technische universität
dortmund

Bachelor Thesis

A Survey on Subspace Clustering

David Brendan Spain
September 2013

Evaluators:
Prof. Dr. Katharina Morik
Dipl.-Inform. Christian Pölitz

Technische Universität Dortmund
Faculty of Computer Science
Chair of Artificial Intelligence (VIII)
http://www-ai.cs.uni-dortmund.de/

Contents

Chapter 1

Introduction

Cluster analysis has become an essential classification method in machine learning. To cluster a set requires its partition into smaller sets (clusters) under the mandate that objects in a cluster need to be as similar to each other as possible. Text clustering is the application of these techniques on documents. The most common use of text clustering in information retrieval is to group queried documents into different genres based on their contents.

The biggest challenge of text clustering is the so called "curse of high dimensionality" [9]. The most common representations of documents in information retrieval have their dimensionality bound by their common vocabulary, which leads to data vectors with more than 10000 different dimensions. This high dimensionality renders classical clustering methods, which heavily rely on Euclidian metrics to calculate similarity between objects, useless.

Subspace clustering provides a solution to this problem by first searching for small subspaces of the data space, in which objects of the same class are close to each other. In contrast to global dimensionality reduction the subspaces found preserve possible relations between documents. Another advantage of clustering in much smaller subspaces is that Euclidian metrics become a meaningful discrimination criterion again.

In this thesis the viability of three specific subspace clustering methods for the application of homograph disambiguation is discussed; homograph disambiguation has similarities to the task of genre classification, a problem that is often solved satifactory by clustering. A manual categorization of homographs is too resource-intensive because the corpora that are used for language research are too large. Good homograph disambiguation could aid linguistic research in quantifying the usage of the different senses of a homograph plotted against time. It could also help to find the origin of a new sense.

1.1 Thesis Structure

Chapter two provides the linguistic background of homograph disambiguation. The problem is then linked to genre indexing; a common application of clustering in text mining. KobRA, which provides the data for the experimental evaluation, is introduced.
Chapter three presents the clustering procedures that operate on the global data space in addition to common methods of dimensionality reduction, as well as the vector space model, which is used to represent documents in text mining.
The next chapter explains in detail why common clustering methods do not work on high dimensonal data and the infeasibility of global dimensionality reduction as a solution for text clustering. An overview of the current branches of subspace clustering is given. The advantages of subspace clustering for text clustering are presented.
In chapter four and five the three subspace clustering algorithms CLIQUE, FTC and K-Subspace, and their implementation into the data mining suite RapidMiner, as an extension, are introduced.
Chapter six presents the experimental evaluation method and the results of the empirical analysis of the implemented algorithms.

1.2 Notations

$dist(a, b)$	Distance between a and b
$X = \{X_1, \ldots, X_n\}$	Data set
X_i	Data Point i
$C = \{C_1, \ldots, C_k\}$	Clusters
C_i	Cluster i
$D = \{D_1, \ldots, D_n\}$	Document corpus
D_i	Document i
$T = \{t_1, \ldots, t_j\}$	Vocabulary of the corpus D
t_i	Term i
$R_{\{I\}} = \{\underset{i \in I}{\times} t_i\}$	An axis parallel subspace containing the terms indexed by set I
	for example$(R_{\{1,2,5\}} = t_1 \times t_2 \times t_5)$

Chapter 2

Linguistic Background

2.1 Homographs

A homograph is a word that shares the same spelling to a set of words with different meanings. Most homographs are also homonyms; they share the same pronounciation in addition to the same spelling. There are two main categories of homonyms[16].
The first one is the coincidental homonym. A homonym of this category may be the result of a phonetic convergence. Two previously different spelling forms merged, or a new word was given as spelling to a sense that coincided with an unrelated older word with a different meaning.
The second category is that of the polysemous homonym. In this case multiple divergent meanings derived historically from the same word often preserving a perceptible metaphoric or semantic link.
There are two instantly obvious lexicographic problems that arise from the definition of homographs. The first one is the correct assignment of a word in either the coincidental or the polysemous category. Two coincidental homonyms may also share a weak semantic link while a polysemous semantic link may become opaque.
The second problem is general disambiguation based on context and the search for the original sense of a set of homonyms and their natural order in terms of first appearance and frequency of use.
Homographs only discriminate based on spelling; because of that heteronyms are also a part of the set of homographs. The fact that heteronyms share the same spelling but have a different pronounciation for different meanings leads to homograph identification being an essential step in text to speech synthesis[33].

2.2 Genre Indexing for Homograph Disambiguation

Given a set of documents and a set of genres the task of genre indexing encompasses correctly assigning a document to one (or more) genre(s) based on the content of a document; multiple mentioning of 'spaceships' and 'faster than light travel' will most likely result in an assignment to the 'Science Fiction' category. One method of assigning data to genre categories is clustering. Points that share a cluster (the genre) are similar to each other, just like books of a certain genre often feature a noticable jargon.
The idea this thesis discusses is that the meanings or 'senses' of a homograph are similar to genres. If a homograph is found in a document, the surrounding text may provide enough context to identify the correct meaning of the homograph [15]. Unfortunately a few problems arise by utilizing genre classification methods for homograph identification. The first is that unlike genres the growth of the number of senses for a homogaph is volatile. New documents may feature senses that have not yet been officially recorded. This may lead to false identifications or the categorization of a new meaning as noise.
The second consideration that has to be made when comparing classification of homograph meanings with genre is the lack of ambiguity on the side of homographs. Given proper context and ignoring edge cases like poems and jokes homographs always relate to only one sense whereas a book may belong to multiple genres.
Another problem is finding multiple instances of the same homograph in one document. Some homographs like 'lie' are common enough to appear multiple times with different meanings in the same document; because of this, the only context that can confidently be used for disambiguation are sentences in direct proximity to the homograph, when applying the categorization.

2.3 KobRA

"Korpus-basierte Recherche und Analyse mit Hilfe von Data-Mining" in short, KobRA translates to: Corpora based research and analytics with the help of data mining [21]. Its goal is to develop and apply data mining and machine learning methods to German language corpora. These methods are to aid German linguists and lexicographs in finding and verifying new theses. There are currently three specific research fields covered by KobRA [20]:

Internet lects and communication The analysis of speech features in internet-based communication and its comparison to standard-compliant written language found in articles, books and other standard texts.

Lexicography Finding significant and unusual documents to discover the origins of semantic changes. The visualisation of these changes plotted against time.

Diachronic linguistics The development of vocabulary, syntax, morphology in a given investigation period. Studies on processes of lexical change and the influence of contact languages and diatopic varieties.

Chapter 3

Regular Cluster Analysis and PCA

3.1 Cluster Analysis

Cluster analysis is a set of procedures that aims to detect similarity between objects belonging to a large data set. The grouping of objects that have been determined similar under a set of conditions are grouped together. These groups are referred to as "clusters". In addition to grouping together objects, cluster analysis also has the goal of discovering previously unknown groups of data and providing a comprehensible description of the groups main similarity features. A significant advantage provided by group discovery is that a data set can be analysed by clustering methods without requiring any prior knowledge, also refered to as an unsupervised classification.
The main distinction of different clustering algorithms is the way objects are assigned to a cluster. We differentiate between hard clustering - objects are either assigned to a cluster or they are not - and soft clustering - objects are given a probability distribution of belonging to one of all found clusters.

3.1.1 Hierarchical Clustering

The core idea of hierarchical clustering is that objects that are more similar to each other reside in a smaller distance to each other than dissimilar objects. To describe its clusters hierarchical methods use the distance it takes to connect all components of a cluster. If we plot the distance against the set of found clusters a graph is created that starts with every point being a cluster. As the distance grows more points will satisfy the condition of reaching their neighbors by travelling the given distance. Clusters grow gradually until there is only one cluster containing all objects [28]. The graph that now shows the hierarchy of clusters is called dendrogram.
There a are two approaches to achieve hierarchical clustering: The first one is to start from the bottom with every point being a cluster and growing the distance until enough points have merged into clusters to reach a satisfyingly low number of clusters.

The other method is to start with one cluster containing the whole data set and recursively splitting clusters until a number of clusters high enough is reached.
There are many different methods to determine when two clusters are merged into one based on their distance. The most popular are:

Single-Linkage The smallest possible distance between the objects of cluster A and B is compared against the threshold distance $min\{dist(a,b) : a \in A, b \in B\} < d$

Complete-Linkage The largest possible distance between the objects of cluster A and B is compared against the threshold distance $max\{dist(a,b) : a \in A, b \in B\} < d$

Average-Linkage The average distance of each point A to B is calculated and compared against the threshold distance $\frac{1}{|A|\cdot|B|} \sum_{a \in A, b \in B} dist(a,b) < d$

3.1.2 Centroid-based clustering

In centroid based clustering each cluster is represented by a single point that does not need to be part of the data set; this point is called the cluster center. An object is assigned to a cluster C_i if its distance in relation to the point representing the cluster is smaller than any other cluster center. Most centroid based clustering methods require the user to specify the number of clusters that need to be found. The calculation of the ideal positions of the cluster centers to minimize the dispersion of the clusters is NP-hard. Instead of explicitly calculating the ideal clusters the most popular centroid clusterers like k-Means [18] approximate the ideal clusters by randomly selecting cluster centers and iterating until a local optimum is found.

3.1.3 Density-based clustering

Density based clustering methods define a cluster as a region in which a higher density of data points exist compared to the complete value range of the data set. One way to calculate the density of the data space is to partition the feature space into units of equal size. The disadvantage of this method is that the clustering quality is heavily dependent on the size of these units. If no previous knowledge of the data, which is to be clustered, exists it may take several runs with different unit sizes to get a satisfying result.
The most popular density based algorithm is DBSCAN. It determines dense regions by constructing an ϵ-region around each data point [13]. If enough neighbors of the point lie in the ϵ-region the point is set as part of a dense cluster. For each found neighbor the same check is performed if they also have enough neigbors in their ϵ-region they are also added to the cluster and the same check is performed for their neighbors. If they do not have enough neighbors in their ϵ-region they will be added to the cluster but their neighbors

will not be checked again. After the check is performed for all the points they are either assigned to a dense cluster region or categorized as noise.

3.2 Metric

Almost all clustering methods rely heavily on the use of distance measurements to determine the assignment of a point a to a cluster b. The shape of a cluster can change radically if the metric used for the distance measurements is changed. Some points that were nearest neighbors under one metric might be the farthest appart under another. The most common distance metrics used for clustering are:

- Squared Euclidean distance: $\|a-b\|_2^2 = \sum_i (a_i - b_i)^2$
- Manhattan distance: $\|a-b\|_1 = \sum_i |a_i - b_i|$
- Maximum distance: $\|a-b\|_\infty = \max_i |a_i - b_i|$
- Cosine similarity: $cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|}$

3.3 Vector Space Model

Since metrics are heavily utilized to make the assignment of an object to a cluster, any data that is used needs to support them. This provides a challenge for document clustering, because it is impossible to provide a universally correct distance between two different documents. The first step of document clustering therefore has to be the transformation of a text into a formal mathematical representation of itself, the vector space model [27].
The simplest vector space model is the boolean model. With a collection of n documents that are part of a corpus D as an input, each document d_i of this corpus is given a unique index number $1 \leq i \leq n \in \mathbb{N}$. The first step is to create the collection of all terms that occur in the documents of D. This set of terms is called the vocabulary and is denoted by T, each term is also given a unique index $1 \leq j \leq m \in \mathbb{N}$.
The boolean model is only interested in the occurrence of a term out of T in a document d_i. For each document a binary vector $v_i \in \{0,1\}^m$ is created. If the j-th term out of T occurs in document d_i the j-th component of v_i is set to 1. All document vectors are then aggregated into a $n \times m$ matrix that now represents the corpus D as a boolean model.
While the boolean model is great for its simplicity, it lacks providing the complete information about the terms of a document. The "bag-of-words" model gets nearer to a full representation of a document by substituting the binary occurence of a term with the actual number of occurences $TF_{d_i}(t_j)$ in the document d_i.
The most commonly used vector space model for document representation is TF-IDF (term

frequency-inverse document frequency) [7]. In addition to modelling the local representation of a document d_i, it also models its context in the corpus by creating a weighted term vector for each document. The idea is to weight terms that occur in a great number of documents in D, lower because they might not be as significant in modelling the specificity of that particular document. The inverse document frequency is defined as:

$$IDF(t_j) = log(\frac{n}{DF(t_j)})$$

$DF(t_j)$ indicates how often the term t_j occurs in all documents of D. A logarithmic scale is used to mitigate the effect of the IDF for very small numbers of $DF(t_j)$. The j-th position of the document vector $v_i \in \mathbb{R}^m$ is calculated as

$$TF_{d_i}(t_j) \cdot IDF(t_j)$$

3.4 Principal Component Analysis

Principal component analyis (PCA) is one of the most well known procedures to structure large sets of data. For a set of possibly correlated variables PCA returns a set of principal components. The set returned is often orders of magnitude smaller than the initial data set. Components of this set are orthogonal to each other and thus linearly uncorrelated. They are defined as the indicators of variance of the original data. The first component accounts for the largest amounts of variance and further components are ordered descendingly by their accountability of variance.

PCA's most common use is dimensionality reduction [11]. The original data set is transformed into a new system spanned by the set of principal components.

The most intuitive way to describe how principal components are found is to think of a set of scattered points in a multidimensional cartesian system. The first principal component is a line which is closest to every data point. The solution to this problem is the minimization of the sum of Euclidian distances of all points to the line. The first step of this "intuitive" calculation is to find the centroid of all scattered data points. A line that is closest to all data points must include this point. If we were to calculate the sum of distances for all lines going through this point and select the line with the smallest sum of distances we successfully selected the first principal component. Further principal components are selected the same way with the additional constraint that the next selected line has to be perpendicular to all the lines that were selected before [19].

The statistical model most closely reflecting this intuitive but computationally unfeasible approach is the eigenvalue decomposition of the covariance matrix. In the first step the covariance matrix Σ of the data set is created. Covariance is a measurement to find any

relationship between the dimensions of the data set; the covariance of a set of observations with itself is the variance. The covariance between a vector A and B is

$$cov(A,B) = \frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{n-1}$$

where $\bar{A}$ is the mean of the set A. The covariance matrix Σ is a representation of all covariance relations the data set contains. Next the eigenvalue matrix Λ is calculated. This matrix has the eigenvalues of Σ in a descending order on its diagonal; all other values of the matrix are zero. The eigenvectors calculated to the corresponding k first eigenvalues build the system of principal components of the data set [19].

While the eigenvalue decomposition might be a very simple method for calculating the principal components, it has the disadvantage of memory use. The required dense covariance matrix has a memory footprint of $\mathcal{O}(n^2)$ with n being the number of examples of the example set. In the implementation of PCA in this paper the method of single value decomposition is used to calculate principal components [1].

Chapter 4

Clustering High Dimensional Data

4.1 High-Dimensional Data

Data in natural language processing, contingent upon the domain of a modern dictionary, features a high number of attributes in its most common representational structures (bag of words, term frequency vector). Apart from obvious disadvantages, like difficulties in intuitive understanding and memory allocation, high-dimensional data poses one more challenge, which is especially relevant to clustering algorithms, in the form of distance convergence [9]:

$$\lim_{dimensions \to \infty} \frac{distance_{max} - distance_{min}}{distance_{min}} \to 0$$

Most distance measurements become useless in high-dimensional feature spaces, because the difference between the nearest and farthest neighbor of one data point becomes negligible [14]. Since most conventional clustering algorithms rely on these distance measures to assign points to a cluster, another more meaningful way for assignment needs to be provided.

4.2 Global Dimensionality Reduction

In some instances, high dimensional datasets may have their points distributed within a lower dimensional manifold. Dimensionality reduction methods aim to correctly model the structure of the manifold project all points of the higher dimensional space into the lower dimensional manifold space. Classical methods like PCA require the manifold to be embedded linearly into the data space.
More recent non-linear dimensionality reduction methods only require local linearity. ISOMAP, for example creates a matrix containing the pairwise distance between points sharing a local neighborhood and then uses the Floyd-Warshall algorithm to calculate the global geodesic distance between all points on the manifold [31]; after this simple linear multidimensional

scaling is performed, which is similar to PCA but uses a dissimilarity matrix instead of the covariance matrix [11].
NLDR methods get closer to solving the problem of high dimensionality. The criterion of local linearity is shared by some subspace clustering methods that will be discussed in the next section. Unfortunately most NLDR methods still rely heavily on euclidian distance metrics, which could lead to classification problems based on the nearest/farthest neighbor convergence.

4.3 Subspace Clustering

In the last decade subspace clustering has become the proposed solution to deal with the problems, created by high dimensionality, that render regular clustering algorithms useless. Instead of using the whole d-dimensional dataspace to determine clusters, subspace clustering algorithms first perform a search for relevant subspaces [14]. The motivation for performing a subspace search first is that we assume that the data space is a union of different arbitrary subspaces that were induced by the clusters in addition to noise. There are many different forms of subspaces. In text clustering, which uses the vector space model for representation, all data instances are contained in a space that is equipped with the Euclidian topology. A subspace search in this particular space has to take into account that any subset of the original data space found needs to be equipped with a topology that is element of its parent space to be called a valid subspace [12]. A few examples for a valid subspace in text clustering include:

- Discretization: The data space is partitioned into arbitrary pairwise disjoined intervals. The original data is now represented by the interval they reside in.
- Domain restriction: Instead of looking at the whole domain of a dimension only a smaller subset of the domain is considered as relevant.
- Feature selection: The dimensions of the data space are given a weight for their relevance in relation to a cluster.

The main problem in performing a search for an arbitrary subspace is that the number of possible valid subspace candidates is infinite. Subspace clustering algorithms therefore only concentrate on a bounded subset of all possible subspaces in the hope to approximate the true cluster subspaces enough to provide a valid clustering.

4.4 Axis Parallel Subspaces

The most common restriction that is used for the initial search is to only focus on axis parallel subspaces. We can define all axis-parallel subspaces as a special case of feature

weighting where the only possible weights that can be used are either zero or one. The biggest advantage of applying the axis-parallel restriction on the search is that the number of subspaces that need to be searched through is dramatically reduced.
If we want to find a k-dimensional subspace in a d-dimensional dataset at maximum $\binom{d}{k}$ subspaces need to be examined to determine the relevant subspaces. If we need to search through all axis-parallel candidates of a d-dimensional subspace the number increases to

$$\sum_{k=1}^{d} \binom{d}{k} = 2^d - 1$$

which is computationally infeasible for data of high dimensionality. Most subspace clustering methods of this category use either a top-down or bottom-up heuristic to perform the subspace search efficiently [14].

4.4.1 Top-Down Approaches

Top down approaches start by assigning each data point to a cluster by either random selection or with the use of a regular clustering algorithm. In the first step of these algorithms each dimension of the feature space is weighted the same. In the next step the algorithm assigns each dimension a weight for each cluster. The algorithm then iterates by alternating between clustering and weight generation until either an iteration threshold is reached or the cluster assignment has converged.
PROCLUS, the first top-down clustering algorithm, finds its clusters by first generating a set of k medoids based on a sampling of the data [2]. A greedy algorithm selects medoids with the goal of them being far appart. For each medoid a subspace is chosen in which the average point/medoid distance is small compared to statistical expectation.
After the initialization PROCLUS refines its clustering iteratively by selecting a subset of medoids then replacing bad medoids with new randomly selected new ones and determines if clustering quality has improved. The measure for quality is attained by calculating the average distance between data points and the nearest medoid in the determined subspace.
Most top-down approaches operate under the locality assumption. It must be possible to create a cluster based on the local neighborhood of the cluster center in the complete data space. If this condition is not fulfilled a correct cluster assignment is not possible.
Newer algorithms like FINDIT and σ-Clusters have refined their subspace search by developing new locally sensitive distance measuring methods and can even find non-axis-parallel subspaces by allowing continuous feature weights [32].

4.4.2 Bottom-Up Approaches

Bottom-up algorithms utilize the downward monotony behavior of clusters in axis-parallel dimensions. It states that each cluster in a high dimensional space is also a cluster in

each axis-parallel subspace. The search of subspaces starts by finding clusters in one-dimensional subspaces using regular clustering methods. These subspaces are then used to construct subspaces of a higher dimensionality using a APRIORI style search [4]. It looks for an overlap of lower dimensional clusters in a higher dimension. If an overlap exists a new higher dimensional cluster, the cut of both lower dimensional clusters, is found and the subspace is added to the list of relevant subspaces.
Algorithms of this category find all subspaces that clusters exists in. They create a hierarchical clustering based on dimensionality. The clusters they produce overlap, which means that one instance can be assigned to more than one cluster. One of the first bottom-up clustering algorithms was CLIQUE [5], which will be described in detail later in this paper.

4.5 Biclustering

Biclustering is based on finding certain patterns in the data matrix. The first biclustering algorithm was developed by Mirkin in 1996. A cluster of a biclustering algorithm is a submatrix of the initial data matrix which satisfies a set of predetermined conditions. These submatrices may overlap. In 2008 the similarity measure χ-Sim was introduced [10], which calculates the similarity between objects and their features. Utilizing this measure it is possible to use regular clustering methods to perform biclustering, because the distances correspond directly to biclustering conditions. A few examples for conditions in biclustering are:

Constant values: Each parameter of a cluster submatrix has to be the same. If this condition is applied to a text clustering data matrix that uses the boolean vector space model each submatrix returned is the document support of a frequent term set. The columns of the cluster correspond to the terms of the term set while the rows correspond to the documents supporting the term set.

Constant values in rows/columns: Every parameter in a row/column needs to be the same. If a submatrix under the constant column condition is projected into the subspace determined by its columns all points corresponding to the selected rows will be the same in the projection. If the condition is relaxed by allowing all values in a column to be in a ϵ-region it enables biclustering to be used to determine dense regions of a data set in every subspace.

Coherent values: The coherent value condition can be used to determine points that are positively correlated in certain subspaces. If v_{ij} is the value of the parameter in the i-th row and j-th column the condition that needs to be satisfied for a coherent value is

$$v_{ij} = c + r_i + c_j.$$

c is a constant that deals with any offset the submatrix could have. r_i are adjustment constants for each row and c_j adjusts each column.

4.6 Correlation Clustering

Axis-parallel subspaces have many advantages in text clustering. If we find a cluster using this method we can easily generate a set of words associated with it by simply using the domain of the subspace it resides in. The great disadvantage of these methods however is that it only takes the associated words into account. If the data for clustering is too sparse to find any meaningful dense regions based on the term sets a quality cluster assignment is not possible. A proposed solution is correlation clustering. It generalizes the axis-parallel subspace model by allowing arbitrarily oriented, affine subspaces.
The most common implementations of correlation clustering use principal component analysis to determine the subspace clusters reside in. ORCLUS expands the axis-parallel algorithm PROCLUS by applying PCA to all points assigned to a medoid [3]. The eigenvectors corresponding to the smallest eigenvalues are calculated. If the points belonging to the medoid are projected in this subspace their average distance to the medoid will be minimal because the system defined by the smallest eigenvectors indicates the subspace in which the points are most dense in.

Chapter 5

Implemented Clustering Algorithms

5.1 K-Subspace

K-Subspace [6] was developed as an extension to the well known k-Means[18] algorithm. Its purpose is to model arbitrarily oriented subspaces and to assign data points to clusters, that reside in them. Two of the subspaces that are modelled are a line and a plane. In addition to these two subspace models, that were first presented in the k-Plane algorithm [22], k-Subspace also provides a more advanced model for spherical clusters.
The algorithm works in two phases in each iteration step. The algorithm terminates after a user-selected threshold of iterations is reached.

Cluster assignment phase Each data point x out of the data set is assigned to one of k (an input parameter) clusters C_i. The decision is based on Euclidian proximity

$$i = \arg\min_{1 \leq i < k} dist(x, C_k)$$

in the initial iteration of the algorithm the data points are assigned randomly to calculate the first clustering models.

Model selection phase After each assignment of data points the algorithm calculates the parameters for all three cluster models for all K clusters. The decision for the cluster model that will be used for the next cluster assignment phase is made by selecting the model with the smallest model dispersion. The model dispersion in k-Subspace is defined as

$$Dispersion = \sum dist(x_k, C_k)$$

the sum of distances of the assigned data points.

5.1.1 Cluster Models

Line Shaped Clusters A line shaped cluster in k-Subspace is represented by two parameters a point c_i and normalized directional vector a_i. The distance between a data point x and the cluster C_i is defined as the squared Euclidian distance of its perpendicular component in relation to the line:

$$dist(x, C_i) = \|x^{\perp}\|^2$$

The perpendicular component $x^{\perp}$ is calculated by first attaining the scalar parallel component $x^{\parallel}$ e.g. the stretched normalized directional vector to make it line up perpendicular to x

$$x^{\parallel} = a_i[a_i^T(x - c_i)].$$

The perpendicular component is aquired by simple vector substraction

$$x^{\perp} = (x - c_i) - x^{\parallel}.$$

Let $x_i \in C_i$ be a data point assigned to the cluster C_i. Given a cluster assignment C_i the point parameter c_i is computed as the centroid of all points in C_i

$$c_k = \frac{1}{|C_i|} \sum_{x_i \in C_i} x_i$$

and the directional component a_i as the first normalized principal component of all x_i.

Plane Shaped Clusters Plane and Hyperplane shaped clusters are parameterised analogous to line shaped clusters. The model uses a point parameter c_i and two directional vectors a_i^1, a_i^2, for hyperplanes additional directional vectors a_i^j ($j <$Dimensionality of the data space) are added.
The perpendicular distance $\|x^{\perp}\|$ is calculated as

$$dist(x, C_i) = \|x - c_i - \sum \alpha_j a_k^j\|$$

with α being

$$\alpha_j = (x - c_i)^T a_i^j$$

. The calculation of the parameters given a cluster assignment is also analogous to line shaped clusters. The point c_i is centroid of all $x \in C_i$ while the directional parameters are the first two or more, depending on the dimensionality of the hyperplane, principal components of all $x \in C_i$.

Sphere Shaped Clusters The easiest approach to generate sphere shaped cluster would be to adapt the centroid based model, which k-Means uses for its clustering. K-Subspace does however model an explicit sphere shaped cluster region. In this case the distance between a data point and the cluster is not the distance between the centroid and the point. Instead the measured distance is the squared Euclidian distance between the hull of the sphere region of the cluster and the data point.

$$dist(x, C_i) = max(0, \|x - c_i\|^2 - \eta\sigma^2)$$

The spherical model requires the input parameters η and σ. If a data point is located inside the sphere defined by the parameter c_i with the radius $\sqrt{\eta}\sigma$ the distance is set at zero.
The model parameter c_i given a cluster assignment C_i is calculated by minimizing the dispersion of all assigned data points.

$$\min_{c_i} \sum_{x \in C_i} max(0, \|x - c_i\|^2 - \eta\sigma^2)$$

The solution to this minimization problem is

$$\frac{1}{1-\eta}\left(\frac{\sum_{x \in C_i^>} x}{|C_i^>|} - \eta \frac{\sum_{x \in C_i} x}{|C_i|}\right)$$

with $C_i^>$ being the set of all data points asigned to the cluster C_i that are outside of the sphere's radius.

5.2 CLIQUE

CLIQUE [5], one of the first subspace clustering algorithms, combines density based and grid based clustering. The algorithm first identifies all relevant subspaces and then proceeds to find all clusters in each one of them. The subspaces that are found by CLIQUE are axis parallel. A bottom up approach is used to grow the dimensions of possible subspaces candidates that may contain clusters. The algorithm operates in three steps.

1. **Subspace identification:** Using the data points that are provided by the example set, one dimensional subspaces are created. The previously created (k-1)-dimensional subspaces are used to find the next set of k-dimensional subspaces until no new relevant subspaces are found.
2. **Cluster identification:** In each subspace that was previously found a search for clusters is performed.
3. **Cluster description generation:** For every cluster a cover is calculated to give the cluster a comprehensible description.

5.2.1 Subspace identification

The input of CLIQUE consists of n d-dimensional points $X = \{X_1, X_2, \ldots, X_N\}$ with $X_i = (x_{i1}, x_{i2}, \ldots, x_{id})$. The attributes of the data space are represented by the set $T = \{t_1, t_2, \ldots, t_d\}$, the vocabulary of the vector space model. They span the feature space $S \subseteq \mathbb{R}^d$ all points reside in. v_{ij} stands for the value the i-th data point takes in the j-th dimension corresponding to attribute/term t_j.
CLIQUE starts by turning the continuous feature space into a discrete subspace. Each dimension is partitioned into ξ intervals of equal length. After this discretization the new feature subspace consists of ξ^d units. A unit U is defined by the intervals $\{u_1, u_2, \ldots, u_d\}$ with u_i being a right open interval $[l_i, r_i[$. If a point has the value $l_j \leq x_{ij} < r_j$ in all its dimensions it is added to the support of the unit U. The function $\mathit{support}(U)$ returns the number of all points that are contained within the unit U.
An axis parallel subspace R_I of the original feature space S is defined as

$$R_I = \{ \bigtimes_{i \in I} t_i | I \subset \{1, 2, \ldots, d\} \}$$

a simple selection of terms from T. A unit in an axis parallel subspace is defined as an intersection of all intervals of the attributes that are contained in the subspace. A point is added to the unit's support if all values of the attributes contained in the subspace lie within the bounds of the unit's intervals.
CLIQUE conducts its search for relevant subspaces in a bottom up fashion. It starts by initializing all one dimensional subspaces $R_{\{1\}}, R_{\{1\}}, \ldots, R_{\{d\}}$ and pruning them for relevancy. A subspace is called relevant if it contains at least one dense unit. A unit U is called dense if $\mathit{support}(U)/n > \tau$, τ is the user specified density threshold parameter. The set containing all dense one-dimensional subspaces is called RD_1
In the next iterations the (k-1)-dimensional dense subspaces are used to generate the candidates for the k-dimensional subspaces. CLIQUE utilizes the monotonicity of points in a cluster to generate the correct candidates.

> **Monotonicity** If a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any (k-1)-dimensional projections of this space.[5]

The step to generate the k-dimensional subspaces gets the set RD_{k-1} as an input. The subspaces in RD_{k-1} are sorted lexicographically by their dimensions. The set is joined with itself. Two subspaces R_I and $R_J \in RD_{k-1}$ with $|I| = |J| = k - 1$ are joined into a k-dimensional subspace if they meet the following conditions [4].

1. In the ordered sets of subspace attributes I and J the first k-2 elements need to be identical

2. The last element of each subspace needs to be different to ensure a k-dimensional subspace after the joining process

3. Let i and j be the last elements of I and J: $i < j$

After the subspace join the units of the new k-dimensional subspace are created. The unit join process performs the same checks but uses the units' intervals to compare. The new k-dimensional units need to be pruned again for their density because the monotonicity of clusters infers that density might be lost going from a lower into a higher dimension. If the newly generated k-dimensional subspace includes at least one dense unit it is added to the set D_k. The process is repeated until no new k-dimensional subspace containing at least one dense unit is found.

5.2.2 Cluster identification

The first step of CLIQUE created a superset $RD = \{RD_1, \ldots, RD_k\}$ of subspaces that contain dense units. A cluster is defined by CLIQUE as the maximum set of connected units in a subspace. Two units U_1 and U_2 are connected in a subspace R_I if

$$\exists i \in I : \{l_{i1} = r_{i2} \vee r_{i1} = l_{i2} | [l_{i1}, r_{i1}[\in, U_1 [l_{i2}, r_{i2}[\in U_2\}$$

If we transform the dense units into nodes and the statement of the connection requirement into a vertice we can reduce the problem of finding the maximal set of connected dense units into the connected component problem of undirected graphs.
To find all clusters CLIQUE now has to perform a depth first search for all units that have not already been assigned to a cluster. Each unit/node that is traversed in the DFS is put into the same cluster. The cluster assignment stops when all dense units of a subspace are assigned. The cluster assignment is performed for all subspaces in D.

5.2.3 Cluster description generation

The goal of the final step is to generate the minimum amount of rectangular regions for each subspace so that the clusters that were found in the previous step in the cluster set C are covered. The set of generated rectangles is the cluster description. The minimal description of a cluster would be a set of rectangles that only contain the dense connected unit of the cluster itself. Unfortunately it is not computationally feasible to generate the minimum set of rectangles for the minimal cluster description.
Instead of calculating the minimal cover we use a greedy algorithm to determine the maximum regions that the cluster covers.
The algorithm starts by randomly selecting a unit of the cluster that has not yet been covered and propagates a new region into both directions of each dimension until it does not find a connected dense unit. The right bound of the most right unit and the left bound

of the most left unit is selected as the edge of the cover region for each dimension. The algorithm terminates when all dense units of the cluster are covered by the descrition. The final output of CLIQUE is the set of all cluster descriptions for all subspaces.

5.3 Frequent Term-Based Text Clustering

Frequent term based clustering [8] was specifically developed to deal with the challenges that are posed by text clustering. It deals with the high dimensionality by constructing its clusters bottom up, using frequent term sets as the domain for its subspaces. It does not require the number of clusters as an input. The output of FTC is a clustering, in which each document was assigned to excactly one of the k found non overlapping clusters.
The input for FTC is a set of word n Vectors $X = \{X_1, X_2, \ldots, X_n\}$ that represent the documents $D = \{D_1, D_2, \ldots, D_n\}$ in the format of the boolean VSM. The vocabulary set T contains all words that are contained in the documents of D, |T| is the dimensionality of the word vectors. The algorithm starts by determining all relevant frequent term sets F. It does this by first creating the single item term set $F^1 = \{F_1, \ldots, F_{|T|}\}$ from the dictionary T. Each word is given its own term set.
After this the support for each term set is calculated. A document D_i from D supports the term set F_i if the word vector X_i has a value other than 0 in the dimension that represents the word in F_i.
With a single iteration over the database the support for every term set in F^1 is calculated. The results are saved in a two-tuple $Sup_i = (F_i, S_i)$ where S_i contains all the documents that support the term set F_i.
In addition to the support tuples for the term sets a second support set is created. This set keeps track of how many term sets are supported by a particular document. If R is the set of all relevant term sets the integer $f_j = |\{F_i \in R | F_i \subseteq D_j\}|$ tells us how often a document was assigned to a different relevant term set.
The function $support(Sup_i)$ calculates the document support of a term set in relation to the size of the database ($|S_i|/n$). If $support(Sup_i) < \tau$, the user-specified minimum document support, the term set F_i is removed from F^1. The now pruned term sets are used to create the term sets of a higher cardinality similar to the subspace joining in CLIQUE.
To join two term set the tuple that represents the set support needs to be joined, too. The operation to join two term set tuples is:

$$Sup_i \cup Sup_j = (F_i \cup F_j, S_i \cap S_j)$$

It is obvious why the restrictions used in the CLIQUE algorithm for the join make sense if $|F_i| = |F_j|$ and $|F_i| - 1$ items in both set are identical then $|F_i \cup F_j| = |F_i| + 1$ and the newly found term set's cardinality grows by one in each pass over the previously generated set F^{k-1} in the k-th step. After each join the newly generated term sets need to be pruned

again in relation to the minimum support parameter because $|S_i \cap S_j| \leq min(|S_i|, |S_j|)$. The process of growing the frequent term sets is repeated until no new term set that can satisfy $support(Sup_i) < minS$ is found.

FTC defines the clustering description as a selection of terms $CD \subseteq I = \{1, 2, \ldots, |R|\}$ that satisfies the condition $|\bigcup_{i \in CD} S_i| = |D|$ which means that every document needs to be contained in at least one of the term sets that were selected by the cluster description. FTC also includes the empty term set as relevant, it ensures that all documents will belong to at least one relevant term set.

The cluster selection of FTC is guided by the goal to minimize the cluster overlap. When a term set is selected as a cluster C_i the standard overlap with other possible clusters (term sets) is defined as:

$$Overlap(C_i) = \frac{\sum_{D_j \in C_i} f_j - 1}{|C_i|}$$

The set C_i contains all the documents from S_i that have not already been selected by the cluster description. We could calculate the overlap for each term set as a cluster candidate and pick the one with the smallest overlap as the next cluster until a cluster description is reached.

Unfortunately the monotonicity of frequent term sets, analogous to the monotonicity of clusters, means that a document in a term set with the cardinality m supports at least m-1 other smaller term sets. This means that FTC using the standard overlap would predominantly select term sets with a low cardinality.

To get around this problem FTC uses the entropy overlap [26] to determine the next cluster candidate.

$$EO(C_i) = \sum_{D_j \in C_i} -\frac{1}{f_j} \cdot ln(\frac{1}{f_j})$$

The cluster selection now consists of the following steps:

1. Calculate the entropy overlap for all remaining term sets in R
2. Remove the term set with the smallest entropy overlap out of R and select it as C_i
3. Add C_i to the cluster description and mark all documents in C_i as selected
4. Recalculate the f_j for all remaining documents

This is done until a valid cluster description is found or all terms out of R are selected. The algorithm returns a flat clustering that covers the whole database and assigns each document to only one cluster.

Chapter 6

Implementation

6.1 Rapidminer and EJML

In the process of classification clustering is only one step out of many. Many tasks like pre-processing, data collection, formating and result visualisation are as important as the clustering itself to achieve a comprehensible classification result.
RapidMiner provides a framework for data mining tasks running in a Java environment [24]. Rapidminer operates under a modular paradigm, processes are built from a selection of operators, the building blocks of Rapidminer. Operators receive, process and send the transformed data to the next operator of the process sequence or to a result screen. The biggest advantage of RapidMiner is its expandability. If an operation that is not already part of the core operations of RapidMiner is required it is possible to create a new operator, that is fully integrated into RapidMiner.
For the integration of the subspace clustering algorithms an extension was created featuring an unique operator for each. In addition to new operators a subspace cluster input/output object was created to make it possible for RapidMiner to process subspace clusters.
RapidMiner already features a PCA operator in its core distribution but for memory efficiency the Efficient Java Matrix Library (EJML) [1] was chosen for the principal component generation in K-Subspace.

6.2 Sparse Data Structures

As explained in the chapter about vector space models, the dimensionality of the data is often very high because a document is represented in relation to the whole vocabulary of the corpus. A second interesting feature, which is a direct result of this high dimensionality, is data sparsity.
A data set is called sparse if more than half of all values in the data matrix are zero. It is easy to see that in the case of homograph discovery data will naturally be sparse, because

Regular Data Row

Att0	Att1	Att2	Att3	Att4	Att5	Att6	Att7	Att8	Att9	Att10	Att11
0	0	0	5	0	2	0	0	0	7	1	0

Sparse Data Row

Index	3	5	9	10
Value	5	2	7	1

Table 6.1: An comparison between a regular and a sparse data row

the texts that are used to create the vector space model are only document fragments e.g. the surrounding sentences of a matched homograph. While full texts still originate from many different domains providing them with a huge vocabulary they also are long enough to generate a significant overlap with other documents. Text fragments often contain less than fifty unique terms in a data row with more than ten thousand dimensions. The result of the density calculation of the homograph data matrix $V^{n \times m}$ using the measure

$$\rho(V) = \frac{|\{v_{ij} \neq 0\}|}{m \cdot n}$$

came in significantly lower than 1%.
Fortunately it is possible to use the sparsity of the data set to our advantage - to lower the computational and memory cost - in the implementation of the presented subspace clustering methods.

6.2.1 Rapidminer's Sparse Data Structure

Raw data in Rapidminer is stored in the ExampleTable. Its structure is very similar to the data matrix of the vector space model. The data is organized in data rows, each instance of the data set is given its own data row. An arbitrary number of these data rows is combined into the data table, each data row therefore must have the same number of columns.

To implement sparse data RapidMiner has extended the regular data row to store data in the compressed sparse row format. A single array, with the length of the full dimensionality d, that stores all values of the data row, is replaced by two shorter arrays. The first array stores all indeces, for which the values of the data row are different from zero. The second array stores the actual values of the indexed columns. The example in 6.1 illustrates how the sparse data row reduces the memory footprint for any data set with $\rho(V) \leq 50\%$. The price for this reduction in memory comes at the price of increased computational complexity for single value retrieval which now lies in $\mathcal{O}(logn)$ utilizing binary search.
It is essential for data mining tasks to put the raw data into context. For this task the ExampleSet class is available. Operators in RapidMiner almost exclusively operate on this layer of data representation. Each ExampleSet has a parent ExampleTable, a set of Attributes and Examples. Attributes are used to give the raw data that is stored in a

column of the ExampleTable context. There are two different types of attributes: Regular attributes, which indicate that the values in the column they refer to are features of the data set. In the vector space model all values of the data matrix are in the regular attribute space. The second type of attribute is the special attribute, which is used for additional meta information like IDs, predictions, cluster labelling or the original text the vector was created from. In clustering processes those special attributes are ignored.
The most important property of the ExampleSet is that it only references the ExampleTable. Each Attribute of the ExampleSet includes the column number of the ExampleTable it references, similarly each Example includes a reference to the DataRow of the ExampleTable it references. These references do not need to be one to one. Figure 6.1 shows that the n-th Attribute of the ExampleSet does not neccesarily correspond with the n-th column of the ExampleTable. The same goes for Examples.
If we want to retrieve a value of an Attribute-Example pair, RapidMiner gets the DataRow the Example is referencing out of the parent ExampleTable, retrieves the column number of the Attribute and returns the value of the column [23]. This method of data retrieval presents a challenge for the implementation of sparse methods for algorithms that predominantly operate on ExampleSets:

- The order of attributes does not correspond with the order of the DataRow $\Longrightarrow$ It is not possible to simply use the indeces of the sparse DataRow.
- Regular and special Attributes are stored in the same DataRow $\Longrightarrow$ A method to discriminate between indeces for regular or special attributes is needed.

The solution to these problems is to create a mapping between the the regular Attributes' ordering and the columns of the ExampleTable. Using FastExample2SparseTransform we are able to now retrieve the indeces of all regular attributes that are different from zero, for each data row, in the correct order, for operations on the ExampleSet [24].

6.2.2 Sparse CLIQUE

In the process of testing CLIQUE a problematic behavior in dealing with sparse data materialized. The algorithm always created as many dense one-dimensional subspaces as there were dimensions in the data matrix regardless of how the density threshold parameter was set. As a result of this, a greater than expected number of higher dimensional subspaces was created, in some instances approaching the maximum of $2^d - 1$ possible subspaces.
The cause of this unwanted behavior was identified in the initialization of the one-dimensional subspaces. CLIQUE, in its standard form, treats each instance of the vector space model as a point in the data space. If the data is sparse CLIQUE will find a dense region in the interval $[a, b] : a < 0 < b$. These "false" dense regions and their subspaces heavily outnumber the dense regions created by non zero instances. Furthermore false clusters

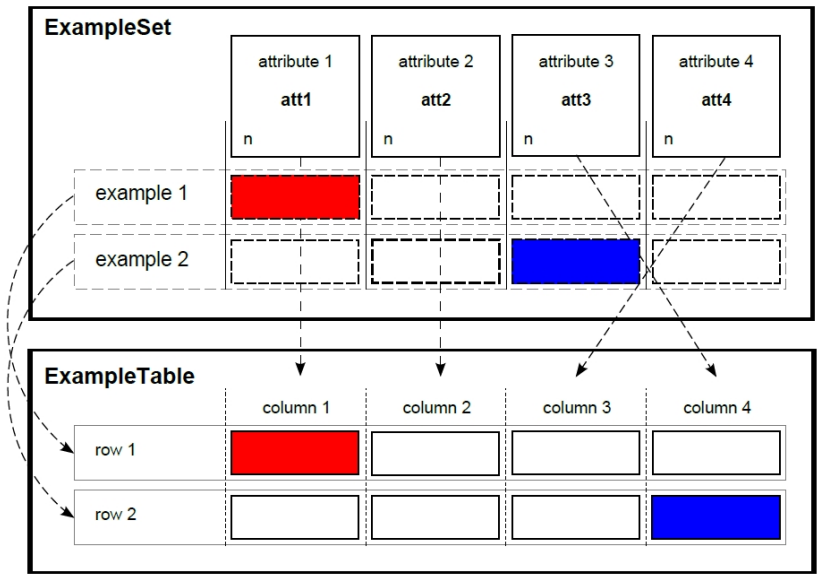

Figure 6.1: An ExampleSet referencing an ExampleTable [23]

including a non zero region and a zero region can be created.
To eliminate this unwanted behavior this implementation of CLIQUE gives the user the choice to perform a sparsity check on the data matrix. If the matrix is deemed too sparse to apply the regular CLIQUE algorithm a slightly altered procedure for the initialization that ignores all zero values is used.

6.2.3 Mean and Distance Calculation

Require: Sparse Points $X = (I[k], V[k])$ and dimensionality d
  $SUM[i] = 0$ for all $i = 1, 2, \cdots, d$
  for all $x \in X$ **do**
    for $j = 0 \cdots k$ **do**
      $SUM[I[j]] \leftarrow SUM[I[j]] + V[j]$
    end for
    for all $SUM[i]$ **do**
      $SUM[i] \leftarrow SUM[i]/|X|$
    end for
  end for
  return SUM

Algorithm 6.1: Sparse Mean Calculation

Mean calculation is an essential part in the calculation of the model parameters in K-Subspace. For the line and plane model one centroid needs to be calculated using the mean

of all cluster points and the sphere model requires two mean calculations for its parameter. The sparse row format gives us the opportunity to increase the speed of mean calculation. The mean is calculated seperately for each dimension. If a point that is added to the mean has a zero in a particular dimension it will not influence the mean in it. The sparse index array of the row model therefore provides us with all dimensions that will influence the mean if we add this point. Using this the time for mean calculation will be proportional to the sparsity of the data. The algorithm 6.1 illustrates how a faster mean calculation is achieved by utilizing the array iteration advantange of the sparse row format.

6.2.4 Initialization of Term Sets

In Frequent Term-Based Text Clustering we construct our relevant frequent term sets similar to CLIQUE's subspace creation. Because of that we need to initialize the term sets that only contain one item. The naive approach would be to create a single item term set out of an attribute (the attributes of a VSM are terms) and then to iterate over all examples adding documents that are found to contain the term, e.g. having a value greater than zero, to the term set's support.
Unfortunatelly RapidMiner does not support the sparse column format, which would let us directly calculate the document support by retrieving the index array size. Instead the sparse row format is utilized to only iterate over all non zero attributes for each example. First a single item term set for each attribute is initialized. Then for each example the indices array is retrieved. The example is then added as document support of each term set that corresponds to a non default index.

6.3 Notable Implementation Details

Most of the implementation work was compromised of simply translating the the operations of the algorithm into Java code. There were, however, some instances in which additional implementation work was required to make the algorithms work in the RapidMiner environment.
The next sections detail the implementation of a new cluster model object, that deals with the challenges presented by subspace clustering tasks. In addition, a few important implementation decisions concerning the inner operations of CLIQUE and FTC are expanded on, including the one-dimensional subspace initialization and parts of the self join process of the APRIORY search.

6.3.1 The Subspace Cluster Model

RapidMiner already supports a broad collection of clustering algorithms. The standart input for them is an ExampleSet and a ClusterModel object is returned as the result in

addition to a labelled ExampleSet. In the implementation of subspace clustering algorithms a few problems surfaced based on the structure of the ClusterModel class. ClusterModel can only deal with flat clusterings, which makes it useless for CLIQUE. Rapidminer also includes an extension of the cluster model for hierarchical clusterings but this model is used for dendrogram visualisation and cannot deal with a subspace hierarchy created by the APRIORI search of CLIQUE. For an output of a hierarchical cluster model it also has to be flattened in advance.
Another disadvantage of the RapidMiner ClusterModel is that the cluster assignment has to be set as a whole instead of cluster by cluster, which would lead to complications in FTC where the next cluster assignment is dependent on the previous assignments.
SubspaceClusterModel provides more flexibility and better organization to resolve those problems. Clusters can be added one by one using the AddCluster method. In addition to the clusters itself additional information about the subspace is stored in the form of a SubspaceModel class. A SubspaceModel can return the description of itself and project an example into itself.
Internally, clusters are organized based on the subspace they reside in instead of single list. It is possible to get a cluster assignment in a per-subspace fashion. For CLIQUE the example set can be labbeled with all clusters. For each subspace a unique attribute is created that contains the cluster assignment for the one subspace.
Visualization is done textually. All subspaces and all clusters residing in them are listed. Additional information for each cluster and subspace is given, like cluster size and subspace dimensionality.
Unfortunately RapidMiner has no evaluators that can deal with multi label data. Therefore all evaluation measures that will be presented in the next chapter needed to be implemented again to work with the SubspaceClusterModel.

6.3.2 Initialization of a One-Dimensional Subspace

The first step in the identification of dense subspaces is the construction of one dimensional subspaces. These subspaces need to satisfy multiple conditions. It needs to be possible to identify the dimensions this axis parallel subspace covers to create a comprehensible cluster description and to construct higher dimensional subspaces. For the representation of the dimensional cover of the subspace the class BitSet was chosen. The BitSet can be thought of as a dynamic binary vector. When a one dimensional subspace is created out of a regular attribute the algorithm retrieves a unique constant integer, the attribute ID, and sets the bit of the BitSet at this position (Algorithm 6.2 line 3).
In the next step the units of the subspace are initialized. They represent the partition of the subspace. In the implementation of CLIQUE they are represented by the subspace they reside in and a HashMap that maps each dimension of this subspace to an interval. Each

unit also possesses a support set in which all IDs of instances (documents) that support the unit are saved. Lines 6-10 in 6.2 explain how the initial intervals are calculated based on the user parameters and range of values of the dimension and the assignment of documents to the support of units.

After the assignment of all instances of the data set is done the subspace's units are

Require: Dimension ID $d - ID$, values of dimension d: V, gridnumber ξ
Create dimension set $D = \emptyset$
2: $D \leftarrow D \cup \{d - ID\}$
$MIN \leftarrow minimum(V)$
4: $MAX \leftarrow maximum(V)$
for $i = 0 \ldots \xi$ **do**
6: Create Unit u_i with interval $[MIN + \frac{MAX-MIN}{\xi} \cdot i, MIN + \frac{MAX-MIN}{\xi} \cdot (i+1)[$ in dimension d
end for
8: **for all** $v_i \in V$ **do**
$j \leftarrow \lfloor (v - MIN) \cdot \frac{\xi}{MAX-MIN} \rfloor$
10: Add point belonging to v_i to support of u_j
end for

Algorithm 6.2: Subspace Initialization

pruned against the density threshold τ. If one or more units remain in the subspace after the pruning the subspace is included in the set of one dimensional dense subspaces.

6.3.3 Construction of Subspaces/Frequent Term Sets

For the construction of the k-dimensional subspaces out of the (k-1)-dimensional the previously stated conditions need to be met [4]. In the implementation subspaces are put in a list that corresponds to their dimensionality. If we want to construct the list of k-dimensional subspaces we first retrieve the (k-1)-dimensional list and sort all subspaces in their lexicographic order. We achieve this order by comparing the pair BitSets that indicate the dimensions of the subspaces against each other. We iterate both BitSets of the compared subspaces and compare the position of the next set bit with each other. If the positions are equal, we continue to iterate, if we find different positions, we stipulate that the subspace that has the next set bit at a later position also lies lexicographically later than the other subspace.

With the subspaces ordered we can start the self join process. First the subspace at the beginning of the subspace list is taken and attempts to join with all subspaces remaining in the list. A join of two subspaces will only take place if algorithm 6.3 returns TRUE. The

Require: Dimension sets D_1, D_2
$D_{check} \leftarrow D_1 \oplus D_2$
2: **if** $|D_{check}| \neq 2$ **then**
return FALSE
4: **end if**
$l_1 = D_1 \cap D_{check}, l_2 = D_2 \cap D_{check}$
6: **if** $l_1 \neq max(D_1) \vee l_2 \neq max(D_2) \vee l_1 \geq l_2$ **then**
return FALSE
8: **end if**
return TRUE

Algorithm 6.3: Join conditions check

first two lines of 6.3 perform the check if the two subspaces only differ in one dimension. An XOR ($\oplus$) operation is performed with the BitSets of both subspaces. This is possible because the two dimensional sets D_1 and D_2 have the same cardinality. If a new set from the exclusive disjunction of both is created it will contain all the dimensions both differ in. A cardinality of two then means that one out of each set is different. This is a necessary condition to satisfy the statement that (k-1)-dimensional subspaces need to share the first k-2 dimensions for a join. Line five and six of 6.3 perform a check for the sufficient condition. In addition to checking that both BitSets only differ in one dimension we also make sure that the dimension they differ in is the k-1-th one. The check $\vee l_1 \geq l_2$ is added to satisfy the lexicographic order of the subspaces. After the join the resulting subspace is put in the list of k-dimensional subspace candidates.
The subspace that was checked against the remaining subspaces in the list is then removed from the list. The procedure is repeated until the list of subspaces is empty.
The k-dimensional subspace candidates are then pruned for density. In the actual join operation the dense units are also joined in a similar fashion. The same checks are performed using the intervals of each unit to make sure that only units that have the same first k-1 intervals are joined. The support for a newly joined unit is the conjunction of the support of the units it was joined from. All joined units' support is then checked against the density threshold and non dense units are removed from the subspace. The k-dimensional subspace list is then created from all joined subspaces that include more than one dense unit.

6.4 Extension Documentation

To install the plugin containing the presented algorithms the plugin jar file needs to be copied into the lib\plugin folder of the RapidMiner directory. For the plugin to operate

correctly, RapidMiner needs to run on Java-SE 1.7 and the EJML library 0.22 needs to be installed.
The subspace clustering operators all operate using the sparse data row format for data retrieval. A user error will be thrown if the exampleset containing the cluster data is not formated in the double sparse array format.

K-Subspace This operator creates a flat clustering based on the K-Subspace algorithm over all regular attributes in the example set.

I/O	port-name	Description
I	example set input	The exampleset containing the vector space model of the corpus for the clustering
O	cluster model	The flat clustering model of the exampleset
O	example set	The original exampleset with the cluster assignments added as an attribute

I/O Ports of *K-Subspace*

Parameter	Description
k	The number of clusters that K-Subspace needs to find
eta	The η parameter of the sphere model. Sphere radius $r = \sqrt{\sigma^2 \eta}$
sigma	The σ parameter of the sphere model. Sphere radius $r = \sqrt{\sigma^2 \eta}$
max optimization steps	The number of iterations of the model and cluster assignment
maximum order	The maximum dimensionality of the plane model. A one dimensional plane is equivalent to the line model
use sphere model	Toggle if the sphere model will be used for the cluster assignment. If off the algorithm performed is K-Plane

CLIQUE This operator performs the CLIQUE clustering operation for the input exampleset. Unfortunatelly RapidMiner does not support the output of regions in disjunct normal form as a valid cluster model. The output of this CLIQUE operator consists of all found clusters in the model each cluster contains the IDs of all instances that support it. Additionally an individual attribute for each found subspace is created. This attribute shows the flat cluster assignment for each subspace.

I/O	port-name	Description
I	example set input	The exampleset containing the vector space model of the corpus for the clustering
O	cluster model	The flat clustering model of the exampleset
O	example set	The original exampleset
O	example set	The original exampleset with the cluster assignments added as an attribute for each subspace an individual attribute is created

I/O Ports of *CLIQUE*

Parameter	Description
gridnumber	Indicates in how many segments of equal length each dimension needs to be partitioned. The partitioning will take place over the value range in each dimension.
threshold	The τ parameter of CLIQUE. A unit will be seen as dense if it is supported by more than this fraction of all instances of the dataset.
ignorezero	If set, the operator will ignore zero values in its initial assignment of instances into units in the one dimensional subspace creation.

FTC This operator performs the Frequent Term-Based Text Clustering algorithm for the input exampleset. A flat clustering is returned as the cluster model. An attribute of the cluster assignment is added. The name of the cluster assignment is the contents of the frequent term set it belongs to.

I/O	port-name	Description
I	example set input	The exampleset containing the vector space model of the corpus for the clustering
O	cluster model	The flat clustering model of the exampleset
O	example set	The original exampleset with the cluster assignments is added as an attribute

The only input parameter for FTC is the required **minimum support** for a relevant frequent term set. The parameter is defined for all values between zero and one and stands for the fraction of examples of the exampleset that need to be in a term set for it to be called relevant.

Chapter 7

Experimental Evaluation

In this chapter the previously presented subspace clustering methods will be evaluated on the task of correctly assigning occurences of homographs in different texts to their true meaning. First the data that will be used for the empirical tests will be presented. In the following section the neccessary pre-proccessing steps to transform the raw data into the vector space model will be explained. After a brief overview of the preformance measurements that were used for evaluation and comparison of the different clustering methods the results of the classification using CLIQUE, K-Subspace, and FTC will be listed and discussed.

7.1 Clustering Data

To perform the empirical evaluation a data set created for the task of homograph disambiguation will be utilized. The data was collected under the KOBRA initiative. The data collected in this data set originates from the DWDS core corpus. A query for a known homograph was performed, in this case the homograph in question was "'Leiter"' (senses for the German word "'Leiter"' include: leader, ladder). If a match was found the sentences containing the match as well as the neighboring sentences were saved as a document d of the data set D.

ID	Match	Texttype	Date	Sense	Source	Copyright
1	... Leiter...	Article	1900-12-31	Leader	Kafka, Franz...	OR7W

Table 7.1: The format of the initial data set used for the experiments

In addition to the matching sentences the data table also includes some metadata (Table 7.1) belonging to the text the sentences was extracted from. This data is not significant for the evaluation and is removed from the table.

The only important attribute of the original data other than the found match is the "Sense" label. It contains the meaning of the homograph that was determined manually by a human. The data set used contains 1993 manually labelled occurences of the homograph "'Leiter"'. The classes for the labels are:

- Ladder
- Electrical Conductor
- Leader
- Musical Scale
- Unknown Meaning

7.2 Preproccessing

After the homograph data has been imported it needs to be transformed into a VSM to calculate the clustering. The text proccessing extension of Rapidminer is used to create the VSM. Two different representations are created, binary term occurrence and TF-IDF. Both models are used to evaluate CLIQUE, the binary model is used exclusively for the evaluation of FTC and TF-IDF is the exclusive model for K-Subspace.
In the creation of the VSM the following steps have to be performed to achieve a good representation of the initial documents.

Tokenization and Filtering The first step in the creation of the VSM is to partition the continuous text of the original data into single terms, the tokens. The easiest way to achieve this tokenization is to simply parse the text and create a new token whenever a non alphabetical character is parsed. Unfortunately this method is prone to create tokens that are too small. Abbreviations containing periods, floating point numbers or other mathematical representations would either be cut into single letter tokens or ignored completely. To solve this problem a method that tokenizes a text into its linguistic fragments is used; it adds a set of linguistic and grammatical rules to the tokenization process such as: *A sentence never starts with a small letter.* In addition to the application of rules the tokenizer queries a dictionary with possibly ambiguous tokens to determine a correct partition.
After the text is tokenized a filter is applied that removes all tokens under a certain length. Some texts contain single letter words (indeces, floor numbers/letters, mathematical variables). Their removal avoids possible false similarities between documents. An example would be the token "'E"', which could stand for energy in a physics text while also marking the ground floor of a building (The letter E marks the ground floor in German buildings).

Stop Word Removal and Pruning In this step the most common words of the German language, the stop words, are removed. Stop words like "'a"' and "'the"' (in this case their German counterparts) have almost no value if we want to find discrimination criterions of different text classes, because they most likely appear within every document class. Figure 7.1 illustrates how the frequency of a word corresponds to its value in information retrieval. This relation was first described by H.P.Luhn [17]. In addition to the standard stop words the homograph term is also removed because it occurs in every document of the data set.

The second part of this step is the pruning of the least frequent terms. A term that

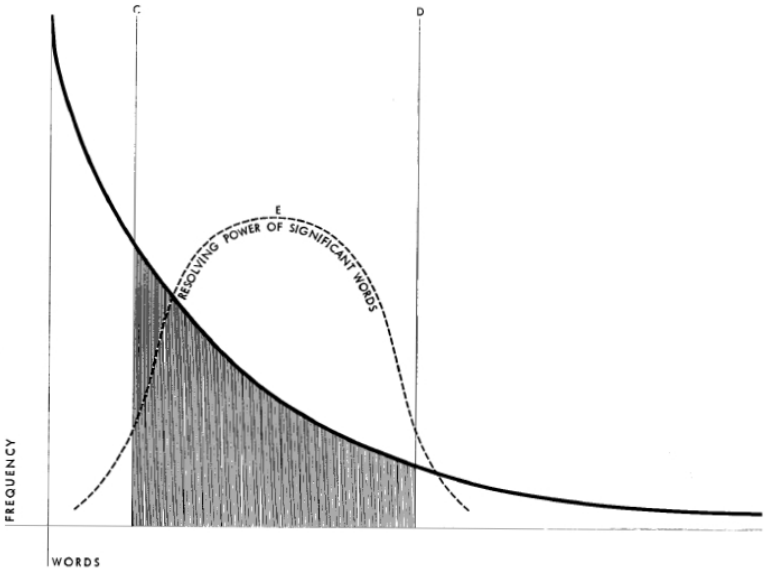

Figure 7.1: Word-frequency diagram. Words are arranged in order of frequency [17]

is supported by only one or two documents of the whole data set also has very little informational value as seen in 7.1. Another reason for this pruning procedure is that the dimensionality of the VSM is reduced dramatically even if the pruning threshold is very low. Furthermore two of our three algorithms (CLIQUE, FTC) already use an internal pruning procedure with thresholds that are higher than the pruning threshold applied at the VSM creation.

In the experiments that were performed for this evaluation the pruning threshold was set at the absolute value of 3, which is lower than the cardinality of the smallest class of the data set.

Stemming In the last preprocessing step all terms are reduced to their stem to further reduce the dimensionality of the VSM without using any significant informational value. In the actual data generation this step is performed between stop word removal and pruning to ensure that the least amount of terms are pruned out of the VSM. Terms that fell under the threshold before stemming could exceed it by merging with its different conjugations.

7.3 Evaluation Measures

Evaluation measures of a clustering can be seperated into two categories. The measurements of the first category quantify the characteristic that objects within a correct cluster need to be more similar to each other than objects of another cluster. The measurements are called internal because no knowledge of the true classification, no information external to the clustering itself, is neccessary to evaluate clusters using these methods. The most general form of an internal measure is a simple quantification of the similarities of all objects within a cluster for all clusters. This measurement is called the intra cluster similarity. The other internal measurement compares how different clusters are to each other and generates inter cluster similarity.
In our experiments internal measurements will not be used for evaluation because they rely on distance metrics, which fail in high dimensionality. It would be possible to compare clusters within a subspace using internal measurements but it would be impossible to apply an internal measurement over the whole clustering because inter measurements cannnot be applied between different subspaces and intra measurments cannot be compared in a meaningful way.
To evaluate CLIQUE, K-Subspace and FTC a selection of external measurements was made. The labelled data makes it possible to quantify the degree of classification quality of the clustering produced by the three algorithms. It is, however, neccessary to create a unique set of measurements for each algorithm because they each produce a different type of clustering. K-Subspace produces a flat, hard clustering that covers all instances of the data set with as many clusters as true classification classes. FTC also produces a flat, hard clustering but the number of clusters is not fixed and will often exceed the number of actual classes. FTC produces a cluster for the empty term set, which can be interpreted as an assignment to noise. CLIQUE produces a clustering for each found subspace. If we want to evaluate the whole clustering produced by CLIQUE we need measurements that can deal with overlapping, hierarchical clusterings.

Matching Matrix In a matching matrix each column represents the instances in a predicted class, while each row represents the instances in an actual class. The diagonal of the matrix shows how many true positives are in the classification of the clustering [29]. A matching matrix requires that as many predicted classes (clusters) are produced as the number actual classes, which means that it will only be used to evaluate K-Subspace.
Utilizing the matching matrix a number of essential performance measurements can be calculated.
Precision quantifies the degree of how many documents in a cluster actually are classified

correctly after the cluster has generated a prediction label. The precision for a cluster C_i that for the class K_j is defined as

$$precision_{K_j}(C_i) = p(K_j|C_i) = \frac{|C_i \cap K_j|}{|C_i|}.$$

Recall measures how many documents that belong to a certain class were extracted by a cluster

$$recall_{K_j}(C_i) = p(C_i|K_j) = \frac{|C_i \cap K_j|}{|K_j|}.$$

Precision and recall can be calculated for any cluster in relation to a class but the matching matrix makes it possible to calculate the overall accuracy of the clustering by assigning a fixed predicted class to a cluster. The overall accuracy based on the matching matrix where each cluster was assigned a class out of K is calculated as

$$accuracy(C) = \sum_{K_j \in K} precision_{K_j}(C_{K_j}) \cdot recall_{K_j}(C_{K_j})$$

Purity A more general form of determining the classification performance of a clustering is the purity measure. Its only requirement is that the produced clustering is flat. Each cluster is assigned to the actual class that is most frequent within it. The number of true positives of each cluster acording to their labeled class is aggregated and then divided by the total number of instances n.

$$\frac{1}{n} \sum_{C_i \in C} \max_{K_j \in K} |C_i \cap K_j|.$$

All presented measurements until now have significant disadvantages when it comes to evaluate the classification of an unbalanced data set. If there was a two class data set that had 100 instances with 90 belonging to class A and 10 belonging to class B a classification which would put all objects into a single predicted class would still receive an accuracy and purity measure of 90% . The simplest way to create a purity measurement that considers unbalanced data sets is to add the additional condition that the class selection for clusters needs to give more weight to the recall. It is added as a maximization constraint to the original purity measure; each cluster is now assigned to the class producing the largest recall for the cluster.

Entropy Until now performance of a cluster was measured by assigning a cluster to one specific class. The ultimate goal of any classification algorithm is the elimination of uncertainty. To evaluate its performance the initial number of actual classes needs to be taken into account because corretly assigning an instance to one of ten classes is likely a harder task than the assignment to one out of two. Entropy provides this by measuring

the remaining uncertainty after the clustering for each cluster over all actual classes [26]. For a single cluster the entropy is computed as

$$H(C_i) = \sum_{K_j \in K} -p(K_j|C_i) \cdot log(p(K_j|C_i)).$$

The entropy is lowest (zero) if the instances located in a cluster have no uncertainty of belonging to a class, which is equivalent to the cluster only featuring one class of instances. It is highest ($ln(|K|)$) if the objects are equally distributed over all classes.
The entropy for the whole clustering is a simple weighted aggregation of all single cluster entropies.

$$H(C) = \sum_{C_i \in C} \frac{|C_i|}{n} \cdot H(C_i)$$

F-Measure The clustering provided by CLIQUE requires a measurement that can deal with cluster overlap and hierarchical structures. While most presented measurements work for the evaluation of a single subspace of CLIQUE the whole clustering cannot be evaluated, because the normalization of the measurement aggregate produces false results for non-flat clusterings.
F-Measure [25] produces a classification performance that considers both precision and recall

$$F_\beta = (1 + \beta^2) \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

for a cluster. The most common F-Measure is F_1 which calculates the harmonic mean between precision and recall. For evaluation F_1 and F_2, which weights recall higher will be used.
What makes F-Measure so valuable for the evaluation of CLIQUE is that its aggregation is performed over the actual classes in contrast to clusters [30]. The aggregation for any clustering is computed as

$$\sum_{K_j \in K} \frac{|K_j|}{n} \max_{C_i \in C} (F_\beta(C_i, K_j)).$$

Using this measurement a comparison between all three implemented algorithms is possible.

7.4 Results

7.4.1 CLIQUE

The first series of test performed on the CLIQUE operator aims to evaluate how the number of intervals each dimension is partioned in affects the clustering result. The input was a TF-IDF vector space model. The density threshold τ was left constant at 0.2%.

The diagramms of 7.2 show that a increase of partitions leads to a decrease in subspaces found especially in higher dimensions. Another interesting finding is that the average

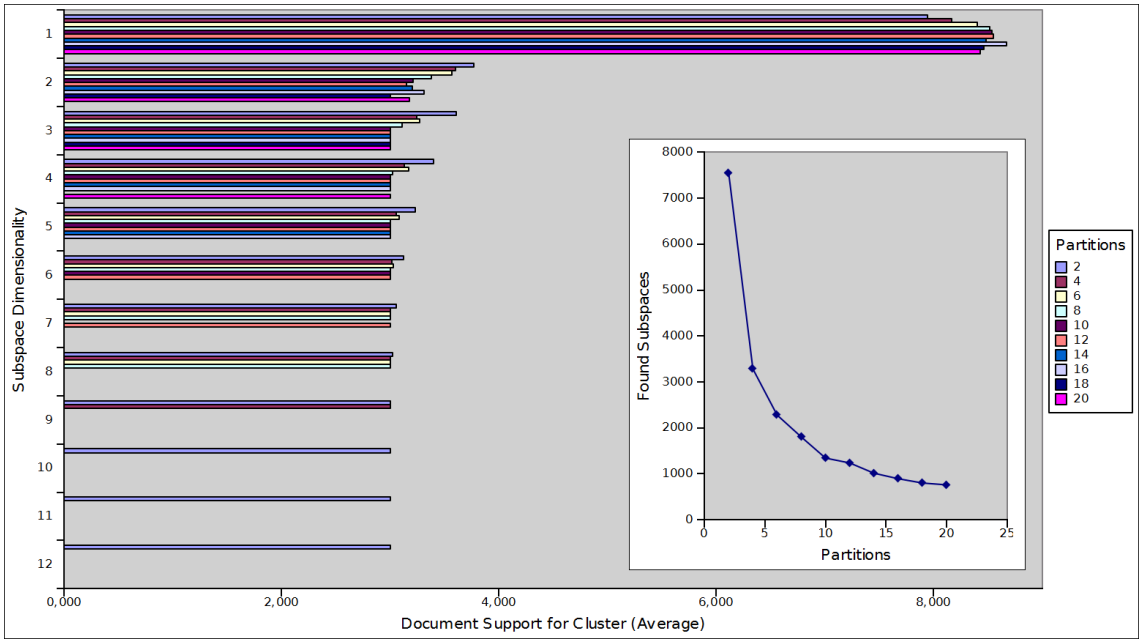

Figure 7.2: The effects of the number of partitions on average cluster size (big diagramm) and found subspaces (small diagramm)

cluster size of the low dimensional subspaces is not affected as much as the number of subspaces found. The number of found clusters in the one dimensional subspaces predominantely stayed at 1 indicating that a cluster was built over the whole TF-IDF value range of the particular term.

The number of partitions also affected the overall clustering quality. Table 7.2 shows that

ξ	$F_1(C)$	$F_2(C)$
4	0.216	0.131
8	0.214	0.129
12	0.209	0.122
16	0.208	0.118
20	0.210	0.119

Table 7.2: Effects of the partition number on the clustering quality

the clustering quality based on aggregated F-Measure got worse as the number of partitions increased. The effect on the F_2 measure was also greater indicating that a higher partitioned space will result in a worse recall overall in text clustering. The reason for this is that documents that share a single term set with more than one item will not be clustered together in higher partitioned space if the single terms of the term set appear in different frequencies; especially with sparse datasets.

To find all possible higher dimensional subspaces it was decided to partition the space into only one interval. With this restricion added to the previous sparse data restricion CLIQUE's subspace search is equivalent to the relevant term set search of FTC.
The last round of test performed on CLIQUE evaluated the affects of the density threshold τ on the cluster quality. The results in table 7.3 show that CLIQUE returns it best

τ in %	No. Subspaces	$F_1(C)$	$F_2(C)$	Maximum Dimensionality
0.2	9191	0.218	0.131	12
0.3	1739	0.217	0.131	6
0.4	930	0.217	0.131	4
0.5	621	0.214	0.130	3
0.6	461	0.213	0.130	3
0.7	333	0.213	0.130	2
0.8	261	0.213	0.130	2
0.9	219	0.213	0.130	2
1.0	181	0.213	0.130	2
1.1	149	0.212	0.130	2
1.2	121	0.212	0.130	2
1.3	105	0.212	0.129	2
1.4	84	0.212	0.129	2
1.5	70	0.212	0.129	1

Table 7.3: Effects of the density threshold

result under the F-Measurement at a completely unacceptable number of subspaces. The clustering quality does not decrease dramatically with higher thresholds but the subspaces found only contain one or two terms. Furthermore, the clustering produced is still overlapping. The conclusion of the tests show that CLIQUE has no advantages to a simple term fequency mining with regards to its application for homograph disambiguation.

7.4.2 Frequent Term-Based Text Clustering

The only parameter that influences the clustering of FTC is the minimum support for relevant term sets. The pictures in 7.4 show that number of clusters reduces dramatically for even a small increase in the required minimum support.
With higher minimum support FTC starts to produce a cluster that belongs to the empty term set. For this evaluation documents that belong to this cluster are declared as unclustered because no enrichement of the data took place.

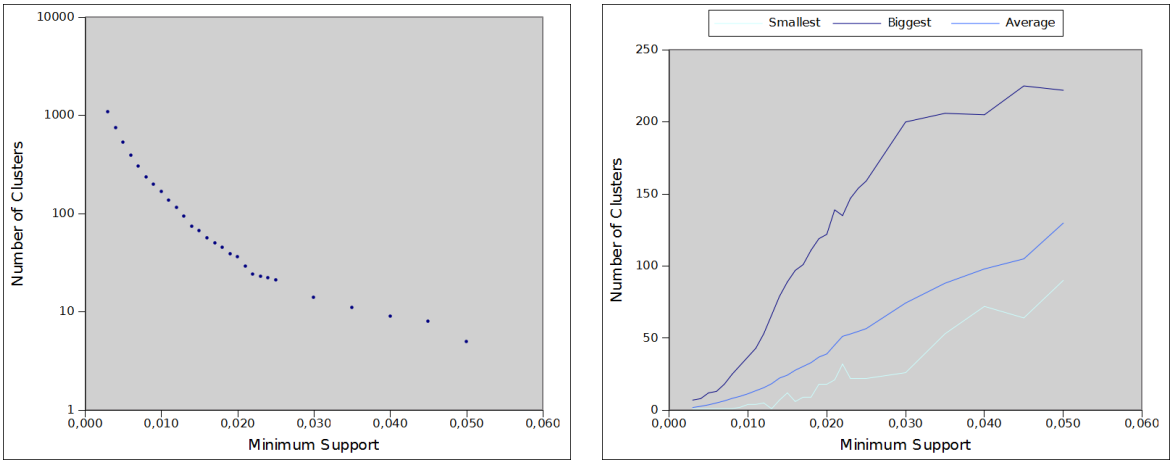

Table 7.4: The cluster number (logarithmic scale) and sizes in relation to the minimum support

An interesting point to note is that the biggest and smallest clusters do not increase strictly monotonically with higher minimum support. This is an result of the entropy overlap selection. If two clusters with a high document overlap are selected after each other the cluster that was selected first will contain all documents while the second only keeps its non-overlapping documents; even high minimum support can lead to very small clusters this way.

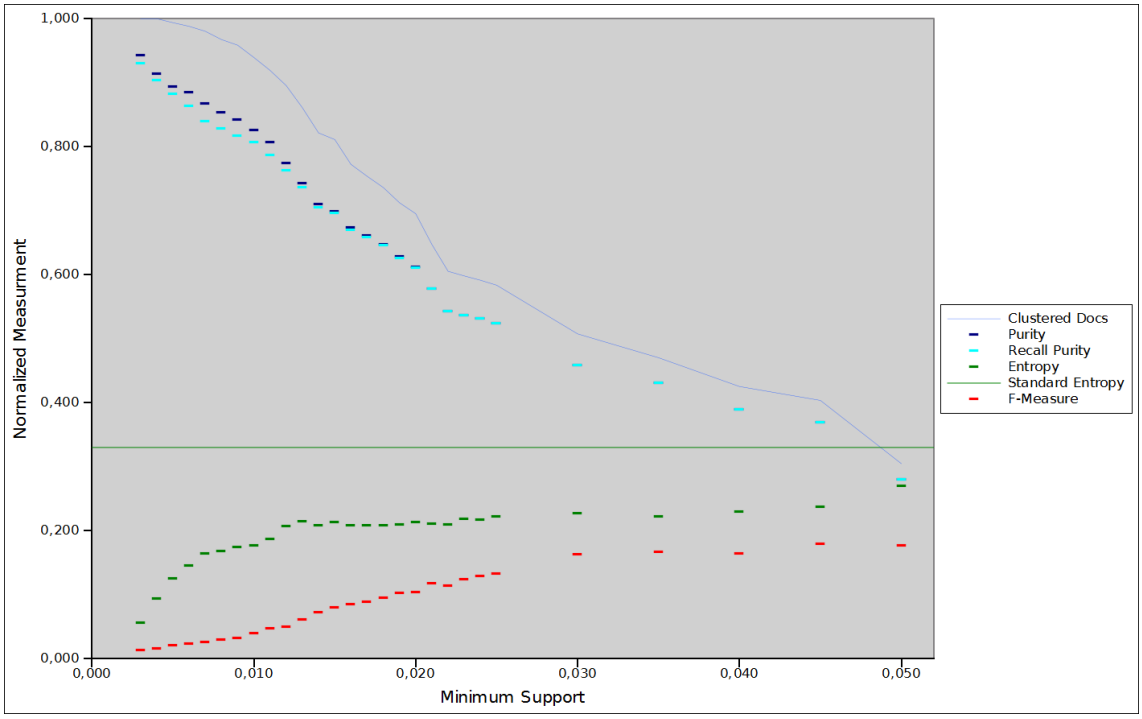

Figure 7.3: The different evaluation measurements normalized.

Diagramm 7.3 shows the different clustering quality results. The entropy measure was normalized from its $[0, ln(|K|)]$ value range. The "Standard Entropy" line represents the entropy of the class distribution of the data set.
FTC produces seemingly excellent results for very small minimum supports based on the entropy. Unfortunately the good entropy is the result of the small average cluster size. Many clusters do not even reach a size higher than the number of classes which means that many very small clusters produce a single cluster entropy of zero, which skews true cluster quality based on entropy. The true measurement of cluster quality for small minimum support values is better reflected by F-Measure, which punishes the large amount of small clusters better than entropy.
Another interesting observation for small minimum support values is the divergence between the regular purity measure and the purity measure based on recall, which is generally lower. FTC seems unable to produce pure clusters for documents located in smaller classes. The biggest problem of FTC is, however, the increasing number of unclustered documents for higher minimum support values. While the entropy measure stays under the standard entropy, which means a reduction in classification uncertainty, it is important to note that it only applies to a decreasing number of documents (represented by the "Clustered Docs" line in 7.3). At a minimum support of 3% FTC still produces 14 cluster that only contain 52% of the documents. Furthermore, the term sets that produce the clusters often only contain one item, for the same reasons CLIQUE only produces one dimensional subspaces for higher density thresholds.
FTC has many advantages over CLIQUE like flat clustering and the correct treatment of sparse data but the too high number of clusters make it only useful for data enrichment and disqualify it from actual automatic homograph disambiguation.

7.4.3 K-Subspace

K-Subspace required a different testing procedure because it is the only examined algorithm that is not deterministic (initilization of the first assignment). For tests of K-Subspace ten passes of clustering were performed. Boxplots were chosen for the visualization of the results.
The first set of test that was performed with K-Subspace regarded the radius of the Sphere model. Using the TF-IDF for input it was discovered that even small increases in the radius of the sphere model would decrease the classification accuracy. The cause for this reduction is that all values in TF-IDF lie in a very small interval and a sphere with even a small radius covers a huge region of the whole data set, leading to almost all assignments going to one sphere model. Therefore the sphere radius (defined by parameters σ and η) was set as zero.

K-Subspace still performed badly with accuracies that barely lie over random chance

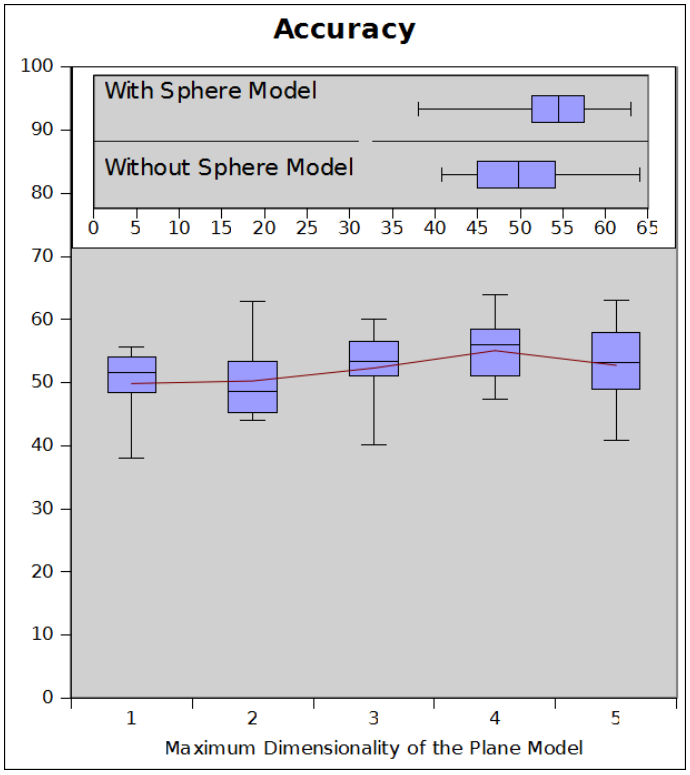

Figure 7.4: Accuracy of K-Subspace classification in %. Red line represents the average accuracy.

assignments. The entropy evaluation 7.5 shows that almost no uncertainty is removed by the cluster classification.

The relatively good accuracy performance around 50% for a five class classification problem 7.4 is a result of the inbalance of the data set. The cluster purity measure of all passes stayed at a constant 0.831, which is the fraction of documents that belong to the biggest class.

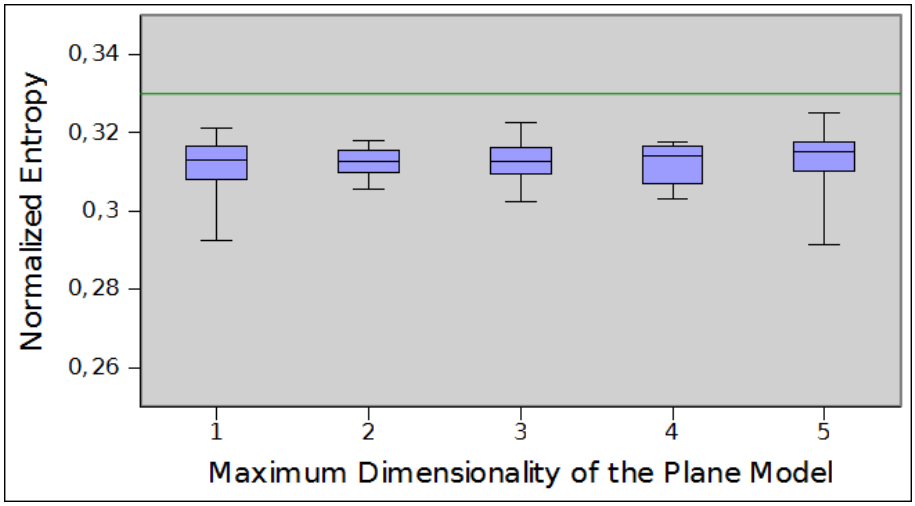

Figure 7.5: K-Subspace entropy. Line represents the standard entropy of the data set.

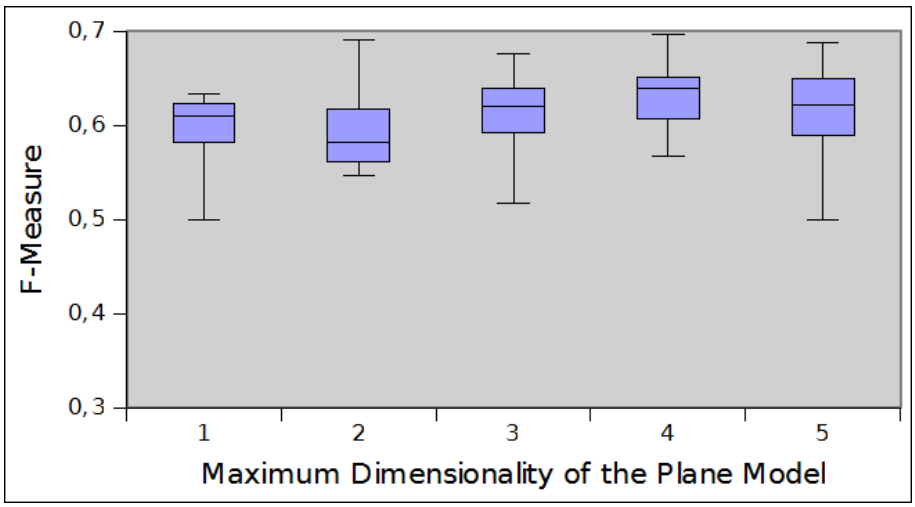

Figure 7.6: F-Measure of K-Subspace

A k-Nearest-Neighbor learner was used for comparison. K-Subspace was outperformed in all measurements. A balanced sampling of 10% of the data set was used for training.

	Accuracy	Precision	Recall
K-Subspace	54.29%	22.12%	18.05%
k-NN	67.08%	29.21%	40.18%

Table 7.5: Average accuracy measurements of K-Subspace in comparison to a k-NN learner. Precision and recall are the mean weighted to the predicted class size.

Chapter 8

Summary and Conclusion

The high-dimensionality of data in text processing makes the development of subspace clustering algorithms with good classification accuracy an essential task for text mining. This work discussed the merit of the three clustering algorithms CLIQUE, FTC, which have a strong relation to the frequent term set search algorithm APRIORI, and K-Subspace, a correlation clustering algorithm.

The empiric evaluation on the task of homograph disambiguation of all three algorithms showed that both algorithms that search for axis-parallel subspaces, CLIQUE and FTC, produce a result with too many clusters for a satisfying classification. FTC showed promise for data enrichment because the cluster descriptions, the frequent term sets, could be used to train a supervised classifier.

K-Subspace was able to provide a clustering that can be used as a classification, but its accuracy is too low and too dependent on the randomized initialization to be of any aid for homograph disambiguation.

While the results were dissapointing they only represent a fraction of the current subspace clustering methods and were only performed on a small data set. The performance of k-NN suggest that a top down subspace clustering algorithm, which relies on local similarity for clustering might perform better than the algorithms in this work.

Appendix A

Apendix

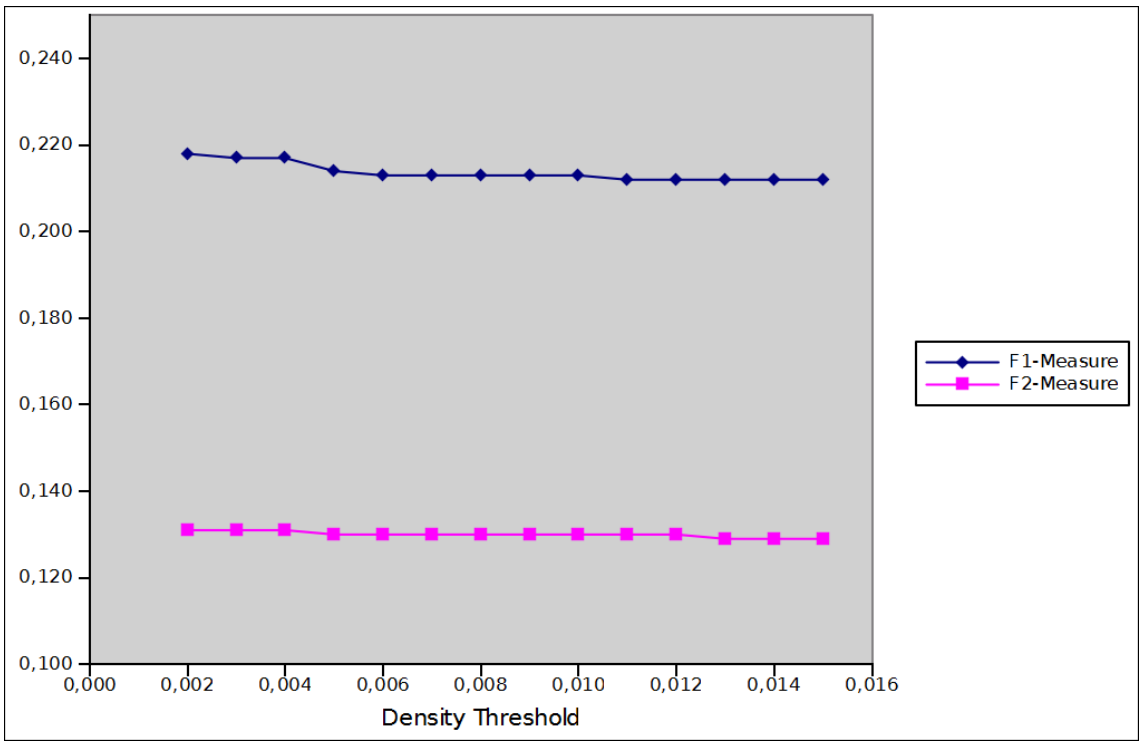

Figure A.1: F-Measures of CLIQUE based on the density threshold.

MinSupp	Purity	Purity (recall)	F-Measure	Non Clustered	#Clusters	Cluster Size Biggest	Smallest	Average
0,003	0,943	0,930	0,013	0	1082	7	1	1,842
0,004	0,914	0,903	0,016	0	743	8	1	2,682
0,005	0,894	0,882	0,020	12	535	12	1	3,703
0,006	0,884	0,863	0,023	23	393	13	1	5,013
0,007	0,867	0,839	0,026	38	304	18	1	6,431
0,008	0,853	0,828	0,029	63	236	25	1	8,178
0,009	0,842	0,817	0,032	80	199	31	2	9,613
0,010	0,825	0,807	0,039	117	166	37	4	11,370
0,011	0,806	0,787	0,047	156	136	43	4	13,507
0,012	0,774	0,762	0,050	202	115	53	5	15,574
0,013	0,743	0,736	0,061	269	94	66	1	18,340
0,014	0,710	0,705	0,072	345	74	79	7	22,270
0,015	0,699	0,696	0,080	365	67	89	12	24,299
0,016	0,673	0,670	0,085	440	56	97	6	27,732
0,017	0,661	0,658	0,088	476	50	101	9	30,340
0,018	0,647	0,646	0,095	510	45	111	9	32,956
0,019	0,628	0,626	0,102	556	39	119	18	36,846
0,020	0,612	0,611	0,104	589	36	122	18	39,000
0,021	0,578	0,578	0,117	682	29	139	21	45,207
0,022	0,543	0,543	0,113	763	24	135	32	51,250
0,023	0,536	0,536	0,123	777	23	147	22	52,870
0,024	0,531	0,531	0,128	790	22	154	22	54,682
0,025	0,523	0,523	0,132	805	21	159	22	56,571
0,030	0,458	0,458	0,162	952	14	200	26	74,357
0,035	0,430	0,430	0,166	1024	11	206	53	88,091
0,040	0,389	0,389	0,164	1111	9	205	72	98,000
0,045	0,369	0,369	0,179	1153	8	225	64	105,000
0,050	0,280	0,280	0,177	1344	5	222	90	129,800

Table A.1: Result data of FTC experiments

List of Figures

List of Algorithms

Bibliography

[1] Abeles, Peter: *efficient-java-matrix-library*. `https://code.google.com/p/efficient-java-matrix-library/`, 2013. [Online; accessed 13-August-2013].

[2] Aggarwal, Charu C., Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc and Jong Soo Park: *Fast algorithms for projected clustering*. SIGMOD Rec., 28(2):61–72, June 1999.

[3] Aggarwal, Charu C. and Philip S. Yu: *Finding Generalized Projected Clusters in High Dimensional Spaces*. In *In SIGMOD*, pages 70–81, 2000.

[4] Agrawal, Rakesh and Ramakrishnan Srikant: *Fast Algorithms for Mining Association Rules in Large Databases*. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[5] al., Agrawal et: *Automatic Subspace Clustering of High Dimensional Data*. Data Mining and Knowledge Discovery, 11:5–33, 2005.

[6] al., Wang et: *K-Subspace Clustering*. Machine Learning and Knowledge Discovery in Databases, 5782:506–521, 2009.

[7] Baeza-Yates, Ricardo A. and Berthier Ribeiro-Neto: *Modern Information Retrieval*, pages 29–30. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[8] Beil, Florian, Martin Ester and Xiaowei Xu: *Frequent term-based text clustering*. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 436–442. ACM, 2002.

[9] Beyer, Kevin, Jonathan Goldstein, Raghu Ramakrishnan and Uri Shaft: *When Is "Nearest Neighbor" Meaningful?* In *In Int. Conf. on Database Theory*, pages 217–235, 1999.

[10] Bisson, Gilles and Syed Fawad Hussain: *Chi-Sim: A New Similarity Measure for the Co-clustering Task*. In Wani, M. Arif, Xue wen Chen, David Casasent,

Lukasz A. Kurgan, Tony Hu and Khalid Hafeez (editors): *ICMLA*, pages 211–217. IEEE Computer Society, 2008.

[11] Borg, I. and P.J.F. Groenen: *Modern Multidimensional Scaling: Theory and Applications*, page 207–212. Springer, 2005.

[12] Bourbaki, Nicolas: *Elements of mathematics. General topology. Part 1.* Hermann, Paris, 1966.

[13] Ester, Martin, Hans peter Kriegel, Jörg S and Xiaowei Xu: *A density-based algorithm for discovering clusters in large spatial databases with noise.* pages 226–231. AAAI Press, 1996.

[14] Hans-Peter Kriegel, Peer Kröger, Arthur Zimek: *Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering.* ACM Transactions on Knowledge Discovery from Data, 3:1–58, 2005.

[15] Hearst, Marti A.: *Noun Homograph Disambiguation Using Local Context in Large Text Corpora.* In *University of Waterloo*, pages 1–22, 1991.

[16] Hurford, James R. and Brendan Heasley: *Semantics : a coursebook*, page 123. Cambridge University Press, 1983.

[17] Luhn, H. P.: *The Automatic Creation of Literature Abstracts.* IBM Journal of Research Development, 2:159–165, 1958.

[18] MacQueen, J. B.: *Some Methods for classification and Analysis of Multivariate Observations.* Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, 1:281–297, 1967.

[19] Pearson, K.: *On lines and planes of closest fit to systems of points in space.* Philosophical Magazine, 2(6):559–572, 1901.

[20] Prof. Dr. Angelika Storrer, Prof. Dr. Katharina Morik et al.: *KobRA Projektbeschreibung.* `http://www.kobra.tu-dortmund.de/mediawiki/index.php?title=Projektbeschreibung`, 2012. [Online; accessed 13-August-2013].

[21] Prof. Dr. Angelika Storrer, Prof. Dr. Katharina Morik et al.: *Korpusbasierte linguistische Recherche und Analyse mit Hilfe von Data-Mining.* `http://www.kobra.tu-dortmund.de/`, 2012. [Online; accessed 13-August-2013].

[22] P.S. Bradley, O.L. Mangasarian: *k-Plane Clustering.* Journal of Global Optimization, 16:23–32, 2000.

[23] Rapid-I: *How to Extend RapidMiner 5.* http://docs.rapid-i.com/files/howtoextend/How%20to%20Extend%20RapidMiner%205.pdf, 2013. [Online; accessed 13-August-2013].

[24] Rapid-I: *RapidMiner.* http://rapid-i.com/, 2013. [Online; accessed 13-August-2013].

[25] Rijsbergen, C. J. Van: *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[26] R.L. Dobrushin, V.V. Prelov: *Entropy.* In *Encyclopedia of Mathematics*, 2001.

[27] Salton, G., A. Wong and C. S. Yang: *A vector space model for automatic indexing.* Commun. ACM, 18(11):613–620, 1975.

[28] Sibson, R.: *SLINK: An optimally efficient algorithm for the single-link cluster method.* The Computer Journal, 16(1):30–34, 1973.

[29] Stehman, Stephen V.: *Selecting and interpreting measures of thematic classification accuracy.* Remote Sensing of Environment, 62(1):77 – 89, 1997.

[30] Tan, Steinbach, Kumar: *Machine Learning: Cluster Evaluation.* http://www.uni-weimar.de/medien/webis/teaching/lecturenotes/machine-learning/unit-en-cluster-analysis-evaluation.pdf, 2013. [Online; accessed 13-August-2013].

[31] Tenenbaum, Joshua B, Vin de Silva and John C Langford: *A Global Geometric Framework for Nonlinear Dimensionality Reduction.* Science, 290:2319–2323, 2000.

[32] Woo, Kyoung gu, Jeong hoon Lee, Myoung ho Kim and Yoon joon Lee: *FINDIT: a Fast and Intelligent Subspace Clustering Algorithm using Dimension Voting.* In *PhD thesis, Korea Advanced Institute of Science and Technology*, page 2004, 2002.

[33] Yarowsky, David: *Homograph Disambiguation in Text-to-Speech Synthesis.* In Santen, J. van, R. Sproat, J. Olive and J. Hirschberg (editors): *Progress in Speech Synthesis*, pages 159–175. Springer-Verlag, New York, 1996.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 2. September 2013

David Spain