Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Evelina Gabašová

Text Clustering and Classification (Klastrování a klasifikace textů)

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Marta Vomlelová, Ph.D.
Studijní program: Informatika, obecná informatika

2007

Ráda bych poděkovala Mgr. Martě Vomlelové, Ph.D. za cenné připomínky a odborné vedení této bakalářské práce.

Contents

Title: Text Clustering and Classification
Author: Evelina Gabašová
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Marta Vomlelová, Ph.D.
Supervisor's e-mail address: Marta.Vomlelova@mff.cuni.cz

Abstract: Text clustering and classification are important machine learning tasks. In this work, a combination of their approaches is presented. The main purpose was to automatically prepare a set of clusters (or generally concepts), which would subsequently serve as a training data for learning of a classifier. This work comprises of theoretical background, implementation details and experimental results of clustering and classification of text documents. A train set of documents is first hierarchically clustered by the bisecting k-means algorithm. The result is offered to an expert for modifications and possible improvements of the hierarchy. Following this, the resulting structure is used for learning of a naive Bayes classifier and a test set of documents is classified by it. A program was developed to perform these tasks and its results are evaluated and compared in processing document collections written in both English and Czech.

Keywords: clustering, classification, machine learning, naive Bayes classifier, k-means algorithm

Název práce: Klastrování a klasifikace textů
Autor: Evelina Gabašová
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: Mgr. Marta Vomlelová, Ph.D.
e-mail vedoucího: Marta.Vomlelova@mff.cuni.cz

Abstrakt: Klastrování a klasifikace textů jsou důležitými úlohami strojového učení. V této práci je prezentována kombinace jejich přístupů. Hlavním účelem bylo automaticky připravit množinu klastrů (nebo obecně konceptů), které by následně sloužily jako trénovací data pro naučení klasifikátoru. Tato práce zahrnuje teoretické pozadí, detaily implementace a výsledky experimentů pro klastrování a klasifikaci textových dokumentů. Trénovací soubor dokumentů je nejprve hierarchicky klastrování algoritmem bisecting k-means. Výsledek tohoto procesu je možné upravovat a vylepšovat s využitím expertní znalosti. Tímto způsobem vytvořená hierarchická struktura je použita pro naučení naivního bayesovského klasifikátoru, který je následně využit k roztřídění testovací množiny dokumentů. Pro tyto účely byl vyvinut program, jehož výsledky jsou zhodnoceny a porovnány při zpracování českých a anglických dokumentů.

Klíčová slova: klastrování, klasifikace, strojové učení, naivní bayesovský klasifikátor, algoritmus k-means

Chapter 1

Introduction

Classification of text documents is becoming a very important task as the amount of documents, e.g. on the internet, rises. To be able to manipulate large collections of documents, their organization to some predefined categories (classes) according to their content is very practical. It helps to effective resource management.

Manual classification of documents is very time-consuming. Therefore, demand for automated tools for this task is increasing. The automatic classification algorithms work well, however they have to be learned on a collection of documents, which have already been classified to some predefined classes. After the classifier is trained on such documents, it is able to automatically categorize new documents.

The main problem of classification is the described necesity to have an already classified set of documents. For learning of a successful classifier, this collection should be large enough to contain as much as possible cases, which can occur. Indexing of documents by an expert is again very demanding and time-consuming.

An alternative to learning a classifier on some previously indexed documents is clustering. This process is unsupervised and does not require any external data. It works only with the information present in the texts themselves. The documents are grouped into subsets according to similarity of their content. However, this method does not usually reach the accuracy of classification. Word usage, choice of expressions and form of documents can greatly vary, even if the documents share the same topic.

In this work we present an approach, which combines both classification and clustering. It first uses clustering for creating a hierarchy of clusters. These clusters can be modified with use of expert knowledge to reduce the number of documents assigned to an improper group. Following this, the clustered documents become a training collection for a classifier. The classifier is learned on the clusters and can be afterwards used for categorization of new documents.

For clustering, a hierarchical modification of the k-means algorithm, the bisecting k-means, was implemented. To perform classification, a naive Bayes classifier was used.

This work comprises of theoretical background of clustering and classification, implementation of a program, which allows to execute the described tasks, and evaluation of its results.

In the chapter 2, conversion of documents to a form suitable for clustering and classification is described. In the chapters 3 and 4 an overview of algorithms used for clustering and classification is given. In the chapter 5 we describe, how the whole task was implemented. Following this, the chapter 6 shows methods of measuring clustering and classification quality and how outputs of the program were evaluated. The last chapter 7 contains conclusions.

The first appendix A describes the document collections, which were used for testing the program. The appendix B contains a user documentation of the program in a form of a sample walk through the individual clustering and classification tasks.

Chapter 2

Preprocessing

Clustering and classification methods operate with a space of document vectors, which is prepared in the preprocessing stage. In this vector-space model, each document is represented by a vector of its features, which are constructed from the words of the document.

There is a large amount of various methods for creating the document vectors. The following provides just a brief description of the techniques, that were used in this work.

2.1 Feature extraction

As the initial step, each document is transformed to a vector of words, which appear in the given document. Together these vectors form a word-by-document matrix

$$A = (w_{ij}),$$

where the elements w_{ij} represent a weight of a word j in the i-th document. Number of various words that appear in a collection of documents is usually huge, therefore the document vectors form a vector space of a very large dimension. To be able to perform clustering and classification, the number of dimensions should be reduced in attempt to extract features, that characterise each document well and are capable of distinguishing documents from each other.

A common preprocessing steps in working with mostly English documents are removal of stop words and stemming. The so-called stop words are mostly functional words and do not have much relation to a document's content. It is eligible not to process them. The stemming works by reducing a word to its base or root, because words with common root share mostly the same meaning. These words are therefore considered to represent a single word in its basic form. This reduces the dimension of the feature vector space and lowers the amount of noise in the data.

Nevertheless, these methods were not used in this work. One of purposes of this work was to use the clustering and classification techniques for processing of documents written in Czech. In case of the Czech language, stop words removal and stemming are much more complex tasks than in English. They represent a separate comprehensive problem. Therefore we chose not to perform them and concentrate on the main clustering and classification tasks.

The following sections show, how weights of words (features) in individual documents are extracted from the document collection. Possible further processing of the features is described in the section 2.2.

2.1.1 Feature thresholding

Features that appear very frequently and are included in most documents do not help to distinguish between documents and bear just a little information about the individual document's content. Such features are usually general or auxiliary words. Also features that appear very rarely through the whole collection of documents are not helpful. They do not indicate any similarities between documents.

For these cases, the method of feature thresholding [1] is used. An upper and a lower threshold for the feature frequency is set. Afterwards, all features, which appear in more documents in the collection than the upper threshold, or in less documents than the lower theshold, are not considered.

2.1.2 Feature weighting

As a next step, the raw frequencies (i.e. counts) of words are converted to feature weights. There are numerous weighting methods. The simplest technique assigns Boolean vallues to the document features

$$f_i = \begin{cases} 0 & f_i \text{ does not occure in the document} \\ 1 & f_i \text{ occurs in the document} \end{cases}$$

Another simple method is to use directly relative frequencies of features in the document. There are also many more sophisticated techniques, whose main attempt is to take into account other factors, such as the lenght of the document, frequency of a given word in the whole collection of documents etc.

The weighting scheme, which was used for this work, is the entropy weighting as described in [1]. This method comes from the information theory and considers also the distribution of words in the whole document collection. The weight of a word i in a document j is computed as

$$w_{ij} = \log(f_{ij}+1) \cdot \left(1 + \frac{1}{\log(n)} \sum_{k=1}^{n} \left[\frac{f_{ik}}{n_i} \log\left(\frac{f_{ik}}{n_i}\right)\right]\right),$$

where f_{ij} is the frequency of the word i in the document j, n is the number of documents and n_i is the number of occurences of word i in the whole document collection.

2.2 Further feature processing

As mentioned, the vector space formed by the document vectors has a very large dimension. In order to process these data effectively, the dimensionality is usually reduced. Also to prevent overfitting of the model, some generalization provided by the dimensionality reduction is appropriate.

Again, many techniques exist for this problem. A form of dimensionality reduction is the feature thresholding described earlier. A more elaborate method is the latent semantic analysis (indexing) [3].

2.2.1 Latent semantic analysis

Latent semantic analysis (LSA) is an algebraic method for reducing the dimensionality. The main purpose of this technique is to replace individual features with fewer more general concepts.

The aim is to suppress effects of polysemy (a single word with multiple different meanings) and synonymy (a single meaning expressed by a number of different words) to expose the "latent semantic" structure of the text. This latent semantic structure attempts to capture the underlying patterns in word usage throughout the document collection, which is hidden because of different choice of expressions in various documents.

The mathematical background of LSA comes from the singular value decomposition (SVD). Let A be a word-by-document matrix of size $m \times n$, where m denotes number of words and n number of documents, $m \geq n$. The rank of A is r, $r \leq m, n$. The singular value decomposition of A is defined as

$$A = U \Sigma V^T.$$

The matrix Σ is a diagonal matrix of size $r \times r$, where its diagonal elements $\sigma_1, \sigma_2, \dots, \sigma_n$ are the so-called singular values. The matrices U and V are orthogonal and contain the left and right singular vectors, which correspond to the singular values. U is of size $m \times r$ and V $r \times n$.

Let the elements of Σ be ordered by size from the largest to the smallest (together with the corresponding singular vectors) and $\sigma_1, \sigma_2, \dots, \sigma_k$ be the k largest singular values. The matrix

$$A_k = \sum_{i=1}^{k} u_i \sigma_i v_i^T, i.e. A_k = U_k \Sigma_k V_k$$

is the best approximation of A of rank k. Therefore, the decomposition reduces the dimensionality of the original problem. Moreover, the matrix A_k as an approximation reduces the noise and ambiguity of the original data while preserving the structure. Words with similar patterns of usage are considered close or similar in the k-dimensional space. Similar documents may be close in the approximate model while not sharing any terminology. Also the relations between documents and features (words) are retained.

The SVD also offers a reduction of the computational demand. To compare two documents, cosine between two respective columns of A_k or cosine between two rows of $V_k\Sigma_k$ can be used. The cosine value indicates the degree of similarity between the contents of the two documents. Similarly, two words are compared by the cosine between two columns of A_k or two rows of $U_k\Sigma_k$. Finally, to get the measure of relevance of a feature (word) i to a document j, cosine between the i-th row of the matrix $U_k\Sigma_k^{1/2}$ and $V_k\Sigma_k^{1/2}$ is used.

To incorporate new documents into the computed LSA model or to compare new documents with the already created document vectors, the ideal way is to recompute the whole decomposition with added documents. However, less computationally demanding methods can be used. To update the model or to compare a new document, the added document is *folded-in* the existing SVD [3]. The *folding-in* is performed by

$$\hat{d} = d^T U_k \Sigma_k^{-1},$$

where d is the vector of the new document. The created vector $\hat{d}$ can be either appended as a new row of the V_k matrix, or compared with other processed documents by the already described method.

Folding-in is a simple method of updating the LSA model. However, it does not consider any possible new features in the added document, which were not present in the original document collection. It is therefore only a limited method for updating the model. Other methods of updating are described in [3].

Chapter 3

Clustering

Clustering is a process of partitioning the collection of documents (in general points) to clusters - subsets of points which are considered similar according to some criterion. It is a method of unsupervised learning.

Clustering algorithms use two main approaches to the clustering process, hierarchical and divisional. The next section gives an overview of both approaches.

3.1 Clustering algorithms

Hierarchical and partitional methods produce different results. Partitional algorithms create a division of the original set to a given number of clusters. These clusters are determined all at once. On the contrary, the hierarchical algorithms build the clusters in a sequential order by processing the previously created clusters and thus producing a hierarchy of clusters. The result of hierarchical clustering can be graphically demonstrated as a tree of clusters, so-called dendrogram (see the figure 3.1).

3.1.1 Hierarchical clustering

Hierarchical clustering produces a tree of nested clusters. There are two approaches of building the hierarchy, agglomerative (bottom-up) and divisive (top-down).

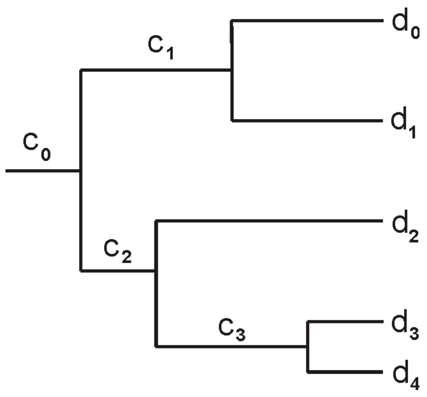


Figure 3.1: An example of a dendrogram; $\mathbf{c_0}, \ldots, \mathbf{c_3}$ are clusters, $\mathbf{d_0}, \ldots, \mathbf{d_4}$ are clustered documents.

The agglomerative method begins with each point as a single cluster and repeatedly compares the clusters with each other and merges two (or generally k) points, which are most similar according to a chosen criterion (see the section 3.2). The process ends with one cluster that contains all the points. For this algorithm, important is the choice of the clusters to merge.

On the contrary, the divisive approach starts with one cluster, which contains all the points. Afterwards, the cluster is recursively split to two (or generally k) subclusters until each point represents a single cluster. Again, crutial is the way to divide the cluster and the choice of the cluster to divide.

There are many methods to choose, which cluster to split next. The simplest one is to choose the cluster with the largest cardinality (number of points) at the moment. However, when in the data there are clusters of unbalanced size, the larger clusters are split earlier. Another method is to choose the cluster with the largest inner variance.

The tree built by these methods offers a more in-depth view of the data with each level of the hierarchy. The main disadvantage of this approach comes from the need to compare all the clusters with each other during the clustering process to decide, which clusters to merge/split. Consequence is its quadratic complexity and therefore larger computational demand.

3.1.2 Partitional clustering

The partitional clustering techniques attempt to detect k clusters at the same time. They do not create any hierarchical structure of clusters, the result is a flat division of elements into subsets. An important member of this group of algorithms is the k-means algorithm, which will be discussed in more detail.

For the k-means algorithm, each cluster has its representant, a centroid. The centroid indicates the center of the cluster and is taken as the mean (or weighted average, median etc.) of the points assigned to the given cluster. In the process of clustering, the points are assigned to the centroid, which lies nearest according to some distance measure (see the section 3.2).

The basic version of the k-means algorithm can be described as follows:

1. Choose the number of clusters k.

2. Randomly initialize the initial k centroids.

3. Assign points to their nearest centroid.

4. Recompute the centroids.

5. Repeat the steps 3. to 4. until the distribution of points between clusters does not change or some other criterion is fulfilled.

This algorithm is straightforward and efficient (linear to the number of points). However, it has also some disadvantages. In the first place, there is no clear choice of the number of clusters k. The results can be misleading when an inappropriate k is chosen. Furthermore, the clustering depends greatly on the choice of the initial centroids. One of the solutions of this problem is to repeat this process a couple of times and then choose the best result measured by some chosen criterion function (see the section 3.2). However, the algorithm is not guaranteed to find the global minimum of the criterion function, it finds only a local minimum.

3.1.3 Bisecting k-means algorithm

Both clustering approaches can be combined to get a hierarchy of clusters and at the same time reduce the computational complexity. The bisecting k-means algorithm as described by [8] and [9] is one of these methods. The algorithm performs repetitive iterations of the k-means algorithm and thus creates a nested hierarchical structure of clusters.

This method is based on the hierarchical divisive approach but the split is performed by the k-means algorithm with $k = 2$, i. e. the cluster is bisected.

The algorithm builds the clusters in the following way:

1. Start with all points in the same cluster.
2. Choose a cluster to split.
3. Use k-means algorithm with $k = 2$ to bisect the cluster.
4. Repeat steps 2. to 3. until each cluster contains only a single point (or some other criterion is satisfied).

As an improvement, the bisecting step can be repeated several times with different choice of the initial centroids to choose the best result. The computational complexity remains linear in the number of points.

3.2 Similarity and distance measures

The results of described clustering algorithms depend mainly on the chosen measures of distance between clusters and similarity of points in a cluster. Measures of distance are used to determine the degree of similarity of two clusters and according to this to find the division into subclusters, i.e. to assign a cluster to the nearest centroid in the case of the k-means algorithm. The measures of similarity are on the other hand used to measure the quality of the clustering, i.e. to find the best result between some repeated clusterings. The clustering algorithms attempt to find the optimum of the quality function.

Some of the common distance measures include the Euclidean distance (2-norm distance) and unquestionably the cosine measure. This measure computes the cosine of the angle between two vectors in the n-dimensional space. It is defined as

$$\cos(d_i, d_j) = \frac{d_i \times d_j}{\|d_i\| \|d_j\|},$$

where d_i and d_j indicate two document vectors. The value of this measure does not depend on the lenght of the vectors. The result is equal to 1 for identical vectors and to 0 for orthogonal vectors. For similar vectors, the value comes closer to 1, for diverse vectors it is nearer to 0. The cosine measure is widely used especially for the document vectors comparison.

The clustering similarity measures are used to find the optimal clustering result. Again, there are numerous functions to measure this property. As shown by [9], for the bisecting k-means algorithm, the best results were achieved by maximizing the sum of cosine distances between the cluster centroid and documents assigned to the cluster, summed over all clusters:

$$maximize \sum_{r=1}^{k} \sum_{d_i \in C_r} \cos(d_i, Centr_r),$$

where k is the number of the clusters, d_i are the document vectors, C_r represent the created clusters and $Centr_r$ is the centroid of the cluster C_r. This can also be rewrited as

$$maximize \sum_{r=1}^{k} \| \sum_{d_i \in C_r} d_i \|$$

if the document vectors are normalized to lenghth 1. The distance of each document in the cluster to the corresponding centroid is minimized over all the clusters.

Chapter 4

Classification

Document classification (categorization) is a task of assigning documents to some predefined classes. Whereas clustering is unsupervised, what will be discussed in the this chapter is a form of supervised learning. A set of documents with assigned class labels is used as a training set and a classifier is learned on it. Following this, the classifier is used for assigning new documents to the learned classes.

As the clustering algorithms, classification works with vectors of features constructed from documents. In the terminology of probability theory, a feature vector is considered to be a vector of observed random variables, $d = (F_1, F_2, \ldots, F_m)$. Every feature can take values from a given domain, e.g. $F_i = f, f \in \mathbb{R}$. Apart from this, there is also a class variable C assigned to every document vector in the training set, which is unobserved in case of a new set of documents and must be determined. If there are k classes, then C can take values from the domain $\{0, 1, \ldots, k-1\}$. The classification process is to assign a value to the class variable C for every document vector d.

4.1 Naive Bayes Classifier

The naive Bayes classifier is a simple yet effective probabilistic classifier. It is based on the application of the Bayes rule and depends on strong independence assumptions, therefore it is called naive.

The Bayes theorem for our case is stated as

$$P(C|F_1,\dots,F_m) = \frac{P(C)P(F_1,\dots,F_m|C)}{P(F_1,\dots,F_m)},$$

where $P(C|F_1,\dots,F_m)$ is the probability of a class C when given the vector of features $(F_1,\dots,F_m)$ (i.e. probability that the document vector belongs to the class C), $P(C)$ is the prior probability of a occurence of the class C and $P(F_1,\dots,F_m|C)$ represents the probability of a occurence of the vector of features $(F_1,\dots,F_m)$ in the class C. Finally, $P(F_1,\dots,F_m)$ is the probability of observing the feature vector $(F_1,\dots,F_m)$. However, this term is fixed during the computation of $P(C|F_1,\dots,F_m)$ and serves as a normalization constant. Therefore, it can be omitted.

With the naive assumption that the feature variables are conditionally independent given the class variable, the equation for the classifier can be rewritten in the following form:

$$P(C|F_1,\dots,F_m) = \frac{P(C)\prod_{i=1}^{m} P(F_i|C)}{P(F_1,\dots,F_m)} = \alpha P(C)\prod_{i=1}^{m} P(F_i|C)$$

The classification is then performed according to

$$\arg\max_{c\in C} P(c|F_1,\dots,F_m),$$

the most probable class is assigned to the document vector.

The independence assumption is unrealistic and hardly met in the data. In spite of this fact, it has been shown that the classifier works surprisingly well in real situations, even in those with strong dependencies as shown for example in [6]. The estimation of probabilities computed by the classifier is not reliable, nevertheless for the purpose of classification it is very effective.

The parameters of the probability distribution can be easily estimated from the trainig data.

$$\hat{P}(C=c) = \frac{n_c}{n}$$

and

$$\hat{P}(F_i=f|C=c) = \frac{n_f}{n_c}$$

where n_c is the number of documents assigned to class c, n is the number of documents in the training collection and n_f is the number of cases in which F_i has a value f for a document assigned to class c (for details see [7]).

This maximum-likelihood estimate has also its drawbacks. The training collection is limited and some of the probabilities can become zero, when the feature f is not observed in documents of class c. To overcome this, smoothing of the estimates can be used. It assures that no probabilities will be set to zero (or one). A simple form of smoothing is the additive smoothing:

$$\hat{P}(F_i = f|C = c) = \frac{n_f + \delta}{n_c + n_{F_i}\delta}$$

where n_{F_i} is the numer of possible states of the feature variable F_i and δ is a small positive number.

However, the described methods work only in the case of discrete values of the feature variables, e.g. features generated by the boolean weighting scheme from the section 2.1.2. Continuous domains must be handled in a different way. One approach is to assume some probability distribution of the feature values for a given class and to model the probabilities according to this distribution. For example, a common assumption is the Gaussian distribution of the feature values. After obtaining its parameters from the training data, the mean $\mu_{c,f}$ and the standard deviation $\sigma_{c,f}$, the probability estimation can be computed as

$$\hat{P}(F_i = f|C = c) = n(f; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(f-\mu)^2}{2\sigma^2}}$$

with parameter estimations

$$\hat{\mu} = \frac{1}{n_{F_i}} \sum_{j=1}^{n_{F_i}} f_j,$$

$$\hat{\sigma} = \sqrt{\frac{1}{n_{F_i} - 1} \sum_{j=1}^{n_{F_i}} (f_j - \hat{\mu})^2}.$$

In order to work with the continuous features in the same way as the numerical values, also some form of discretization can be used. A simple discretization method is the equal width interval binning, where the domain is cut into intervals of equal width. The size of intervals is computed as

$$l = \frac{f_{max} - f_{min}}{i},$$

where i is a chosen number of intervals and f_{max}, f_{min} are the bounds of the possible values of the feature variable. Each interval is then handled as a separate feature value.

Another similar method is to divide the domain to equal frequency intervals, where each interval contains the same number of values. Given j instances of feature weights, the values are sorted and then divided to i intervals, such that each interval is comprised of j/i adjanced values.

In both these methods, there is a possibility of putting feature values associated with different classes into the same interval, which can deform the classification. Also more sophisticated techniques can be used, which take advantage also of the class labels. For these more elaborate methods and their comparison with the simple ones see [4]. In implementation of this process, the division to equal width intervals was chosen.

Chapter 5

Implementation

For this work, a program was developed to perform the described clustering and classification algorithms on document collections. Its main purpose is to first cluster a set of documents, then to allow an expert to modify the created hierarchy of clusters and afterwards to learn a classifier on it. Following this, the classifier can be used for categorization of a new set of documents. The whole implemented process is illustrated in the figure 5.1. The program also offers a graphical user interface for working with hierarchies of clusters or classes.

The primary input of the program is a directory with text documents to cluster. The documents are converted to vectors of features with the techniques described in the chapter 2. First, the thresholding is applied. The user can specify the upper and lower threshold and features, which appear in more/less documents than the given upper/lower threshold wil not be considered in the further processing. After determining the features, their frequencies in documents are computed and weighted by the entropy measure.

As the next step, the latent semantic analysis (LSA) is performed. The singular value decomposition is done on the matrix of document feature vectors. The user can set a number of singular values that will be computed. This number must be smaller than the number of documents, which are processed. Smaller numbers of singular values will provide more generalization but also more simplification, therefore there is a danger of supressing important features. Larger numbers of singular values are, on the other hand, more

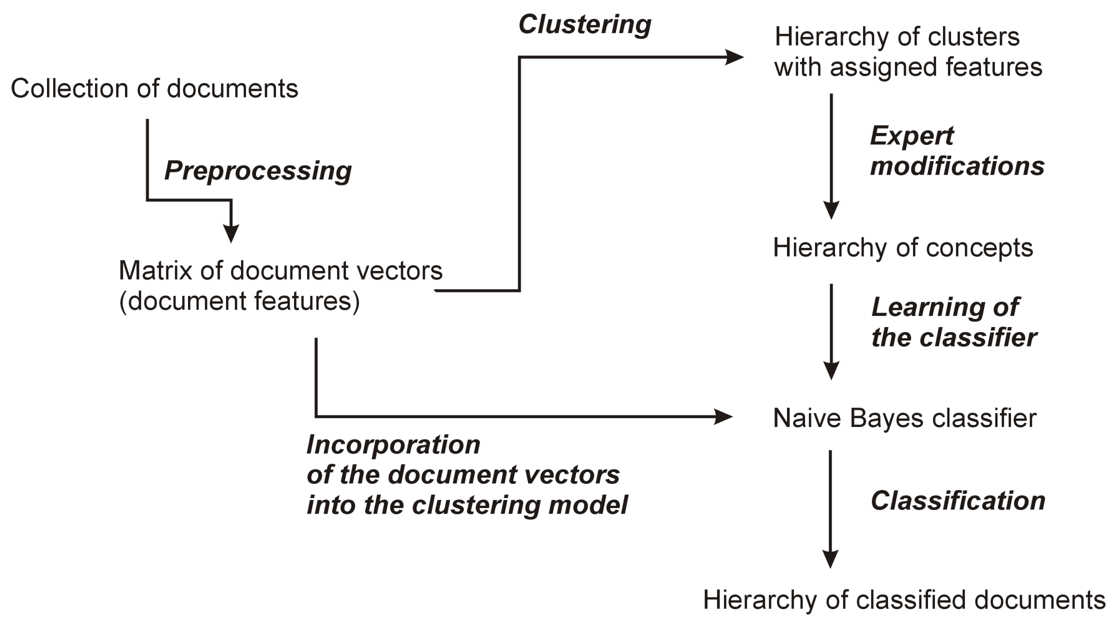

Collection of documents
Preprocessing
Matrix of document vectors (document features)
Clustering
Hierarchy of clusters with assigned features
Expert modifications
Hierarchy of concepts
Learning of the classifier
Naive Bayes classifier
Incorporation of the document vectors into the clustering model
Classification
Hierarchy of classified documents

Figure 5.1: The processing of documents implemented in the program.

computationally demanding and are more liable to overfitting of the later clustering. The singular value decomposition is computed by the ARPACK software [5] (a set of Fortran77 subroutines to solve large scale eigenvalue problems).

The matrices produced by the singular value decomposition are used to compare document vectors between themselves during the clustering step. Clustering is performed by the bisecting k-means algorithm. The process starts with one cluster containing all the documents and recursivelly splits the cluster to get the complete hierarchy. Each split can be recomputed several times with different random choice of the initial centroids to achive better results. This number of iterations of the k-means algorithm is also offered to the user for modification.

By default, the clustering ends when each cluster contains a single document. This behaviour can be changed by setting a different maximal number x of documents in a leaf cluster. Splitting of a cluster then stops, if it contains x or less documents.

After the clustering has finished, features are reassigned to the documents and subsequently also to the created clusters. This step must be performed due to the fact, that the feature vectors used for clustering do not represent real words used in the documents. The latent semantic analysis combines the original features to get a generalized model and thus reduces the length of document vectors. These vectors of recomputed features are hard to interpret and cannot be directly converted to words. Therefore, the features must be reassigned to be able to view them.

To take advantage of the effects provided by LSA, the features are not assigned according to originally computed weights, but compared with documents by the method described in the section 2.2.1. By this comparison, the features which are considered relevant to a given document are assigned to it, even if they do not actually appear in its content. The relevancy is measured by the cosine measure. The user can set a threshold for the relevancy. Features with lower relevancy to a given document than the threshold will not be assigned to it. The features of clusters represent the union of features of documents, which belong to the cluster. The weight is determined as the mean of feature weights of contained documents.

When the processing has completed, the created hierarchy is displayed to the user. At this stage, it is possible to modify the hierarchy in various ways. Documents or whole clusters can be moved from one cluster to another, they can be copied, deleted etc. The clusters do not have to remain disjoint, a document can belong to more than one cluster. Also a cluster can be joined - the hierarchy of its subclusters is cleared and only documents, which belong to the given cluster, are retained. Thus the expert can create a more general hierarchy of concepts.

The program also allows the expert to change features assigned to documents and clusters. Features can be deleted or added, or their weight and form can be altered to better reflect the content of the corresponding document/cluster.

Once the modifications of the hierarchy of concepts (clusters) is completed, a classifier can be learned on it. The program implements the learning of a naive Bayes classifier with either the assumption of a Gaussian distribution of the feature weights, or with a discretization of the feature values.

In the case of learning the parameters of the Gaussian distribution of the feature values, for each class the mean and standard deviation of each of its features is computed. These parameters are then saved into a specified file for subsequent classification.

Similarly, for the discretization of the feature weights the feature domain is split into n equal width intervals. For each class in the hierarchy, the probability of the discretized feature weight value given the class is computed. Again, these probabilities are saved into a file to allow classification of a new set of documents. For this stage of the process, the user can specify the number of intervals, into which will be split the feature domain.

Afterwards, the learned classifier can be used to categorize a new collection of documents. They are preprocessed in mostly the same way, as the documents for clustering. One difference lies in the feature selection - the new document vectors comprise only of the features, that were previously used for the document clustering. Other difference lies in the LSA, the new documents are only added to the model created in the previous clustering process to allow relevant comparison with feature vectors used for learning the classifier.

After the preprocessing phase, the created feature vectors are compared with the class (concept) vectors and assigned to classes in the hierarchy. This process can be performed in two possible ways. The first method assigns documents to classes throughout the whole hierarchy. The second method goes only through the leaf classes in the tree and assigns the document directly to them.

In case of the first method, the whole classification process starts with the root class, which contains all remaining classes. Among its subclasses the most probable according to the learned classifier parameters is determined and the document is assigned to it. The computation repetitively continues inside the cluster, to which the document was inserted, until the leaf class in the tree is reached.

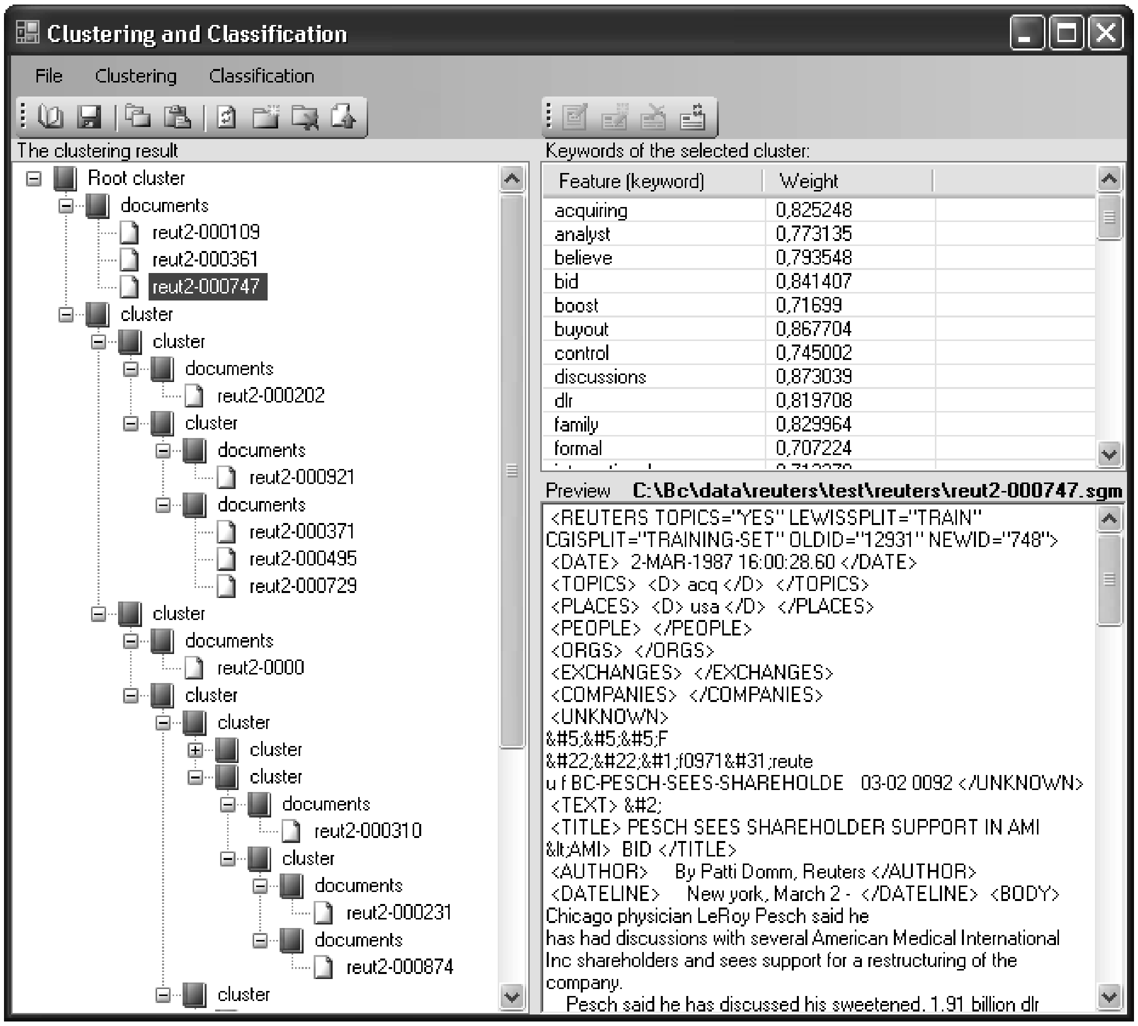
Number of comparisons needed to classify a document depends on the depth of the tree. To classify a document to a leaf class in depth l, $l - 1$ comparisons have to be performed. Because the tree can have branches of very unbalanced sizes, also the number of comparisons greatly varies.

This approach does not guarantee, that a document will be assigned to the most probable cluster throughout the whole hierarchy. The main drawback of this method of classification is, that if the document is incorrectly assigned in some of the the top levels of the hierarchy, it will not be classified to classes, where it should belong. This effect can be reduced by setting a threshold for the probability of a document to belong to a class (concept). With this modification, the document is primarily assigned to all classes, for which the probability of belonging to them is larger than the specified threshold. If the probability for all classes is smaller than the threshold, the document is assigned to the most probable of them.

The second method of classifying a document goes through all leaf classes in the tree. For a document, in each leaf class the probability of belonging to the class is computed and the most probable is chosen. This assures that the most probable class will be chosen. Similarly, the probability threshold can be set to allow a document to belong to more than one class. In this case, number of comparisons needed for classification of a document is alwayes equal to the count of leaf classes.

After the classification, the hierarchy of documents assigned to features is displayed to the user. The user can again modify the resulting hierarchy with use of expert knowledge in the same way as the hierarchy of clusters. There is also a possibility of learning a new classifier on it.

The main part of the program, which provides all the computations, was written in C++ and the code is mostly platform independent. However, the graphical user interface was developed with the use of C# and the .NET Framework and therefore can be run only in the Windows environment. A screenshot of the program is showed in the figure 5.2 and the user documentation is given in the appendix B.


Clustering and Classification
File
Clustering
Classification
The clustering result
Keywords of the selected cluster:
Preview

```
Root cluster
    documents
        reut2-000109
        reut2-000361
        reut2-000747
    cluster
        cluster
            documents
                reut2-000202
            cluster
                documents
                    reut2-000921
                documents
                    reut2-000371
                    reut2-000495
                    reut2-000729
        cluster
            documents
                reut2-0000
            cluster
                cluster
                    cluster
                    cluster
                        documents
                            reut2-000310
                        cluster
                            documents
                                reut2-000231
                            documents
                                reut2-000874
                cluster
```

Feature (keyword)	Weight
acquiring	0,825248
analyst	0,773135
believe	0,793548
bid	0,841407
boost	0,71699
buyout	0,867704
control	0,745002
discussions	0,873039
dlr	0,819708
family	0,829964
formal	0,707224

Preview **C:\Bc\data\reuters\test\reuters\reut2-000747.sgm**

```
 <REUTERS TOPICS="YES" LEWISSPLIT="TRAIN"
CGISPLIT="TRAINING-SET" OLDID="12931" NEWID="748">
 <DATE>  2-MAR-1987 16:00:28.60 </DATE>
 <TOPICS>  <D> acq </D>  </TOPICS>
 <PLACES>  <D> usa </D>  </PLACES>
 <PEOPLE>  </PEOPLE>
 <ORGS>  </ORGS>
 <EXCHANGES>  </EXCHANGES>
 <COMPANIES>  </COMPANIES>
 <UNKNOWN>
&#5;&#5;&#5;F
&#22;&#22;&#1;f0971&#31;reute
u f BC-PESCH-SEES-SHAREHOLDE   03-02 0092 </UNKNOWN>
 <TEXT> &#2;
 <TITLE> PESCH SEES SHAREHOLDER SUPPORT IN AMI
&lt;AMI>  BID </TITLE>
 <AUTHOR>     By Patti Domm, Reuters </AUTHOR>
 <DATELINE>     New york, March 2 -  </DATELINE>  <BODY>
Chicago physician LeRoy Pesch said he
has had discussions with several American Medical International
Inc shareholders and sees support for a restructuring of the
company.
   Pesch said he has discussed his sweetened, 1.91 billion dlr
```

Figure 5.2: A screenshot of the developed program.

Chapter 6

Experimental results

6.1 Evaluation of clustering and classification results

To measure quality of clustering or classification is a difficult task. There are no objective criterions, which would reliably compare solutions and identify the best one. We can only measure some properties and use their combination to determine, which algorithms produce better results.

A level of similarity of documents in a cluster can serve as a measure of clustering quality. If the similarity is small, the clusters are not compact and do not separate natural clusters present in the data well.

The inner similarity of a cluster can be measured as the average pairwise similarity of documents present in the cluster, as measured by the cosine measure:

$$Sim(c) = \frac{1}{n_c} \sum_{d \in c, d' \in c} \cos(d, d'),$$

where c is the cluster, d, d' are documents from the cluster and n_c is the number of documents in the cluster. This measure was used for comparing clustering algorithms for example in [8].

The described measure can be used in situations, when there is no external information about the clustered documents, such as class labels. If such information is available, it is possible to take advantage of it and use it for other measures of quality.

A common approach to evaluating a clustering or classification result with external information is the F score measure, used for example in [8], [9] or [1]. It treats the clusters of documents as a result of a query when class represents the correct result. It measures, how well the result answers to the query (class). The F score is computed as

$$F(l,c) = \frac{2 \cdot \mathit{Recall}(l,c) \cdot \mathit{Precision}(l,c)}{\mathit{Recall}(l,c) + \mathit{Precision}(l,c)},$$

$$\mathit{Recall}(l,c) = \frac{n_{lc}}{n_l},$$

$$\mathit{Precision}(l,c) = \frac{n_{lc}}{n_c}$$

where l represents a class (a class label), c a cluster, n_{lc} is the number of documents from class l, which are placed into the cluster c and n_l, n_c are the numbers of documents in the class l, cluster c. Larger F scores indicate a better clustering.

The F score was designed for a flat clustering. To use it for a hierarchy of clusters, the F score for a class l is determined as the maximum F score reached in all clusters throughout the hierarchy. Subsequently, the F score of the whole hierarchy is computed as the sum of class F scores:

$$F = \sum_{l=1}^{k} \frac{n_l}{n} \max(F(l,c)),$$

k is the number of classes and n the total number of documents.

Another common measure is the entropy. The entropy of a cluster is computed by the formula

$$E(c) = -\frac{1}{\log(k)} \sum_{l=1}^{k} \frac{n_{lc}}{n_c} \log \frac{n_{lc}}{n_c},$$

where again l is a class, c a cluster, n_{lc} is the number of documents in the cluster c, which belong to the class l, n_c is the number of documents in the cluster c and k is the total number of classes. The term n_{lc}/n_c is the probability of a document from cluster c to belong to the class l. Afterwards, the entropy for the whole clustering result is computed as

$$E = \sum_{c=1}^{k_c} \frac{1}{n_{inner}} E(c),$$

where k_c is the total number of clusters and n_{inner} is the number of inner (non-leaf) nodes in the hierarchy. The entropy measure must be used with a caution, that the best entropy shows a result, where each cluster contains only a single document.

6.2 Results

Implemented clustering and classification techniques were tested and compared on English and Czech document collections. Both collections comprise of newspaper articles. As English documents, the widely used Reuters-21578 collection was taken. For testing the performance of the program on Czech documents, a set of various newspaper articles was used. Both collections were split to training and test sets. A detailed description of the data is given in the appendix A.

First clustering of the training collection was performed. The resulting hierarchy of clusters was then used for learning a classifier. Finally, a test set of documents was categorized by this classifier.

No modifications of the clustering hierarchy with use of expert knowledge were performed. Therefore it is possible to see, how well a hierarchy built by clustering can serve for learning of a classifier only by itself. Also the comparisons of results are more objective. With use of some expert modifications, the results would be better.

The Reuters-21578 training collection with 7 063 documents was first used for clustering. For preprocessing, only features, which appear in a single document were disqualified from further computations. This resulted in the total number of 18 890 features. As the number of singular values for the latent semantic analysis, 200 was used. In the computation of the clustering hierarchy, 10 iterations of the algorithm were performed at each split of a cluster and the best of them was used.

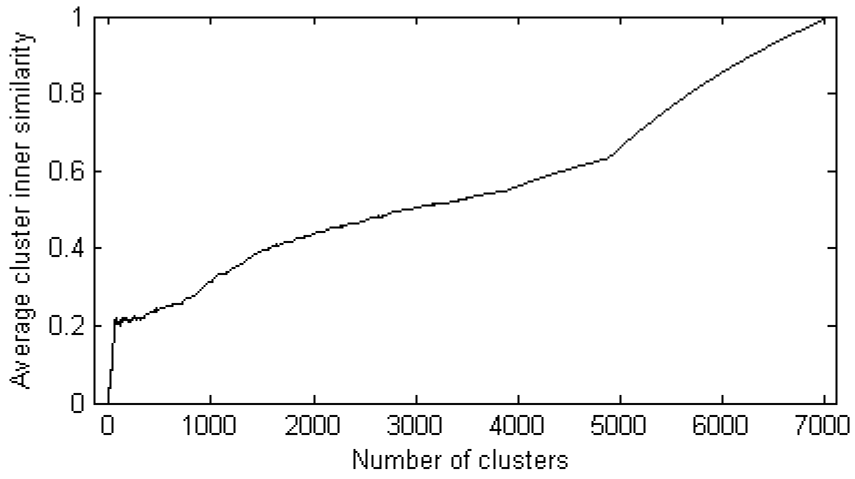


Figure 6.1: Average inner similarity of clusters during the clustering process of the Reuters-21578 training collection of documents.

First, a complete hierarchical clustering was performed, which ends when each cluster contains only a single document. The figure 6.1 shows average inner similarity of clusters in relation to the number of clusters at each step of the algorithm. The inner similarity of clusters was increasing, as the clusters were split to smaller and more compact subclusters.

The average inner similarity increased rapidly in the first approximately 60 splits. This suggests, that there was quite a small number of large natural clusters present in the data. Inside these natural groups, the documents were more compact and did not differ much from each other. The almost linear smooth increase of average inner similarity, which started after the half of the clustering process, was caused by splitting clusters of size 2 to clusters with a single document, which have inner similarity equal to 1.

The F score and entropy value of this result are displayed in the table 6.2 (this result is labeled as Clustering1). The F score value is comparable to the results of the bisecting k-means algorithm reported in [8] on the same dataset. In our case, however, no stop words removal or stemming was used.

As a next step, a clustering hierarchy for learning a classifier was built. The classifier needs to have larger number of documents in each cluster to

Clustering (classification)	**Number of leaf clusters (classes)**	**Average size of leaf cluster (class)**
Clustering1	7063	1
Clustering30	658	10.734
Clustering150	119	59.3529
Classification30 - Gaussian	17	161.294
Classification30 - discretization	67	40.9254
Classification150 - Gaussian	6	457
Classification150 - discretization	48	57.125

Table 6.1: The sizes of clustering and classification results for the Reuters-21578 collection.

learn the parameters of each cluster. To achieve this, a stop condition was set - splitting of a cluster stops, when the cluster contains k documents or less. Some experiments with the various values of k were performed. It should be large enough to allow learning of the classifier and at the same time preserve the hierarchical structure. Clustering with smaller values of k were very liable to overfitting of the classifier, which was learned on them.

For comparison, results for $k = 30$ and $k = 150$ are presented. The number $k = 30$ should preserve the granularity of the data well. On the other hand, the $k = 150$ attempts to capture the larger natural clusters. The sizes of produced clusterings are shown in the table 6.1, the F score and etropy values in the table 6.2 as Clustering30 and Clustering150.

These clusterings were subsequently used for learning of a naive Bayes classifier. The classifier with discretization and with the assumption of Gaussian distribution of feature weights were trained on both clusterings. These classifiers were then used for categorization of the test document collection. Each document was assigned to the single most probable leaf class (cluster) in the hierarchy.

The sizes of produced classifications are shown in the table 6.1 and their F scores and entropies are in the table 6.2 (they are labeled correspondingly to the clusterings, which were used for training). The classifier with discretized feature weights generally outperformed the classifier with Gaussian assump-

Clustering (classification)	**F score**	**Entropy**
Clustering1	0.626601	0.16533
Clustering30	0.616058	0.491801
Clustering150	0.605154	0.777597
Classification30 - Gaussian	0.449385	1.0923
Classification30 - discretization	0.532004	0.665098
Classification150 - Gaussian	0.342882	1.08975
Classification150 - discretization	0.475569	0.833621

Table 6.2: The F scores and entropy values of clustering and classification results for the Reuters-21578 collection.

tion. The criterions of classification quality show worse performance, than the original clustering solution. However, the difference especially for the classifier with discretization is not very large. The worse performance could be caused by overtraining of the classifier on smaller clusters. For this reasons, very few or no documents were assigned to clusters, which were small in the training model. This could be prevented by expert modifications of the original hierarchy to achieve a better result. Unfortunately, some small classes with a few assigned documents were already present in the Reuters collection and could not serve well for training of a classifier.

To prevent the overtraining of the classifier on smaller clusters, also a modification of the clustering process was used, which does not split a cluster, if it will lead to a cluster smaller than some threshold. However, this method led to large clusters, that could not be split because of some small different subset of documents in them, and did not led to an improvement.

The second document collection used in experiments was a set of newspaper articles written in Czech. These documents were not annotated with class labels, therefore entropy measure and F score could not be used for evaluating the results.

For clustering a training set of 4 552 documents was used. These documents were much more diverse than the Reuters collection. More various features were used through the text and therefore more radical feature tresh-

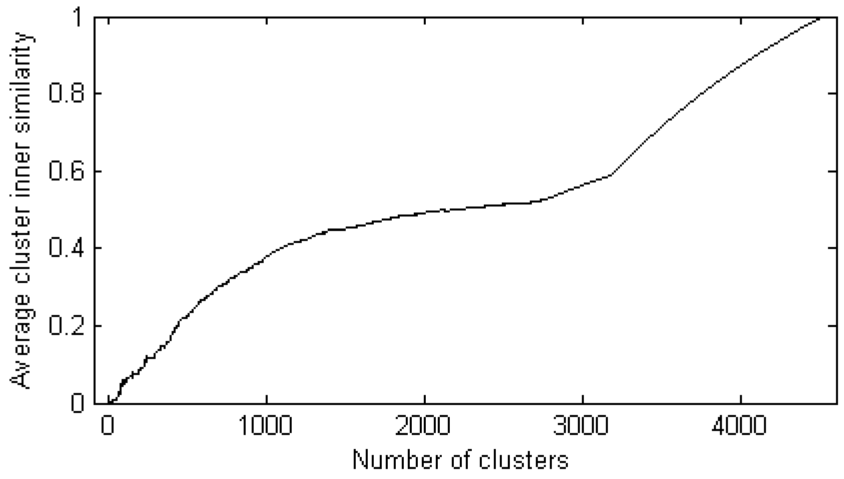


Figure 6.2: Average inner similarity of clusters during the clustering process of the Czech newspaper articles.

olding had to be used. In the preprocessing step, all features which appeared in less than 5 documents and in more than 3500 documents were excluded from the computation. It resulted in the total number of 21 313 features. This number is still larger than in the Reuters documents, although the size of the set of documents is smaller. For the latent semantic analysis, 200 singular values was used as in the previous case.

First, the complete clustering was performed with 10 iterations of the bisecting k-means algorithm at each split. The average inner similarity of clusters during this process was computed and the result is shown in the figure 6.2.

The average inner similarity was increasing more steadily than in the case of Reuters documents. This also confirms larger diversity of the data.

Similarly as in the previous case, the clustering with stop condition of at most 30 and 150 documents in a leaf cluster was performed. The sizes of these clusterings are shown in the table 6.3 as Clustering30 and Clustering150.

On these hierarchies, both Bayesian classifiers were trained and afterwards used to classify the test set of documents. The classifiers turned out to be very liable to overfitting. They tended to assign documents only a few

Clustering (classification)	**Number of leaf clusters (classes)**	**Average size of leaf cluster (class)**
Clustering1	4 552	1
Clustering30	825	5.48121
Clustering150	67	67.4925
Classification30 - Gaussian	3	687.333
Classification30 - discretization	30	68.7333
Classification150 - Gaussian	3	687.333
Classification150 - discretization	17	121.294

Table 6.3: The sizes clustering and classification results for the collection of Czech documents.

large classes and smaller ones were left empty. This was caused partly by the large variability of features in documents and partly by a smaller number of training documents, than in the Reuters collection. The results are shown in the table 6.3

It seems important to use some additional techniques for processing documents in Czech. The stop words removal and some form of stemming could reduce the large variability of the data and improve the result.

Chapter 7

Summary

In this work a clustering of documents was proposed as a way of creating a training collection, which could be used for construction of a classifier. A program was developed to perform these tasks and its results were tested on clustering and classification of document collections in Czech and English.

The clustering provided by the bisecting k-means algorithm on the Reuters-21578 document collection showed similar results as were reported by [8]. The clustering result can be modified with use of expert knowledge. For this work this step was omitted to see, how well the clustering alone can create a training collection for the classifier. A naive Bayes classifier with either the assumption of Gaussian distribution of feature weights or discretization of feature weights to intervals was trained on the clustering. Afterwards, the classifications produced by these classifiers were compared.

The classifier, which used discretized feature values, showed better results in all experiments compared to the classifier with Gaussian assumption. The classification had sligtly worse performance on the test set than showed by the original clustering on the train set. It is caused by the noise in the clustering solution and by overfitting of the classifier on small clusters. This can be prevented by expert modifications of the clustering hierarchy before learning of the classifiers. Also some more sophisticated classification methods than the naive Bayes classifier could be used to improve the results.

The documents in Czech were hard to process because of large diversity of the data. In consideration of this fact, the collection of documents was relatively small to provide enough training patterns for a classifier. The results of clustering and classification of Czech documents were also hard to evaluate due to absence of any class labels in the data. For better results for Czech documents, stop words removal and some form of stemming should be used. This would reduce the noise in the data and also reduce the large diversity.

We have shown, that the clustering can serve as a good tool for creating initial division of documents to clusters, which can greatly simplify the process of creating a training set for document classification by an expert. The described algorithms performed generally better on documents in English than in Czech and more work should be done to process the documents in the Czech language more successfully.

Appendix A

Data used for testing

A.1 Reuters-21578 collection

As the first collection of documents for experiments, one of the most widely used collections was used, the Reuters-21578 text categorization test collection (Distribution 1.0) [1].

It is distributed in 22 files, which contain together 21 578 Reuters articles, which were published in 1978. They were manually indexed with topics (classes). A document can have zero or more topics assigned to itself. There exist a few splits of the collection to train and test set of documents. For this work, the "ModApte"' split was used. It divides the collection to three sets. The first one contains 9 603 training documents, the second one is used for testing and is composed of 3 299 documents. The last one with 8 676 documents is intended for statistical analyses of the data and is not used for clustering or classification.

However, after examination of the data, it turned out that some of the documents from the training and test sets do not have a topic assigned or they have an empty body. In [1], there is an overview of some researches, which were performed with use of the Reuters-21578 document collection. Part of the researches were done with the original split of documents, other part used a modified collection, with some of the documents removed.

[1]This document collection is available at
`http://www.daviddlewis.com/resources/testcollections/reuters21578`

For this work, we chose to disqualify the empty documents or documents without a topic from the processing, following the setup used in [1]. This resulted to 7 063 documents in the train collection and 2 742 documents in the test collection. In the training set, 114 classes (topics) were used to index the documents. The most frequent class was assigned to 2709 documents and the smallest class to only a single document. Average size of a class is 77.24 documents.

The train set was used for clustering and learning of a classifier, whereas the test set was used for classification.

A.2 Czech newspaper articles

The second collection of documents is a collection of czech newspaper articles. Compared to the Reuters-21578 collection, they were not annotated in any way. They come from the collections of documents, which are used as sources of texts for The Prague Dependency Treebank.

The collection is composed of some articles from the following sources:

- Lidové noviny (daily newspapers), ISSN 1213-1385, 1991, 1994, 1995
- Mladá fronta Dnes (daily newspapers), 1992
- Českomoravský Profit (business weekly), 1994
- Vesmír (scientific journal), ISSN 1214-4029, Vesmír, s.r.o., 1992, 1993

The original collection of 6828 documents was split to the training set with 4552 documents and test set with 2276 documents. Documents were assigned to these sets at random.

Appendix B

User documentation for the program

In this chapter, a user documentation for the developed program is provided. It is presented in a form of a demonstrational walk though the whole clustering and classification process, as it is executed by the program. A sample collection of documents is provided on the the enclosed CD, which will be used through this chapter for demonstration. It is recommended to open the program and follow this guide through this sample walk to explore all its options.

To run this program, Windows operating system and .NET Framework 2.0 (or higher) is requiered. The program is located in the `program` folder on the CD and is named `ClusterClass`. Installation of the program lies in the copying of the whole program folder. The dll files, which are also in the folder, must stay in the same folder as the actual program.

B.1 Task: Clustering

As an input for clustering, a folder of documents is used. On the enclosed CD, a sample set of 100 documents can be found in the `sample\clustering` folder.

First open the program and select the item *Start clustering* in the *Clustering* menu. A dialog window will appear, where the parameters of the clustering process can be set. It is composed of four tabs, each represents a step of the clustering process.

In the *General Properties* tab, the input directory with documents to cluster is set. Apart from this, it is also necessary to provide a directory for auxiliary files needed for work. By default, all output files are saved into this folder. When the button *Show details* is pressed, the default filenames are displayed for possible modifications. However, it is recommended to leave them in their default form, because they will be needed in later processing, e.g. during learning of a classifier. Now you can set the documents folder as the `sample\clustering` from the CD. Then select an arbitrary local directory for work.

In the *Preprocessing* panel, you can set parameters for creating the document vectors. First there are the upper and lower thresholds for feature occurences. Features, which appear in a smaller number of documents than the lower threshold, or in larger than the upper threshold will not be included in the computation. To demonstrate, a lower threshold 1 and upper threshold 95 will skip the words, which appear throughout the sample collection in more than 95 documents or which appear only in a single document.

A following property to set up is the number of singular values, which will be computed for the latent semantic analysis as described in the section 2.2.1. Larger values are more computationally demanding and do not provide generalization, smaller values can be too simplifying. Therefore, it is ideal to try a few settings and choose the one which produces the best results. It is important, that the number of singular values must be smaller than the number of documents, which will be clustered. For the sample case, set 30 singular values. Again, it is possible to set manually the names of the working filenames.

As the next step, the parameters for clustering are presented in the *Clustering* tab. Here you can set the number of iterations of the bisecting k-means algorithm described in the section 3.1.3. Larger numbers of iterations increase the computational requirements, but also increase the probability of finding a better solution and not ending in a local minimum of the criterion function. By default, 10 iterations are used. You can also set a stopping criterion for the bisecting k-means algorithm. Processing of a cluster finishes, when it contains at most the specified number of documents. If this value is set to 1, the computation stops in the moment, when each cluster contains only a single document.

The last tab presents the parameters for the feature assignment. Here you can set a threshold for assigning features to documents. Features, which will show a cosine similarity with a given document larger than the specified threshold, will be assigned to it. A default value is set to 0.7, which corresponds to the angle 45° between the vectors.

After completion of these settings, you can push the button *Start clustering* and the whole process will be performed. You can also execute every phase separately by pushing the *Perform preprocessing/Perform clustering/Generate features* buttons. This requires maintaining the correct order of the individual phases and setting correct filenames for each step.

When the clustering of the input document collection has finished, the resulting hierarchy is displayed in the main window. Subsequent actions are described in the next section.

B.2 Task: Modifying the hierachy

This section provides a description of the main window environment. If you have just completed a clustering of a document collection, the resulting hierarchy is automatically displayed. Otherwise, you can open a sample hierarchy from the CD. Select the *File* - *Open* item in the main menu. Then select the collection of documents as `DocumentList.txt` and the clustering result file as `Clustering.xml` from the `sample\results` folder from the CD.

Now you should see the resulting tree of clusters in the left part of the window. The upper-right part of the window serves for displaying a list of features assigned to a selected document/cluster. In the lower-right part a preview of a document is displayed, if a document is selected in the hierarchy. In case of a cluster, it shows a list of documents, that belong to the selected cluster.

You can save the created hierarchy by clicking on the *File* - *Save* item in the main menu. The result is saved in the XML format. To open a clustering hierarchy, it is necessary to provide the file with the clustering result and also a text file with the list of documents, which has been clustered. This file is created during the preprocessing stage and its default file name is `DocumentList.txt`. If you have not changed this name, it should be located in the working folder.

The figure B.1 shows a description of tools available for modifying the hierarchy. It is possible to drag documents and clusters from one location to another, to copy and paste them, to create new clusters or to delete them. For example, if a document is assigned to an improper cluster, it is possible to move it by mouse to another.

After some modifications, the tree of clusters can be degenerated - an empty cluster or a path formed by nested clusters can remain in the tree. To solve this problem, you can push the *Reload hierarchy* button and these malforms will be eliminated.

Another modification is a join of subclusters. The built hierarchy is defaultly a binary tree with each document as a leaf cluster. However, for the purposes of learning of a classifier it is more suitable to have leaf clusters with a larger sets of documents. To enable this, it is possible to join subclusters of a specified cluster, i.e. to delete the structure of its subtree of clusters and to make it a leaf node. This node will contain union of documents from its original subclusters. The join is performed by selecting a cluster in the hierarchy and pushing the *Join subclusters* button on the toolbar.

Modifications of feature vectors is also possible. When a cluster or a document is selected in the hierarchy tree, its features together with their weights are displayed in the upper-right part of the main window. They are sorted alphabetically. You can alter a form of the feature or adjust its weight. It is also possible to add new features and delete them. If a feature is deleted, it is removed only from the actually selected document/cluster. If some features are removed from documents, they remain assigned to their parent clusters. To resolve this, select a cluster and push the *Update features* button. The features assigned to the cluster will be recomputed to contain again union of features from its subclusters. These features will be afterwards used for learning of a classifier. All the described actions are performed by pushing corresponding buttons on the toolbar (see the figure B.1).

After the hierarchy is modified to a satisfactory state, you can save it and proceed to learning of the classifier.

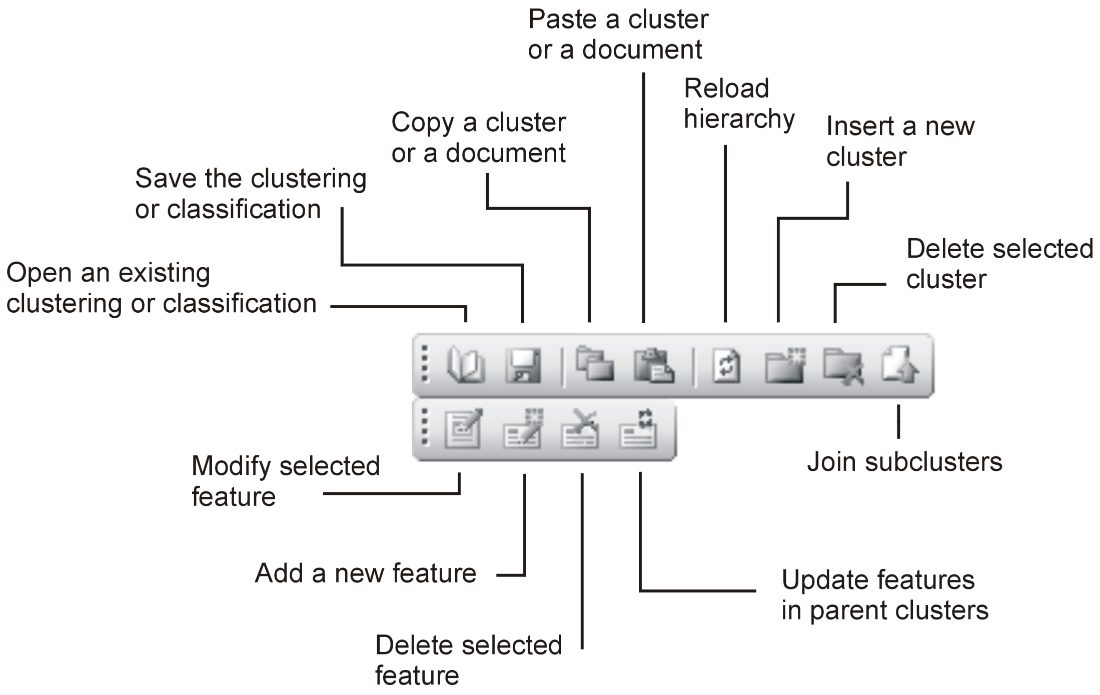

Paste a cluster or a document
Reload hierarchy
Copy a cluster or a document
Insert a new cluster
Save the clustering or classification
Delete selected cluster
Open an existing clustering or classification
Join subclusters
Modify selected feature
Add a new feature
Update features in parent clusters
Delete selected feature

Figure B.1: The program's toolbar - a description of control buttons.

B.3 Task: Learning a classifier

For building a classifier, select *Create classifier from clustering* item in the *Classification* menu. A dialog window will appear. This window has two tabs, each for a different type of classifier.

The first tab allows to build a naive Bayes classifier with an assumption of a Gaussian distribution of feature weights. It requires the user to set two input filenames. The fist one is a complete list of features from the document collection, which was clustered. This list was created during the preprocessing of documents for clustering. Its default filename is `FeatureList.xml` and it should be located in the corresponding working directory. The second input filename is the clustering result, wich will serve as a training model for the classifier. Its default filename is `Clustering.xml`. Again, sample files can be found in the `sample\results` folder on the enclosed CD.

Two remaining fields to fill are the files, to which the classifier will be saved. The first one represents the hierarchical structure of the classes. The next file will contain learned parameters of the classifier. Again, it is recommended not to change the default names. After pressing the *Create classifier* button, the classifier will be learned on the specified clustering.

The second tab presents a building of a naive Bayes classifier with discretization of the feature weights. The first four fields are the same as in the previous case. For this type of classifier, a number of intervals i, to which will be split the feature domain, is also set. The feature domain will be divided to i equal-width intervals. Its default value is 10. Again, after setting these parameters, the classifier can be built.

When the training of the classifier is finished, classification of a new set of documents can be performed.

B.4 Task: Classification

The classification process is in many respects similar to the clustering task. In the *Classification* menu select the *Start classification* item. A dialog window will appear. It is composed of three tabs.

The first tab shows general properties, which are the same, as in the clustering window. You should set a folder with documents, which will be classified, and a working folder. On the enclosed CD, there is also a test collection of documents to use for this purpose. It contains 40 text documents and is located in the `sample\classification` folder. Similarly, after pushing a button *Show details*, the filenames, which will be used, are displayed for possible modifications. It is recommended to set the same working directory, as was used for the previous clustering.

In the *Preprocessing* tab, properties for creating document vectors are set. Preprocessing of documents for classification requires some files created during the clustering of documents, which was used for learning of the classifier. Implicitly, the default filenames are expected in the working directory. If you have changed them in the preprocessing for clustering, you should change them appropriately here. If you used the default filenames, you can simply skip this tab.

In the last tab, parameters for classification itself are set. You can choose a file to store the result of classification. Next, set the filenames, in which the built classifier is stored. On the CD, you can also find the classifier with the Gaussian assumption of distribution of feature values (`sample\results\ClassGaussHierarchy.xml`, `sample\results\ClassGaussParams.txt`) and the classfier with discretization of the feature values (`sample\results\ClassDiscrHierarchy.xml`, `sample\results\ClassDiscrParams.txt`).

At this point, decide if you want documents to be classified to more than one class and what threshold will be used in this case. Documents will be assigned to all classes, for which the probability to belong there is larger than the specified threshold. You can also choose, whether documents should be classified directly to leaf classes in the hierarchy. If you uncheck this option, the classification will start in the root class, where the document will be assigned to its most probable subclass, and proceed recursively until a leaf in the tree is reached.

Similarly as in the other tabs, under the *Show details* button, modifications of used filenames are possible. There is also an eventuality of performing preprocessing of documents and classification separately by pushing the buttons *Perform preprocessing* and *Perform classification* at each tab. To perform the whole process at once, push the *OK* button.

After the classification completes, the resulting assignment of documents to classes is displayed in the main window. This resulting hierarchy can be viewed and modified in the same way as the hierarchy created by the clustering process.

Bibliography

[1] Aas, K., Eikvil, L.: *Text categorisation: A survey*, technical report, Norwegian Computing Center, 1999.

[2] Berkhin, P.: *Survey of Clustering Data Mining Techniques*, Accrue Software, 2002.

[3] Berry, M. W., Dumais, S. T., O'Brien, G. W.: *Using Linear Algebra for Intelligent Information Retrieval*, SIAM Review 37(4):573–595, 1994.

[4] Dougherty, J., Kohavi, R., Sahami, M.: *Supervised and unsupervised discretization of continuous features*, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City 1995.

[5] Lehoucq, R. B., Sorensen, D. C., Yang, C.: *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi methods* , technical report, Rice University, 1997.

[6] Rish, I.: *An empirical study of the naive Bayes classifier*, IJCAI Workshop on Empirical Methods in Artificial Intelligence, 2001.

[7] Russel, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*, Second edn. Prentice Hall, 2003.

[8] Steinbach, M., Karypis, G., Kumar, V.: *A comparison of document clustering techniques*, KDD Workshop on Text Mining, 2000.

[9] Zhao, Y., Karypis, G.: *Hierarchical Clustering Algorithms for Document Datasets*, Data Mining and Knowledge Discovery, Vol. 10, No. 2, pp. 141-168, 2005.