999

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____ Paderborn, 30. Juli 2002