Tag Cloud Control by Latent Semantic Analysis

Master's Thesis

submitted by
Angelina Velinska
IMT

supervised by

Prof. Dr. rer. nat. habil. Ralf Möller
Dipl. Ing. Sylvia Melzer
Software Systems Institute

Prof. Dr. Karl-Heinz Zimmermann
Institute of Computer Technology
Hamburg University of Technology

Dr. Michael Fritsch
CoreMedia AG
Hamburg

Hamburg University of Technology
December, 2010

Declaration

I declare that:

This work has been prepared by myself, all literal or content based quotations are clearly pointed out, and no other sources or aids than the declared ones have been used.

Angelina Velinska

Hamburg
December, 2010

Acknowledgements

I would like to thank Prof. Dr. Ralf Möller for his advice and guidance throughout the work on this thesis.

My special gratitude goes to Dr. Michael Fritsch in CoreMedia AG, Hamburg for his great patience, and constant support. This work would not exist without him. Michael, I owe you the implementation.

A big ”Thank you” to Sylvia Melzer for keeping me on track despite the distance, and my poor German.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1

Introduction

1.1 Motivation

Information Retrieval (IR) systems become more important not only due to their use in Internet and digital libraries, but also because the majority of companies organize their activities around information and depend on digital documents. Finding the right information brings value to the business, and failing to do so usually leads to losses.

Certain factors influence the performance of search applications. On one hand, it is the constantly growing problem of information overload. On the other, it is the peculiarities of humans users which have to be taken into consideration. IR systems need to adapt to their users, and to their information need.

- **Human behavior**. Users searching for information usually submit queries composed of a few keywords only, as shown in studies [1]. For example, the average web users enter around 3 words per query[1]. The search application performs exact matching between the submitted keywords, and the document set it searches through, and often returns a long list of results. When searching with short queries, the focus of users is unclear, and the missing information contributes to long result lists containing many irrelevant hits. Users without domain expertise are not familiar with the appropriate terminology, and therefore they don't submit the right query terms with respect to relevance or specialization. Another issue is the ambiguity of words, in the cases when words have more than one meaning. When relevant documents contain words

[1]`http://www.hitwise.com/us/press-center/press-releases/google-searches-apr-09`, accessed December, 2010

that are semantically relevant to the queries but not the same (synonyms), they will not be judged relevant. As a consequence, search results will not fit the information needs of users.

- **Manual processing of results**. When a large number of matching documents is returned as a search result, only some of the documents can be read, due to human limitations in information processing, and time constraints (users want to find information quickly). Human users need to narrow down the search iteratively by reformulating the query, since it is unclear in which context their queried words are used, and which words could be useful to focus the search. This reformulation can be a frustrating and time consuming process.

- **Information need**. Search applications that implement the keyword search paradigm (or full-text search) are broadly accepted by users; however, the challenge for the next years is the development of search solutions that reflect users' context ("what the user meant" versus "what the user entered as a query"). In other words, solutions that are able to: *a*) organize search results better than in the form of long lists, *b*) adapt to a users personal skills and experience concerning the underlying document collection, and *c*) adapt to the retrieval task a user is concerned with; in other words, to adapt to the users' information need.

The factors listed above have lead to the development of techniques that assist users to effectively navigate and organize search results, with the goal to find the documents best matching their needs. *Document clustering* is a process of forming groups (clusters) of similar objects from a given set of inputs. When applied to search results, clustering organizes the results into a number of easily browsable thematic groups. It contributes to solving the problem of finding a meaningful ordering in a large amount of returned results. *Latent Semantic Analysis* is another method from IR that handles the problem of words having more than one meaning, or words ambiguity. It also filters out noise well, and reduced the number of unrelevant hits. Both techniques have been implemented in this work.

After documents have been clustered into categories according to their topics, they have to be presented to the users. In particular, the categories have to be labeled with characteristic terms for browsing. Which words from the cluster to choose as labels is a difficult problem to solve. This work presents a relatively new algorithm for cluster labeling, called Weighted Centroid Covering (WCC) (Stein and Meyer zu Eissen [2], [3]).

It evaluates it and proposes an improvement of WCC by using external semantic knowledge for definition of the cluster labels. As a part of this work, a domain-specific ontology is developed, which is used as a source of external knowledge.

1.2 Information Retrieval systems

Techniques from IR field are implemented as a part of this work. Therefore, what follows is an overview of the text analysis processes, common to most IR systems. Then, in the context of these processes, the contributions presented in this thesis will be summarized.

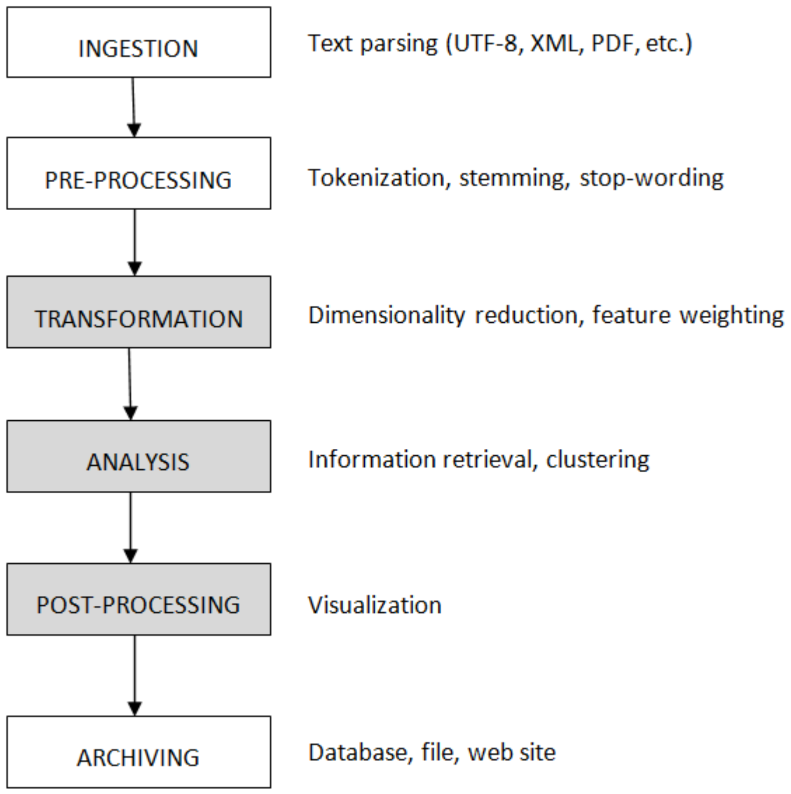
INGESTION
Text parsing (UTF-8, XML, PDF, etc.)
PRE-PROCESSING
Tokenization, stemming, stop-wording
TRANSFORMATION
Dimensionality reduction, feature weighting
ANALYSIS
Information retrieval, clustering
POST-PROCESSING
Visualization
ARCHIVING
Database, file, web site

Figure 1.1: Workflow in IR systems

Most IR systems share common workflow, and follow common data processing stages. Some of the processes involved in text analysis are: lexical analysis to study word frequency distributions of words, retrieval, tagging/annotation, and visualization among the others. The workflow in IR systems usually starts with **document ingestion**, where texts from the document corpus[2], which will be analyzed, are parsed or loaded into memory. After that comes the **preprocessing** phase, responsible for text extraction, tokenization, token-filtering, text folding, etc. In this phase, documents are often represented as vectors, whose components quantify properties like how often words occur in the document. In the **transformation** phase is where a dictionary, matrix or other form of index is created from the extracted texts. In this phase all extracted terms[3] are assigned weights by applying a weight function. In Chapter 2.6 are given more details about the most common weight functions used in IR systems. The **analysis** phase can include a retrieval process by querying the corpus, clustering, etc. The **visualization** phase is where concepts, query results, or summarizations extracted from the text collection are presented to users. And in the final **archiving** phase, results from the text analysis process can be saved.

The workflow in fig. 1.1 doesn't show any iterations or cycles between phases for simplicity. Iterations and repetition of certain phases of analysis, however, are common, e.g. between transformation and post-processing phases, or analysis and post-processing.

The focus of the current work is on transformation, analysis and post-processing phases of IR workflow.

1.3 Goal and scope of work

This works contributes with the following:

1. It makes an evaluation of Weighted Centroid Covering algorithm, proposed for unsupervised topic labeling of clusters ([2] and [3].

2. It proposes an improvement in WCC algorithm, performing topic identification based on external knowledge. A light-weight domain-specific ontology has been developed for this purpose, in order to be used as a reference for external semantic knowledge during cluster labeling.

3. A software application for executing an IR process has been developed. It implements Latent Semantic Analysis (LSA) for information retrieval (analysis phase from fig. 1.1), and presents the main

[2]Throughout this work, *document corpus* is used as a synonym to a document collection, in which IR tasks are performed.

[3]A term in this context denotes a word after preprocessing of the texts. A term is a single unit, e.g. a word, a phrase, a number, or an abbreviation.

concepts contained in a document set using a tag cloud.

1.4 Outline

This chapter motivates the presented work, and summarizes its contributions. It also offers a general overview of the phases of text analysis, common to most IR systems. Chapter 2 gives theoretical foundations for preprocessing and transformation phases of text analysis, and reviews a specific technique for information retrieval, called Latent Semantic Analysis. In Chapter 3, WCC algorithm is evaluated, and a proposition for its improvement is made, by using a light-weight domain-specific ontology. Chapter 4 refers to the post-processing phase of text analysis, giving the visualization means to present the main concepts retrieved from a document set as a tag cloud. The software contribution, developed as a part of this work, is described in Chapter 5, where test results are also present, and an evaluation of the implementation is made. Finally, in Chapter 6 the thesis concludes with ideas for future work.

Chapter 2

Latent Semantic Analysis

2.1 Information Retrieval process

IR systems aim to satisfy user's information needs, by providing access to large collections of documents. In a search application, the IR process retrieves a set of documents which matches a given query. There are three basic processes which an IR system has to support: to represent the content of the documents, to represent the user's information need, and to compare the two representations, based on a chosen similarity measure (fig. 2.1). Therefore, the first stage of constructing an IR system is to extract information about the documents content, and implement a similarity measure, based on which documents and queries can be compared. Representing the documents is usually called the *indexing process*. The comparison of the query against the document representations based on a chosen measure is called the *matching process*.

2.2 Document representation

In order to find documents which are similar to a given query, both documents and query have to be comparable, or have the same representation in the IR system. Various models have been proposed for internal representation of documents and queries. The *Boolean*, *Vector space* and *Probabilistic models* are popular IR models that find wide implementation. The Boolean model represents both documents and queries as a set of terms, and compares them based on boolean functions (AND, OR, NOT, etc.). The probabilistic model uses probabilistic inference for document retrieval [5]. Similarities are computed as probabilities that a document is relevant for a given query. And finally, the Vector Space Model (VSM), introduced first by Salton [6],

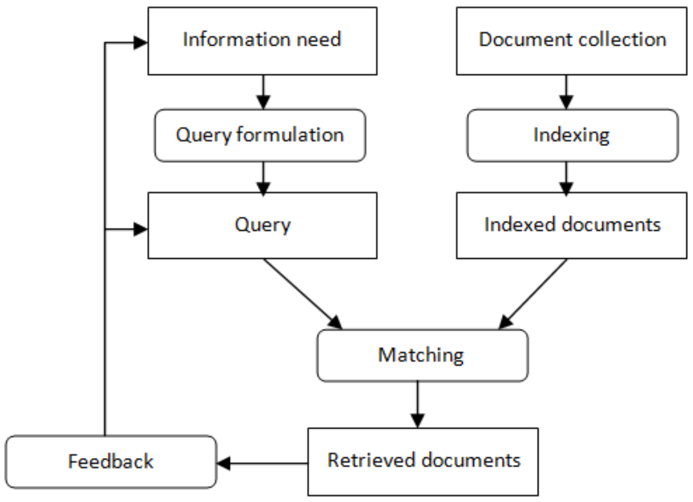

Figure 2.1: Information retrieval process, adapted from [4]. The information need of the user is formulated as a query, which is transformed in the chosen model of the IR system. Documents from the collection are also represented according to the chosen model. Based on the implemented similarity measure, matching documents are identified. Finally, the retrieved results are presented to the user. If the user is not satisfied with the search results, the search query can be reformulated during feedback.

represents both documents and queries as vectors in a multidimensional space, whose dimensions are the terms used to build an index to represent the documents (section 2.2.1 provides more details on VSM). Boolean model is easy to implement; however, when querying, users need to be familiar with boolean operators, which is a drawback of this model. Concerning the probabilistic model, prior knowledge is required for its implementation, as it usually includes tuning of independent probabilistic parameters. VSM and the probabilistic model both have the advantage that they rank the relevant results according to a chosen weight function, but the former is easier to implement.

2.2.1 Vector Space Model

During indexing (fig. 2.1), documents are presented as data structures in memory. In VSM a document is a vector, whose elements represent properties like term frequencies, or frequency of word occurrence within the document. Before documents can be represented as vectors, they have to be tokenized, or converted from a stream of characters to a stream of words. Thus parsed, words will be used in building an index of the document collection. During tokenization one can apply filtering, i.e.

removing HTML tags or other markup from text, as well as stop-words and punctuation marks removal. Stop words are such words that don't convey information specific to the text corpus, but occur frequently, such as: $a, an, and, any, some, that, this, to$.

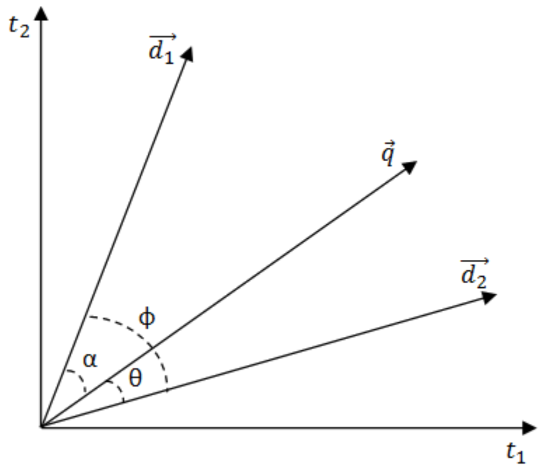

Figure 2.2: The Vector Space Model. Documents $\vec{d_1}$ and $\vec{d_2}$, and a query vector $\vec{q}$ are represented in a two-dimensional vector space, formed by terms t_1 and t_2.

A distinction has to be made between words or terms, and tokens. A term is the class which is used as a unit during parsing, and a token is each occurrence of this class. For example, in the sentence:

> *CoreMedia CMS is shipped with an installation program for interactive graphical installation and configuration of the software.*

the term *installation* is represented by two tokens. There is no universal way to parse a text, and the parsing decisions to address depend on the application in which the text collection will be used.

After tokenization, documents are represented as vectors, where each term is a vector in the vector space, and the documents are the sum of the terms, from which they consist. Thus, all document vectors and terms form a multi-dimensional vector space, where terms are the dimensions, and documents - the corresponding sum vectors. A diagram of the vector space is given in fig. 2.2, where two document vectors $\vec{d_1}$ and $\vec{d_2}$, and a query vector $\vec{q}$ are represented in a two-dimensional space.

2.2.2 Weight functions

Vectors in the VSM have as elements the occurrence frequencies of words in documents. However, some documents are shorter than others, therefore one has to apply a normalization function in order to avoid representing words from longer documents as "more important" than words from shorter documents, as they would occur more often. Such normalization functions are called weight functions, and they are applied after the vector space has been constructed.

Weight functions are generally separated into two categories - local and global. They are often implemented as a pair together, because local functions measure the importance of a given word in a single document, while global functions give the importance of a word in the whole document collection. The most commonly used function pair is *term frequency* and *inverse document frequency.*

Term frequency - inverse document frequency

The simplest local weight is the term frequency $tf_{t,d}$. It assigns a weight to each term equal to the number of occurrences of the term t in a given document d. However, not all words carry the same meaning in text (therefore, stop words are removed during preprocessing, as mentioned in 2.2.1). Words that are common to all documents in the collection don't reveal much information, as compared to words which occur only in several documents. The latter are more likely to contain key information about the meaning of these documents. This is reflected by the weight function *inverse document frequency* (eq. 2.1)

$$idf_t = 1 + log\frac{N}{df_t} \tag{2.1}$$

where N is the total number of documents in the collection, df_t is the frequency of occurrence of term t in the whole collection, and t is a specific term being weighted. Using idf_t is a way to scale down the importance of commonly used words in text. When one combines both tf and idf, a composite weight is produced for each term in each document. The *tf-idf* weighting function assings to a term t in a document d a weight given by

$$(tf - idf)_{t,d} = tf_{t,d} \times idf_t \tag{2.2}$$

.

As defined by Manning et al. [7], the weight assigned to term t in document d by using a combination of local and global weight function is

1. highest when t occurs many times within a small number of documents;
2. lower when the term occurs fewer times in a document, or occurs in many documents;
3. lowest when the term occurs in virtually all documents.

Log - entropy

Another popular pair of weight functions, frequently used in text analysis, is the *log-entropy* pair. The local function *log* takes the logarithm of the raw term frequency, thus normalizing the effect when large differences in term frequencies occur. In eq. 2.3 $L(t, d)$ denotes the log of number of occurrence of a term t in a document d.

$$L(t, d) = \log(tf(t, d) + 1) \tag{2.3}$$

The global function *entropy* $H(t)$ reflects the local relative importance of a term t in the whole document collection (eq. 2.4)

$$H(t) = 1 + \frac{\Sigma_j p(t, d)}{\log n} \tag{2.4}$$

where n is the number of documents in the whole collection. In eq. 2.4, $p(t, d)$ is defined by:

$$p(t, d) = \frac{tf_{t,d}}{gf_t} \tag{2.5}$$

where gf_t is the total number of times that term t occurred in all documents. The entropy measures the distribution of terms over all documents.

2.2.3 Similarity measures

Once the vector space has been built, one can find the documents which are most similar to a given query. During *query formulation*(fig. 2.1), the queries are tokenized and represented as vectors, as already described in section 2.2.1. Therefore, the similarities between documents and queries can be measured based on the angles between their respective vectors (in fig. 2.2.1, these are α and θ). Using the angles between vector representations, one can define a similarity measure which is necessary for the matching process in IR systems. The standard way to computing the similarity between two documents d_1 and d_2 is to compute the *cosine similarity* of their vector representations $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ (eq. 2.6).

$$sim(d1, d2) = \frac{\overrightarrow{V}(d_1).\overrightarrow{V}(d_2)}{\left|\overrightarrow{V}(d_1)\right|.\left|\overrightarrow{V}(d_2)\right|}, \tag{2.6}$$

where the numerator represents the *dot product*[1] of the vectors, while the denominator is the product of their *Euclidean lengths*[2]. The measure from eq. 2.6 is the cosine of the angle ϕ between the two vectors $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$.

Once we represent a collection of N documents as a collection of vectors, it is easy to obtain a natural view of the collection as a *term-document matrix*: this is a $m \times n$ matrix whose rows represent the m terms in the document collection, and each of whose n columns corresponds to a document. And this specific matrix grouping all documents and terms from the collection is used in *Latent Semantic Analysis*, a technique which is discussed next.

2.3 Latent Semantic Analysis

LSA was first introduced by Dumais et al. [8] and Deerwester et al. [9] as a technique for improving information retrieval. It is based on the Vector Space Model, where as already discussed, words from users' queries are matched with the words in documents. Such IR models that depend on lexical matching have to deal with two problems: *synonymy* and *polysemy*. Due to the many meanings which the same word can have, also called polysemy, irrelevant information is retrieved when searching. And as there are different ways to describe the same concept, or synonymy, important information can be missed. LSA has been proposed to address these fundamental retrieval problems, having as a key idea dimension reduction technique, which maps documents and terms into a lower dimensional semantic space. LSA models the relationships among documents based on their constituent words, and the relationships between words based on their occurrence in documents. By using fewer dimensions than there are unique words, LSA induces similarities among

[1]The dot product $\overrightarrow{x}.\overrightarrow{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$.

[2]Let $\overrightarrow{d}$ is the document vector for d, with M components $\overrightarrow{d_1}...\overrightarrow{d_M}$. The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \overrightarrow{d_i^2}}$

words including ones that have never occurred together [10]. The basic steps in using LSA are: document representation (the same as in VSM), Singular Value Decomposition (SVD) with dimensionality reduction, and querying. Next, the theoretical basis for LSA is given, as it has been implemented as a part of this work.

As mentioned above, the first step of LSA implementation is document representation. It is similar to the representation in VSM, therefore refer to section 2.2. After tokenizing all documents in the collection, and computing the corresponding term weights, one has to construct a term-document matrix (eq. 2.7). Having as rows the terms, and as columns the documents, its elements are the occurrences of each term in a particular document, where a_{ij} denotes the frequency with which term i occurs in document j. The size of the matrix is **m x n**, where **m** is the number of terms, and **n** is the number of documents in the text collection. Since every term doesn't appear in each document, the matrix is usually sparse.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{2.7}$$

Local and global weightings are applied to increase or decrease the importance of terms within documents. After presenting the most popular weight functions in sections 2.2.2, we can write

$$a_{ij} = L(i, j) \times G(i), \tag{2.8}$$

where $L(i, j)$ is the local weighting of the term i in document j, and $G(i)$ is the global weighting for term i. As examples of local functions, the term frequency tf and *log* were discussed, while *idf* and entropy were the examples for global weightings. The choice of weight function impacts LSA performance. In section 2.6 reasons are given for the specific implementation decisions made in this work concerning LSA.

2.4 Singular Value Decomposition

After its generation, the term-document matrix is decomposed into three matrices (eq. 2.9) by applying SVD. It is a unique decomposition of a matrix into the product of three matrices - U, V and Σ, where U and V

are orthonormal matrices[3], and Σ is a diagonal matrix[4] having singular values[5] on its diagonal.

$$A = U\Sigma V^T \tag{2.9}$$

After the initial matrix A is decomposed, all but the highest k values of its product diagonal matrix Σ are set to 0. When A is recomputed again following eq. 2.9, the resulting matrix A_k represents the semantic space of the text collection. A classical visual example can be used to presenting SVD (as given in [8]) into three product matrices. It can be seen here how the dimensionality reduction of matrix Σ affects all three product matrices.

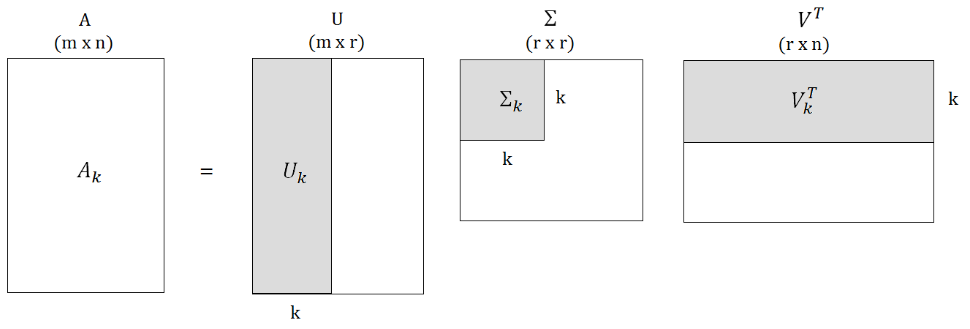

Figure 2.3: Diagram of truncated SVD

A_k - best rank-k approximation of A	m - number of terms
U - term vectors	n - number of documents
Σ - singular values	k - number of factors
V^T - document vectors	r - rank of A

In fig. 2.3, U and V contain the term and document vectors respectively, and Σ is constructed by the singular values of A. An important property of SVD is that the singular values placed on the diagonal of Σ are in decreasing order. Hence, if all but the first k singular values are set

[3]An orthonormal matrix is a matrix, whose columns, treated as vectors, are also orthonormal. A matrix is orthonormal, if its transpose is equal to its inverse. For more information on matrices and matrix operations, refer to [11]

[4]A diagonal matrix is a square matrix, in which the entries outside the main diagonal are all 0.

[5]For a square matrix A, the square roots of the eigenvalues of A^TA, where A^T is the conjugate transpose, are called *singular values*. Given a matrix A, a non-zero vector x is defined to be an *eigenvector* of the matrix, if it satisfies the *eigenvalue equation*: $Ax = \lambda x$ for some scalar λ. The scalar λ is called an *eigenvalue* of A corresponding to the eigenvector x.[11]

to 0, the semantic meaning in the resulting space is preserved to some approximation k, while noise or variability in word usage, is filtered out. Noise in this case are the terms with lowest weights which carry little meaning. By using fewer dimensions k, LSA induces similarities among terms including ones that have never occurred together. Terms which occur in similar documents, for example, will be near each other in the k-dimensional vector space even if they never co-occur in the same document. This means that some documents that do not have any common words with a given query may be near it in resulting k-space.

A factor to be considered when computing SVD is the run-time complexity of the algorithm. For decomposition of very large matrices, it is $O(n^2k^3)$, where n is the number of terms in the text corpus, and k is the number of dimensions in semantic space after dimensionality reduction. Note that k is typically a small number between 50 and 350.

A more detailed description of SVD can be found in [12] and [11].

2.5 Querying the semantic space

In this work LSA is used for IR purpose. Therefore, the final step of applying the technique is to pose queries on the constructed semantic space. A query q is a set of words which must be represented as a document in the k-dimensional space, in order to be compared to other documents. The user's query can be represented by using eq. 2.10:

$$q = q^T U_k \Sigma_k^{-1} \tag{2.10}$$

where q is the set of words in the query, multiplied by the reduced term and singular values matrices. Using the transformation in eq. (2.10), the query is "mapped" onto the reduced k-space. After the mapping, the resulting query vector can be compared to the documents in the k-space, and the results ranked by their similarity or nearness to the query. A common similarity measure used to compute similarity between the query and the document vectors is the cosine (eq. 2.6). From the resulting document set, the documents closest to the query above certain threshold are returned as search results.

2.6 Factors which influence LSA performance

The effective usage of LSA is a process of sophisticated tuning. Several factors can influence the performance of the technique. These factors are pre-processing of texts (removal of stop-words, filtering, stemming), frequency matrix transformations, choice of dimensionality k, choice of similarity measure. Dumais et al. [13] and Nakov et al. [14] have carried out research on LSA performance depending on the choice of the above mentioned factors. They conclude that LSA performance depends on the particular text corpus, as well as on the purpose of LSA application. However, in the case of matrix transform, log-entropy (section 2.2.2) performs better as compared to other matrix transform function combinations, including the popular term frequency - inverse document frequency ($tf - idf$) (section 2.2.2). Therefore, the former is implemented in this work. It has been further stated ([13],[15]) that with respect to similarity measures used, LSA performs optimal when cosine similarity measure is implemented to calculate the distance between vectors in the semantic space (eq. 2.6). It is therefore used in this work to measure the relevance between queries and documents. And finally, the dimensionality reduction parameter k is defined empirically based on the experimentation results presented in Chapter 6.

Chapter 3

Cluster labeling

A major problem in text analysis is to determine the topics in a text collection and identify the most important or significant relationships between them. Clustering and visualization (e.g. in the form of tag clouds) are key analysis methods in order to address this problem. In this chapter, an algorithm for cluster labeling, which is relatively new, is discussed, and in Chapter 4 tag clouds are introduced as a visualization mean used in IR systems.

3.1 Clustering

Clustering is an IR method that groups objects (documents) together based on their similarities in so called clusters. Such methods are often applied in the post processing phase of IR (fig. 1.1) for the visualization of retrieved results. For example, similar search results can be grouped together during presentation of results in search engines.

Clustering can be used to categorize resources. *Document categorization* is the process of assigning a document to one or more categories. A categorization $C = \{c_1, c_2, ..., c_k\}$ partitions a document collection D into subsets $c \subseteq D$, so that $\cup_{i=1}^{k} c_k = D$. C is an exclusive categorization if $c_i \cap c_{j \neq i} = 0, c_i, c_j \in C$, and non-exclusive categorization otherwise. Clustering is an *unsupervised categorization method*, as it organizes documents into clusters without having prior knowledge about the clusters, or predefined cluster categories.

Presenting search results in groups, or categories, should help users gain a quick overview of the retrieved documents. An important part of clustering used for visualization of results, is to choose suitable labels of the categories defined, so that they are usable to human users. The process of *cluster labeling* automatically creates labels for clustered groups of objects.

3.1.1 Clustering algorithms

A large variety of clustering algorithms exists, and they are usually classified according to the characteristic of their output structure. Jain et al. [16] make a classification based on the following properties:

Flat vs. hierarchical clustering
Flat clustering creates a flat set of clusters without relations between clusters due to some structure. K-means is a popular clustering algorithm, which creates as output a flat structure of document clusters. The k-means clustering algorithm is presented as an example for flat clustering in section 3.1.2. *Hierarchical clustering* on the other hand creates a hierarchy of clusters, with parents and children nodes, in a tree-like manner (Johnson [17]).

Hard vs. soft clustering
Another important distinction can be made between *hard* and *soft* (also called fuzzy) clustering algorithms. If each document is assigned to a single cluster only, we have hard clustering. Otherwise, clustering is soft, and it assigns a document as a distribution over all clusters. This means that each document can have a fractional membership in several clusters. LSA (Chapter 2) is a good example for a soft clustering algorithm. K-means is a popular example for hard clustering (section 3.1.2).

Monothetic vs. polythetic clustering
Based on how documents are assigned to different clusters, clustering can be divided into *monothetic* and *polythetic* [18]. Monothetic algorithms assign a document to a cluster based on a single feature, while polythetic algorithms classify documents by overall degree of similarity/difference calculated over many properties. This makes monothetic clustering well suited for generating hierarchies and browsing search results, since a single feature, common to all documents in the cluster, describe each cluster. Users understand easily clusters generated in this way. K-means is an example of polythetic document clustering algorithm, where each cluster is described by several words of phases.

3.1.2 K-means clustering algorithm

In this work the cluster labeling algorithm Weighted Centroid Covering is evaluated. As clustering precedes cluster labeling, the k-means algorithm is used for document clustering. Two factors influence this implementation decision:

- As LSA is used to process the term-document matrix, it can be investigated which dimensionality reduction parameter k is optimal for the specific implementation and data set used in this work, so that the optimal number of topics (or soft clusters) can be obtained for the evaluation data set. Therefore, the dimensionality reduction parameter, previously defined in LSA, can be used as initial parameter for k-means, or the number of clusters inputted to the algorithm. Thus, one of k-means' weaknesses will not affect this work.

- Another reason to use k-means algorithm is its simplicity, as compared for example to Hierarchical Agglomerative Clustering (HAC) - the complexity of k-means is linear, while the complexity of HAC is exponential.

K-means is an example for a flat, hard and polythetic clustering algorithm, based on the classification mentioned previously. It was first introduced by Lloyd [19]. The algorithm outputs a flat unstructured set of clusters. It starts by dividing the document collection into k clusters, where k is an input parameter, and then computes the k *means* of these clusters. The algorithm defines a cluster in terms of a *centroid*, usually the *mean of a group of points*, and is applied to objects (documents) in n-dimensional space (such as vector space). After the initialization, the following two steps are iteratively repeated, until the clusters are not re-assigned any more:

1. Each document in the collection is assigned to the cluster with the closest mean.

2. For each cluster, new means are recomputed based on the documents it contains.

In algorithm 3.1 a detailed step-by-step description of the k-means algorithm is given as pseudo-code.

Algorithm 3.1 K-means clustering algorithm

```
Input: $D = \{d_1, d_2, ..., d_n\}$ - documents to be clustered
  $k$ - number of clusters
Output: $C = \{c_1, c_2, ..., c_k\}$ - clusters
  $m : D \to \{1...k\}$ - cluster membership
  Set $C$ to initial value (e.g. select $k$ random points as initial centroids)
  {Form $k$ clusters by assigning each document to its closest centroid}
  for all $d_i \in D$ do
    $m(d_i) = mindistance_{j \in \{1..n\}}(d_i, c_j)$
  end for
  {Recompute the centroid of each cluster until centroids do not change}
  while $m$ has changed do
    for all $i \in \{1..n\}$ do
      Recompute $c_i$ as the centroid of $\{d | m(d) = i\}$
    end for
    for all $d_i \in D$ do
      $m(d_i) = mindistance_{j \in \{1..n\}}(d_i, c_j)$
    end for
  end while
```

3.2 Cluster labeling

Assume that a categorization over a document collection is determined using an unsupervised approach, such as clustering. To present this categorization to users, it is convenient to label the individual categories with characteristic terms. These terms, called *category labels*, should characterize the content of the associated category with respect to the remaining categories. This implies that cluster labeling should *summarize* a category's content and that it should *discriminate* a category from the remaining categories. This section states desired properties for category labels and presents the Weighted Centroid Covering (WCC) algorithm (Stein and Meyer zu Eissen [2]) which generates such labels. It further makes a proposition for improvement of WCC (section 3.2.3). Finally, in Chapter 5 an evaluation of WCC for labeling of document clusters is offered, created using k-means algorithm.

3.2.1 Clustering and labeling

In their paper from 2009 [20], Carpineto et al. claimed that there is a difference between clustering of data sets, and clustering of search results, returned from search engines: *"in search results clustering description* *comes first"*. Thus, in contrast to the classical categorization of clustering algorithms previously outlined, they proposed a classification based on how well the clustering algorithms can generate sensible, comprehensive and compact cluster labels, and divided the algorithms in the following three categories:

Data-centric algorithms
Data-centric algorithms were the first ones to be implemented for cluster labeling. Here, clustering is done with some general clustering algorithm, and terms which are close to the cluster centroids[1] are nominated as cluster labels. Cluster centroids are a collection of independent terms from all documents not related to each other. In order to define the terms from cluster centroid, one uses a frequency measure, such as $tf - idf$ (eq. 2.2). WCC (see 3.2.3) is presented as an example for a data-centric clustering algorithm.

Description-aware algorithms
While data-centric algorithms don't consider word order, and process documents as "a bag of words", description-aware algorithms process ordered sequence of words (phrases) instead of terms as candidate labels, and assign labels during clustering process, not after it. Using monothetic term selection (documents are assigned to clusters based on a single feature), one nominates frequent phrases containing a noun as labels, which are interpretable to human users. Suffix Tree Clustering (STC) is an example of a description-aware algorithm, introduced by Zamir and Etzioni [21]. STC builds a tree from the most frequent phrases in documents by using the data structure *suffix tree* [21]. It groups documents that have common phrases under the same nodes of the tree. Thus, all documents below a given node contain the phrase at the node.

Description-centric algorithms
These algorithms operate on the principle *"description comes first"*, and in this sense are the opposite of the data-centric algorithms. The goal here is to find meaningful cluster labels. If there are no suitable labels found, the cluster has no value for the users (doesn't have a proper visualization), and is therefore discarded. Description-centric algorithms are mainly applied for clustering of search results (more information can be found in [20]).

[1]The *centroid* of a cluster is the mean of all vectors in the cluster.

3.2.2 Formal framework for cluster labeling algorithms

When an unsupervised approach is used to categorize a collection of documents, such as clustering, it is also necessary to label the categories defined, in order to present them to users. The category labels should characterize the given categories - labels should summarize category content, and should discriminate a category from other categories [2]. Until now, no uniformly agreed upon formal framework exists that defines the requirements for cluster labeling algorithms. There are publications which name desired properties for cluster labels ([22], [23]), but they all give informal descriptions. Stein and Meyer Zu Eissen [2] have given their definitions for desired properties of cluster labeling algorithms as a formal framework. The definition presented below is based on this source.

If we have an unstructured collection of documents D, a clustering algorithm creates a categorization for this collection in the form $C = \{c_1, c_2, ..., c_k\}$, where the sets c_i are subsets of D, and their union covers D: $\cup_{c_i \in C} c_i = D$. When applying a hierarchical clustering algorithm, this implies a cluster labeling hierarchy H_C over C. Then, H_C is a tree and its nodes are the categories in C, having one node as a root. Let $c_i, c_j \in C, c_i \neq c_j$ are two categories. If c_j is a child cluster of c_i, then c_i is closer to the root of the hierarchy H_C, and we write $c_i \succ c_j$.

For an existing categorization C, we therefore need a cluster labeling $Ł = \{l_1, l_2, ..., l_k\}$.

Let $T = \cup_{d \in D} T_d$ be the term set in the given document collection D. As defined by Stein and Meyer zu Eissen [2], a cluster labeling function $\tau : c \rightarrow Ł$ assigns a cluster label to each cluster $c \in C$, where $Ł_c \subset T$.

Thus, the following properties are desired for a cluster labeling function:

1. **Unique**
   Cluster labels should be unique. The same terms should not be assigned as cluster labels to more than one cluster, or no two labels should include the same terms from two different clusters.

$$\forall_{\substack{c_i, c_j \in C, \\ c_i \neq c_j}} : \tau(C_i) \cap \tau(C_j) = 0 \qquad (3.1)$$

2. **Summarizing**
   If possible, the label of a cluster c should contain at least one term

t from each document $d \in c$. Terms occurring in all documents, which are part of the cluster, represent it better than terms that occur only in few documents.

$$\forall_{c \in C}, \forall_{d \in c} : \tau c \cap T_d \neq 0 \tag{3.2}$$

where T_d is the set of terms in document d.

3. **Discriminating**
Apart from summarizing, terms in labels should be discriminating. They should contribute to discriminate the cluster from other clusters, i.e. the same terms should be present in a considerably smaller set of documents in the other clusters. Discriminating means that in a cluster label c_i, there exists a term t whose frequency of occurrence is relatively higher in the documents belonging to c_i as compared to the documents in other clusters.

$$\forall_{\substack{c_i, c_j \in C \\ c_i \neq c_j}} \exists_{t \in T c_i} : \frac{tf_{c_i}(t)}{|c_i|} \ll \frac{tf_{c_j}(t)}{|c_j|} \tag{3.3}$$

Here, T_{c_i} is the term set in category c_i, and $tf_{c_i}(t)$ is the term frequency of term t in category c_i, or the sum of $tf(t)$ in all documents in cluster $c_i : tf_{c_i}(t) = \sum_{d \in c_i} tf_d(t)$.

4. **Expressive**
Terms forming a label of cluster c should have highest frequency of occurrence in the documents from c:

$$\forall_{c \in C} \forall_{d \in c} \forall_{t \in T_d} : tf_d(t) \leq tf_d(t'), t' \in \tau(c) \tag{3.4}$$

Here, $tf_d(t)$ is the term frequency of occurrence of term t in document d.

5. **Contiguous**
This property holds for consecutive terms, for example belonging to a phrase, idiom or name. Such cluster labels, containing consecutive terms, are more understandable to human users.

$$\forall_{c \in C} \exists_{\substack{t, t' \in \tau(c), \\ c_i \neq c_j}} \forall_{d \in c} \exists_{t_i, t_{i+1} \in T_d} : t_i = t \wedge t_{i+1} = t' \tag{3.5}$$

6. **Hierarchically consistent** If a c_j is a child cluster or a specialization of c_i, such that : $c_i \succ c_j$, this specialization is reflected also in the terms of the corresponding labels:

$$\forall_{\substack{c_i, c_j \in C, \\ c_i \neq c_j}} : c_i \succ c_j \Rightarrow P(t_i|t_j) = 1 \wedge P(t_j|t_i) < 1, \tag{3.6}$$

where $t_i \in \tau(c_i)$ and $t_j \in \tau(c_j)$, and P is the conditional probability that terms t_i, t_j occur in clusters c_i, c_j. This property is required only when using a hierarchical clustering algorithm (e.g. STC [21]).

7. **Irredundant**
   Irredundancy complements the property *unique*. Terms which are synonyms should be avoided in cluster labels.

$$\forall_{c\in C}, \forall_{\substack{t,t'\in\tau c,\\ t\neq t'}} : \text{ t and t' are not synonyms} \tag{3.7}$$

The properties stated above describe ideal conditions and can only be approximated in the real world. One needs external knowledge (e.g. an ontology), in order to fulfill requirements *hierarchical consistency* and *irredundancy*.

3.2.3 Weighted Centroid Covering

Weighted Centroid Covering (WCC) was introduced by Stein and Meyer zu Eissen [2]. It is a data-centric algorithm for cluster labeling, in which labels are generated from sets of frequently occurring terms.

If D are all documents in a collection, which contains a set of terms T, and $t \in T$ is a term from T, and $C = \{c_1, c_2, ..., c_{|C|}\}$ is a categorization over the set D, then we can define a function κ as follows: let $\kappa : T \times \{c_1, c_2, ..., c_{|C|}\} \to C$ is a function with $\kappa(t, i) = c$ iff c is the cluster with ith frequent occurrence of term t [2]. As an example, if t occurs most frequently in a give cluster, we can write $\kappa(t, 1)$, and if it occurs least frequently, we write $\kappa(t, |C|)$.

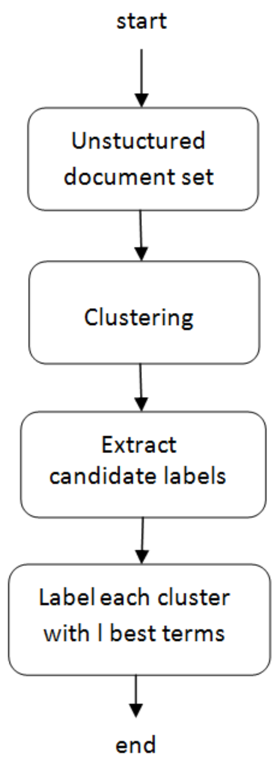

Figure 3.1: Cluster labeling using Weighted Centroid Covering algorithm, where l is the number of terms per label

The WCC algorithm consists of three stages. As input to the algorithm, a clustering C is given:

1. For all terms in the input clusters, it saves the k most frequent occurrences of each word with its term frequency, and the cluster in which it occurs, to a data structure (say a vector Ł).
2. It sorts vector Ł which stores tuples (k, $tf(term)$, term, cluster) in a descending order, based on the frequency of a term in a given cluster $tf_c(t)$;
3. It assigns l terms to each cluster as labels. Therefore, in the end each cluster has a label of l terms, assigned to it in a Round-Robin-like manner.

The complexity of WCC is $O(|T\ log(|T|))$. A pseudo code of the algorithm is given as algorithm 3.2. Additionally, the main steps of WCC can be seen graphically in fig. 3.1.

Algorithm 3.2 Weighted Centroid Covering algorithm for cluster labeling

```
Input: $C$ - clustering
  $l$ - number of terms per label
  $k$ - maximum occurrence of the same term in different labels
Output: $\tau$ - labeling function
  $Ł = 0$
  foreach $c$ in $C$ do
  $\tau(c) = 0$;
  end foreach
  foreach $t$ in $T$ do
  for $i = 1$ to $k$ do
    compute $c = \kappa(t, i)$ from $C$;
    add tuple $\langle t, tf_c(t)\rangle$ to Ł;
  end for
  end foreach
  sort Ł according to descending term frequencies;
  for $labelcount = 1$ to $l$ do
    $assigned = 0$;
    $j = 1$;
    while $assigned < |C|$ and $j \leq |Ł|$ do
      let $t_j = \langle t, tf_c(t)\rangle$ be $j^{th}$ tuple of Ł;
      if $|\tau(c)| < labelcount$ then
        $\tau(c) = \tau(c) \cup \{t\}$;
        delete $t_j$ from Ł;
        $assigned = assiged + 1$;
      end if
      $j = j + 1$;
    end while
  end for
  foreach $c$ in $C$ do
  do sort $\tau(c)$
  end foreach
  return $\tau$;
```

3.3 Cluster labeling using external knowledge

WCC was previously presented as an algorithm for cluster labeling. It nominates cluster labels by selecting as candidate labels the terms with highest frequency of occurrence from the corresponding clusters. However, the algorithm nominates labels based only on cluster content and term frequencies of words in the document collection. In order to improve labeling for users, noun phrases may be used as labels, as they are intuitively understood by humans, and are more descriptive than single terms (Weiss [23]). Another improvement of cluster labeling can be made by using external knowledge to generate more meaningful labels. Carmel et al.([24]) reported promising results using Wikipedia[2] as a source of an external knowledge. Ontologies can also be used as a knowledge source during cluster label generation. Stein and Meyer zu Eissen ([3]) propose using a hierarchical classifier, in order to assign clusters to ontology nodes, and thus nominate cluster labels from the ontology, however, their proposal includes a text classification problem.

Based on the ideas by Stein and Meyer zu Eissen ([3]), and Mugo ([25]), a proposal for improving WCC is made using ontologies as external knowledge, and adding semantic annotations to the document collection. Using the semantic annotations in a cluster of documents, candidate labels can be nominated from both the annotations and the terms, which occur most frequently in clusters. Thus, if the ontology fails to provide suitable labels, the label candidates proposed by WCC can be used.

In order to present the approach, first a short overview on ontologies is given. In order to gain more detailed insight, refer to the given reference literature, as it is not the main topic of this work to investigate into semantic knowledge and ontologies.

3.3.1 Ontology as a source of external knowledge

Formal models of the world can be used to provide formal semantics, or machine-interpretable meaning to information, stored as documents, web pages, databases. When models are intended to represent a shared conceptualization, such as a classification, they are called ontologies [26]. Or if the classical definition from Gruber [27] is used: Ontologies are explicit specifications of the conceptualizations at a semantic level.

[2]http://www.wikipedia.org/

Formal ontologies are represented in a logical formalism which allows for indexing, querying, and referencing purposes over non-ontological datasets, such as documents, databases [23]. An example for a logical formalism is the ontology language Web Ontology Language (OWL)[3].

An ontology is characterized by the following tuple (ordered list):

$$O =< C, R, I, A > \tag{3.8}$$

In the equation above C is a set of classes which represent the concepts from a given domain. Examples for concepts can be databases, resources, repositories. R is a set of relations, also called properties or predicates. These relations are valid between instances of the classes from C. As an example: Resource *isStoredIn* Repository. I is a set of instances, having as elements instances to one or more classes, which can also be linked to other instances or values. For example: *manual2* isStoredIn *onlineRepository1*. And finally, A is a set of axioms, or rules in a logical form about the domain which the ontology describes.

According to the formal language used for their creation, ontologies can be divided into *lightweight* and *heavyweight*. Heavyweight ontologies possess highly predictive and restrictive concept definitions; as compared to them, lightweight ontologies allow for more efficient and scalable reasoning [23]. Based on the conceptualization that they describe, the ontologies can be divided further into *upper-level*, which model general knowledge and *domain* ontologies, specific to a certain domain (e.g. the domain of CoreMedia Content Management System (CMS) in Appendix A.1).

The process of semantic annotation is to add meta data to documents using an ontology. After the annotation, the documents contain additional set of information which was not specified by the initial author of the document. This additional information based on an ontology allows for interpretation of the meaning in texts. Thus, if cluster labels are generated based on semantic annotations, they will better reflect the meaning in the cluster, than labels which only contain the. e most frequent terms. Mugo [25] provides detailed overview of the annotation process.

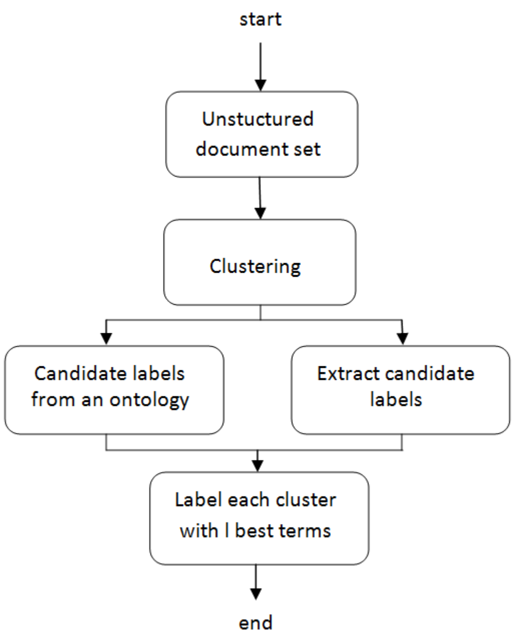
start
Unstuctured document set
Clustering
Candidate labels from an ontology
Extract candidate labels
Label each cluster with l best terms
end

Figure 3.2: Cluster labeling using Weighted Centroid Covering algorithm, augmented by an ontology

3.3.2 Weighted Centroid Covering augmented by an ontology

If a document collection is annotated using an ontology, the semantic annotations can be used as candidate labels for the documents in a cluster, as they reflect the "meaning" of the texts. Candidate labels can be nominated from both the semantic annotations in a cluster, and the most frequent terms (fig. 3.2). Thus, selected ontological annotations will "compete" with inner terms for serving as the cluster labels. For the case when the annotations fails to cover the cluster content, the most frequent terms should be used as labels.

The proposed solution has the advantage of simplicity as compared to the proposal of Stein and Meyer zu Eissen ([3])), because it doesn't add the complexity of a text classification problem to cluster labeling. However, a traditional drawback of semantic annotations is that so far, there is no fully automatic annotation tool, and manual work is required during the process of document annotation ([25]).

[3]http://www.w3.org/TR/owl-features/, accessed December, 2010

Due to time constrains, no experimental results can be provided on running WCC with external knowledge. This remains for further work.

Chapter 4

Tag Clouds

Tagging is the activity of associating one or more key words or phrases to a resource. A tag is a label or a note that makes it easier for users to find documents, web-pages, videos, images or other resources. Tag clouds are generated from a collection of tags, and one of their first uses was for annotating pictures, as a part of the Flickr[1] website in 2004 [28]. Tags created by different users form a folksonomy, which is a flat user-generated classification of a collection of resources. Folksonomies are part of *Web 2.0*, the collection of tools for retrieval, deployment, representation and production of information. In Web 2.0 it is no longer organizations, web designers or authors who generate content - every user can do so. The heavy growth of user-generated content, a part of the so called *information flood*, increases the demand for suitable methods and facilities for storage and retrieval of this content. In order to meet those demands, collaborative information services have been developed, like social bookmarking, photo sharing and video sharing, where users index their own information resources with their tags. This indirect cooperation of users creates a folksonomy for each collaborative information services comprised of each individuals user's tags. Using this folksonomy, all users may then access the resources of the information service in question.

Several concepts are defined below, which are used in this chapter:

Tagging, which is one of the defining characteristics of Web 2.0 services, allows users to collectively classify and find information. Tagging is *manual* (collaborative), done in social bookmarking applications for example, or *automatically generated*, e.g., based on the frequency of term occurrence in a document collection.

Tag clouds are visual interfaces for IR. They provide a global contex-

[1]`http://www.flickr.com/`, accessed December, 2010

tual view of tags assigned to resources (or documents) in the collection (fig. 4.1).

Collaborative tagging systems, also called *social bookmarking* systems, are used to organize, browse and share personal collections of resources by introducing simple meta data about these resources (fig. 4.2).

Folksonomies are flat, user-generated classifications of resources [28].

Web 2.0. Social bookmarking systems are a part of *Web 2.0.* Web 2.0 spans all activities and technical requirements that allow users of the World Wide Web to self-publish content, in the form of profiles, bookmarks, photos, videos, posts etc. and to make it accessible to other users, as well as to communicate with them [28].

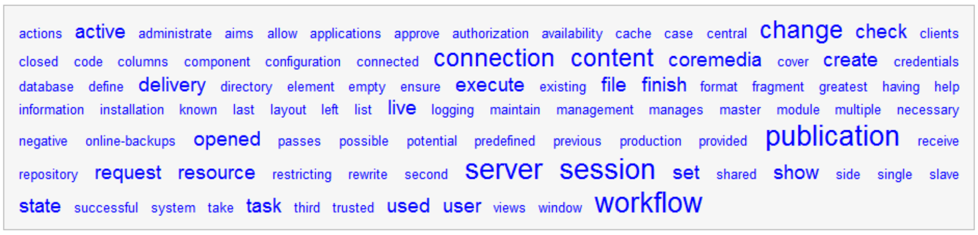
actions active administrate aims allow applications approve authorization availability cache case central change check clients
closed code columns component configuration connected connection content coremedia cover create credentials
database define delivery directory element empty ensure execute existing file finish format fragment greatest having help
information installation known last layout left list live logging maintain management manages master module multiple necessary
negative online-backups opened passes possible potential predefined previous production provided publication receive
repository request resource restricting rewrite second server session set shared show side single slave
state successful system take task third trusted used user views window workflow

Figure 4.1: A tag cloud, where tags have been selected and visually weighted according to their frequency of use.

4.1 Introduction

Tag clouds are simple visualization interfaces. They are wide-spread as a part of Web 2.0. Several types of tag clouds exist:

- Tag clouds generated over partial lists, such as search results. The size of tags depends on the frequency of occurrence of the corresponding terms, measured for example using $tf - idf$ measure (section 2.2.2).
- *Collaborative* tag clouds, where tag frequency and size are generated over all documents in the set D. Collaborative tag clouds present the main concepts in the collection, where the size of tags is defined by some measure, such as frequency of occurrence.

- *Categorical tag clouds*, where the size of tags reflects the size of the corresponding category.

Tag clouds can be further divided into *manual* and *automatic*, depending on the way they are generated. Manual tags are created for example in social bookmarking applications by users, while automatic tag clouds are generated based on a collection of textual resources, and a frequency measure. Clouds enhance the visualization of information, contained in a collection of resources. According to Smith [29] and Peters [28] tagging has the following main application areas:

- *Information retrieval*, e.g. in the online services Last.fm[2] or Engineering village[3], where tag clouds enhance IR, and are used to retrieve resources. Here are also included tag clouds used in e-commerce services, such as Amazon[4].
- *Online libraries* use tag clouds to present a book content as a collection of tags, representing the main concepts in a book, as used in Library Thing[5], or to search for books under a collection of categories, presented as a tag cloud, as used in Wiley Online Library[6]. These are just two examples out of many more.

- *Social bookmarking.* Delicious[7] offers tagging for sharing bookmarks to online resources, and the popular Facebook[8] uses tagging for sharing photos, videos or music.

- As a part of *games with a purpose*, or *GWAP*[9], where tagging is used for improving computer programs, such as programs performing image recognition tasks, or for tagging audio or image files.

As tags and tag clouds are heavily used in collaborative tagging, an example for tag cloud use is given in the context of social bookmarking

[2]`http://www.lastfm.de/`, accessed December, 2010
[3]`http://www.engineeringvillage.org/`, accessed December, 2010
[4]`http://www.amazon.com/gp/tagging/cloud/`, accessed December, 2010
[5]`http://www.librarything.com/`, accessed December, 2010
[6]`http://onlinelibrary.wiley.com`
[7]`http://www.delicious.com/`, accessed December, 2010
[8]`http://www.facebook.com/`, accessed December, 2010
[9]`http://www.gwap.com/`, accessed December, 2010

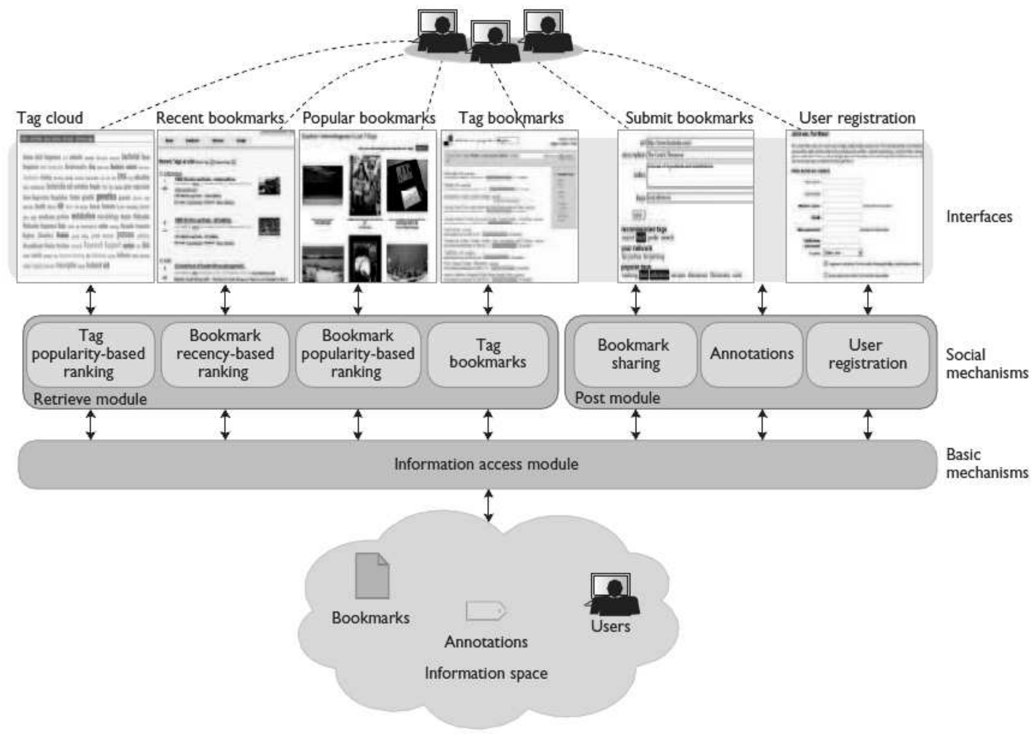

Figure 4.2: Tag clouds in applications for collaborative information services. Source: Heymann, Koutrika, Garcia-Molina(2007, [30])

software (fig. 4.2). In this application users have the roles of both content creators and content consumers. As consumers, they can use tag clouds for retrieval of resources, and visualization. In fig. 4.2, a tag cloud contains the most commonly occurring tags in the system, and tag bookmarks contain for a given tag t, a list of the most recently posted URLs annotated with t.

The extensive amount of people involved in the process of resource tagging can overcome to some extent the flood of information, generated on a daily basis by users. With respect to social bookmarking, Clay Shirky[10], a professor at New York University, says: *"The only group that can categorize everything is everybody."*. However, social tagging has certain drawbacks. There is criticism about the quality of tagging, as it is usually done by laymen, and privacy issues arise, concerning the tagged content [28].

[10]`http://shirky.com/writings/ontology_overrated.html`, accessed December, 2010

4.2 Tag clouds generation

Tag clouds are generated from a collection of tags, and corresponding weights associated with them, which measure their ”importance” by defining the tag size in the cloud. The implementation usually includes a preprocessing phase, as already discussed in section 2.2, such as text parsing, filtering out of stop words, numbers, punctuation. As tag size is defined by frequency of occurrence, for small frequencies the size is normalized to one. For larger values, a linear normalization is applied, where the weight t_i of a tag is mapped to a size scale of 1 through f, where t_{min} and t_{max} are specifying the range of available weights. Thus, the displayed font size of a tag t_i is defined by:

$$f_i = \lceil \frac{f_{max}.(t_i - t_{min})}{(t_{max} - t_{min})} \rceil \text{ for } t_i > t_{min}; \text{ else } f_i = 1 \tag{4.1}$$

where f_i is the displayed font-size, f_{max} is the maximum font-size, t_i is the count of tag t, and t_{min}, t_{max} are limits for minimum and maximum count that a term t can have.

4.2.1 Tag cloud layout

The traditional tag cloud layout is alphabetical. Tag clouds aggregate a statistics of the tag-usage. They are typically sent as in-line HTML to browsers. Since the font size of a displayed tag is usually used to show the relative importance or frequency of the tag, a typical tag cloud contains large and small text interspersed. The following kinds of layout are common:

- Sequential layout, with either a horizontal or vertical arrangement of tags, sorted alphabetically or by some other criteria (e.g., popularity, chronology, etc.).
- Circular layout, with the most popular tags in the center and tags with decreasing popularities towards the borders (or vice versa).
- Clustered layout, in which the distance between tags follows a certain clustering criteria (e.g., semantic relatedness) and related tags are positioned in close proximity [3, 6].

4.3 Existing implementations

In this work, a semantic space is constructed over a set of documents using LSA (Chapter 2), and a tag cloud is used to visualize the main concepts in search results, retrieved from this space. Similar implementations of tag clouds exist, used to enhance IR tasks. Examples of such applications are given below.

Yippy[11] is a tool that visualizes topics based on search queries as a tag cloud, previously known as Clusty. Created by Vivisimo[12], it offers classification of search results.

Google[13] has developed their **Wonder wheel** search visualization tool, in order to display search results as a categorical tag cloud. The tags in Google's Wonder wheel represent categories in the set of search result documents.

Opinion Crawl[14] is web sentiment analysis application (it analyzes the "opinion" of users). It generates a concept cloud from daily scanned blogs, web site articles, and is developed at Semantic Engines LLC.

SenseBot Search Results is another tool from Semantic Engines LLC. It is a plugin for Mozilla Firefox browser that generates a tag cloud of the main concepts returned as search results from Google, included as a part of the SenseBot semantic search engine[15].

These are just a few examples of tag cloud usage presented here, as they are similar to the implementation purpose of the tag cloud developed in this work.

4.4 Conclusion

In this chapter tag clouds were presented as visualization interfaces for enhancing IR services, and the use of tag clouds was over viewed in the context of social bookmarking applications and commercial retrieval systems. In the next chapter, a tag cloud implementation is given, which

[11] `http://cloud.yippy.com/`, accessed Dezember, 2010
[12] `http://vivisimo.com/`, accessed Dezember, 2010
[13] `http://www.google.com/`, accessed December, 2010
[14] `http://www.opinioncrawl.com/`, accessed December, 2010
[15] `http://www.sensebot.net/`, accessed December, 2010

was developed as a part this work, and its function as a part of an IR process is explained.

Chapter 5

Implementation

This chapter describes the implementation part of the thesis work. After presenting in Chapter 2 the theoretical basis behind LSA, and in Chapter 3 cluster labeling, theoretical application and specific implementation decisions are discussed here. All software tools and libraries which were used are pointed out, and code snipplets are given. Then in Chapter 6, test results are shown, and evaluation of the implementation is made.

5.1 Tag cloud summarizer

The prototype implementation is a web application, which outputs a tag cloud based on users' queries. Initially, a preprocessing of the document set is done, then a semantic space is constructed by running LSA and performing dimensionality reduction during SVD (refer to Chapter 2 for the terminology). The initial two stages are performed offline. Once the document set is indexed by a term-document matrix, queries can be made by users. The terms in the documents closest to the queries in the semantic space, are input to the tag cloud. Querying and tag cloud generation are performed interactively online.

The desicion to implement the prototype as a web application is due to two factors. Firstly, tag clouds widely spread in online systems, and thus mainly used online. And secondly, the prototype should be implemented in the online documentation system DocMachine[1] at CoreMedia AG, Hamburg, and should also be available online.

[1] `https://documentation.coremedia.com/`

Preprocessing

The documents used for evaluation are a part of the online documentation at CoreMedia AG, Hamburg[2]. Documents are stored as XML files, and CoreMedia CMS Unified API[3] is used to access the plain text, as shown in Listing B.1. Before executing LSA in order to construct a semantic space from a document collection, a preprocessing the text corpus is made. Stop words, puncutuation and numbers are removed. No stemming is done, as terms from the semantic space are later used to generate a tag cloud.

LSA

The LSA implementation uses the opensource Java-based S-Space library (see section 5.3). Partial code for LSA is given in listing B.2. The complete source code can be found in the online repository where this work in available online[4]).

Querying the semantic space

After the semantic space has been constructed, and dimensionality reduction has taken place, queries can be made to find the documents closest to a given query, or the terms respectively, by querying the documents or term space. As a reminder, the term space consists of the matrix product: $U * \Sigma$, and the document space of the product: $\Sigma * V^t$ (fig. 2.3). In listing B.3 the source code for querying the semantic space is given.

Tag cloud generation

A tag cloud is generated using a collection of terms (or tags) with their corresponding weights. The weights are just the normalized term frequencies after a dimensionality reduction took place during LSA. In listing B.4 is given the source code used for generating the tag cloud.
The Tag cloud summarizer is a tool that should aid users to obtain a quick overview of the main concepts contained in search results they receive. It should be used by users, and can be evaluated best by them. In Chapter **??**chapter:evaluation an simple evaluation based on user feedback is provided.

[2] `https://documentation.coremedia.com/`

[3] `https://documentation.coremedia.com/servlet/content/241548?language=en&version=5.2&book=coremedia:///cap/content/241548`

[4] `https://github.com/angievelinska/Tag-Cloud-Summarizer/tree/master/summarizer/src/main/java/edu/tuhh/summarizer/lsa`

5.2 Cluster labeling

The algorithm for cluster labeling Weighted Centroid Covering is implemented in Java, and can be seen in listing B.5.
For the ontology development, we used Protege Ontology Editor 4.1[5] from Stanfornd University. The ontology is a light-weight domain ontology developed in OWL for CoreMedia CMS domain.

5.3 Tools and libraries used in this work

Repository

This thesis work is available online hosted in a reporitory under GitHub. Prototype implementation[6], project report[7] and LaTeX template[8] can be downloaded freely.

LSA and SVD

In this work the popular SVD C library[9] is used, created by Doug Rohde at the Massachusetts Institute of Technology. The implementation developed as a part of this project is Java-based, therefore for matrix computations the open source SVDLIBJ[10] library is used, which is a Java-based port of SVD C, made available by Adrian Kuhn and David Erni at the University of Bern.

k-means clustering algorithm

Cluto clustering library[11] is used as an implementation of the k-means clustering algorithm.

[5]`http://protege.stanford.edu/`, accessed December, 2010

[6]`https://github.com/angievelinska/Tag-Cloud-Summarizer`

[7]`https://github.com/angievelinska/Tag-Cloud-Summarizer/raw/master/report/thesis.pdf`

[8]`https://github.com/angievelinska/Tag-Cloud-Summarizer/tree/master/report`

[9]`http://tedlab.mit.edu/~dr/SVDLIBC/`, accessed December, 2010

[10]`http://bender.unibe.ch/svn/codemap/Archive/svdlibj/`, accessed December, 2010

[11]`http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview`, accessed October, 2010

Information retrieval

S-Space project created by Jurgensd and Stevens ([31]) was used for constructing a semantic space of the text collection used for evaluation.

Tag cloud

The tag cloud used in this work is based on Opencloud[12] project, authored by Marco Cavallo.

Search and retrieval of search results

Lucene[13] is an open source search engine library, distributed by the Apache Software Foundation. It is used for indexing, query parsing and retrieval of search results.

Building and deployment

Maven[14] is a software management tool, provided by Apache Software Foundation, which is used for building and testing the implementation prototype, and for deploying the web application module.

[12]`http://opencloud.mcavallo.org/`, accessed December, 2010

[13]`http://lucene.apache.org/java/3_0_2/`, accessed December, 2010

[14]`http://maven.apache.org/`

Chapter 6

Evaluation

Evaluation in IR is based on measures. As this work implements techniques for information retrieval, several wide-spread measures are described next, used commonly in IR systems to evaluate performance. Then, the implemented tests are presented, and finally the obtained test results are discussed.

6.1 Measures for evaluation

In order to perform an evaluation of IR systems, one needs in the simplest case:

- a document collection,
- a test suite of information needs, or a set of queries,
- a set of relevance judgments to define for each query-document pair weather the document is relevant or not relevant for the query posed.

Relevance is assessed relative to an information need, not a query. A document is considered relevant if it addresses the users' information need, not only if it contains all the words given in the query (Manning et al. [32]). When short queries (composed of one word for example) are posted, it is difficult to define the information need of users. But nevertheless, the user has one, and the retrieved search results are evaluated based on his or her information need.

The two most frequent and basic measures for evaluation of IR systems are *precision* and *recall* [7]. Recall shows the ability of a retrieval system to present all relevant items, while precision shows its ability to present only relevant items. For the simple case of a system which returns a result set of document to a query, they are defined:

$$\textbf{precision} = \frac{\text{number of relevant documents retrieved}}{\text{total number of relevant documents}} \tag{6.1}$$

$$\textbf{recall} = \frac{\text{number of relevant documents retrieved}}{\text{total number of retrieved documents}} \tag{6.2}$$

6.2 Document set used for evaluation

In order to evaluate the retrieval and visualization of main concepts in a document corpus by the Tag cloud summarizer tool, and cluster labeling by WCC, an evaluation document set was prepared, which contains documents in three categories (see Appendix C). Due to time constraints, only a simple usability evaluation was made, using the queries from Appendix C.

6.3 Evaluation of the Tag cloud summarizer

Dimensionality reduction parameter in LSA

Certain factors influence the performance of LSA, as already discussed in section 2.6.

- *Preprocessing* - before constructing a semantic space using LSA, stop words were removed from the evaluation document set. Stemming is not done, as terms from the semantic space are later used for the tag cloud generation. Stop words and punctuation are removed. The data model used in DocMachine stores the documents as XML files, and CoreMedia CMS Unified API[1] is used to strip the markup, in order to access the plain text (listing B.1).

- *Frequency matrix transformation* - log-entropy weight functions (section 2.2.2) were chosen, due to their better performance as compared to other measures (reported by Nakov et al. [14]).

[1]`https://documentation.coremedia.com/servlet/content/241548?language=en&version=5.2&book=coremedia:///cap/content/241548`

Document set overview		
Category Id	Category	Document #
1	session, connection	5
2	server	5
3	publication, workflow	5

Table 6.1: Overview of the constructed document set

- *Dimensionality reduction parameter* ***k*** - it is defined based on the experiments carried out using the document set for evaluation from CoreMedia CMS domain, described in section **??**.

LSA evaluation results

LSI can be viewed as soft clustering by interpreting SOFT CLUSTERING each dimension of the reduced space as a cluster and the value that a document has on that dimension as its fractional membership in that cluster.

6.4 Evaluation of cluster labeling algorithm WCC

The experiments in this study conducted several objective measures to validate the performance of the described algorithm WCC.

Evaluating the quality of cluster labeling is a non-trivial task and the most suitable evaluation is judged by user's experiences.

The algorithm was implemented in Java on a a 64-bit Windows 7 Ultimate workstation with an Intel Core 2 Duo P8600 @ 2,40 GHz using 4.00 GB RAM.

Document set

In order to test a cluster labeling algorithm, we need a test data set containing predefined categories. WCC is tested using the evaluation data set with three predefined categories selected from the whole collection used for evaluation of LSA implementation.

The document set used for evaluation consists of 3 categories, and 15 documents - each category has 5 documents, and the documents have different length. Table C.2 gives an overview of the constructed data sets. The document preprocessing involves parsing and stop word removal according to standard stop word lists.

Data preprocessing
The evaluation document set needs to be preprocessed before it can be used in the experiments. Analogous to the preprocessing used for the whole data set, punctuation and stop-words were removed. Terms in texts were not stemmed, as they are used as cluster labels, and stemming the terms would lead to worse user experience.

6.4.1 Cluster labeling evaluation

To evaluate the quality of WCC, an evaluation document set of 15 texts in 3 categories was prepared.

To evaluate the WCC algorithm, we prepared a document set with author-defined keywords. The evaluation idea is to cluster the documents, label the resulting clustering, compute f1-f4, and measure to which extent the identified topic labels appear in the keyword list of at least one document of the associated cluster. ???
To measure the extent, standard precision and recall values can be chosen. Note, however, that the precision value is more important for the scenario tested in this work since we nominate only ????(**TODO**: test with different "l" ???) words per cluster label to be presented to users.

Chapter 7

Conclusion and outlook

Modern search applications process huge amount of information, and have to cope with the ambiguity of search queries that the users enter in search engines[1]. To address the problems of word ambiguity, LSA was introduced in Chapter 2 as an IR technique. Clustering is another method used largely in IR systems to group together similar documents, and thus present large amounts of search results in categories, which are better suited for browsing, than long search result lists. An important part of clutering used in search applications is how to label the clusters in a suitable and summarizing way, in order to increase clustering usability for humar users. Therefore, in Chapter 3 the algorithm for cluster labeling WCC was reviewed. It nominates cluster labels from the terms in clusters which occur most frequently. However, clustering and WCC examine neither the semantic structure in the text collection they process, nor the ”meaning” of texts. As stated in the formal framework for cluster labels given in section 3.2.2, labels need to be summarizing and hierarchically consistent. Therefore, an improvement of WCC was proposed using external knowledge from an ontology during cluster label nomination.

Search applications process huge amounts of information. Presenting this information to the users is an important part of these applications, and can improve their usability. We applied clustering as a method to organize search results into browsable groups of documents. In order to present these prepared groups of documents to the end users, informative and summarizing cluster labels are needed. Therefore, we evaluated the implementation of WCC cluster labeling method, and outlined a proposal for its improvement.

[1] `http://www.hitwise.com/index.php/us/press-center/press-releases/2009/google-searches-apr-09/`, accessed December, 2010

With respect to the problem of how to present many search results to users, a tag cloud is implemented, which summarizes the main concepts in search results, thus used as a visualization mean.

TODO: check pavel kazakov 2008 - nice conclusion and outlook.

7.1 Future Work

The goals set for this project and given in section 1.3 were reached. The project can still be developed further. Below are outlined several proposals for future work.

7.1.1 Implementation

! Implement the prototype as a part of DocMachine[2] - the online documentation system at CoreMedia AG, Hamburg.

LSA

One of the major drawbacks in implementing LSA is that computing SVD for large matrices when applied to large document sets is computationally expensive. It has been stated that it is possible to compute SVD in an incremental manner, and with reduced resources via neural-network like approach [33]. However, currently there is no Java-based implementation of this algorithm. As this project is developed using Java, it remains as future work to implement the fast incremental SVD computation, proposed by Brand [33] in Java, in order to use it in this work.

Cluster labeling

Other clustering methods can be investigated, such as STC, which preserves the word order in documents, by presenting them in the form of a tree. Cluster labeling using STC involves also using phrases as candidate labels, instead of separate terms, which improves usability, as compared to the investigates WCC algorithm.

[2] `https://documentation.coremedia.com/`

Cluster labeling using external knowledge

Due to time constraints, it remains for future work to evaluate and present experimental results on running WCC algorithm for cluster labeling with external knowledge from an ontology.

7.1.2 Testing

A larger set of document can be used for evaluation and testing of our implementation. In this work we carried out tests on a document set consisting only of 15 documents in 3 categories. Research has shown, however, that LSA perform better when applied to document collections above 3000 documents, each of which larger than ≈ 60 words, testing our implementation on a larger document collection remains as future work.

Acronyms

CMS Content Management System.

HAC Hierarchical Agglomerative Clustering.

IR Information Retrieval.

LSA Latent Semantic Analysis.

OWL Web Ontology Language.

STC Suffix Tree Clustering.

SVD Singular Value Decomposition.

VSM Vector Space Model.

WCC Weighted Centroid Covering.

Appendix A

Ontology

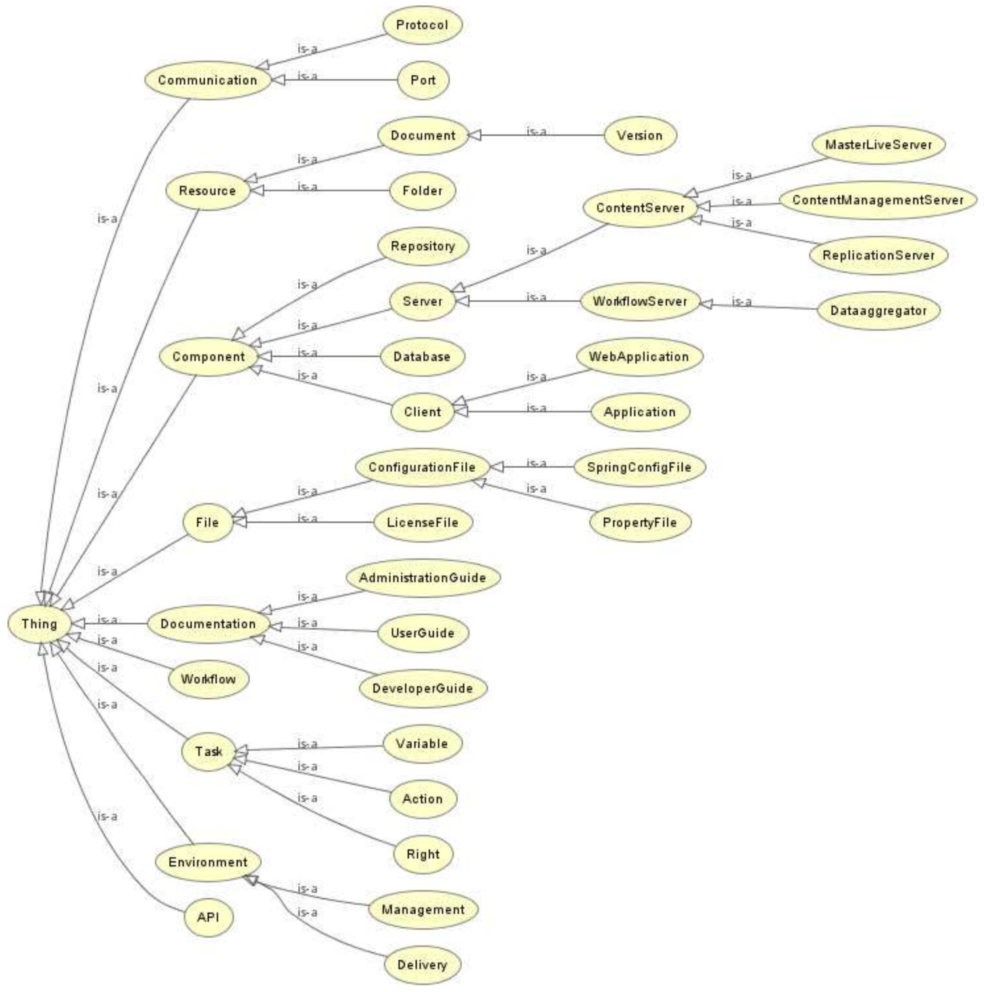

Figure A.1: Upper level ontology for CoreMedia CMS domain

Appendix B

Source code

Listing B.1: Preprocessing of the document collection - markup is stripped from plain text

```java
public class ProcessDocuments extends AbstractDocuments{
  private static Logger log = Logger.getLogger(
      ProcessDocuments.class);
  private static final String path = "summarizer/data/
      output/";
  private static final String extension = ".txt";

  @Override
  protected void processText(Content article){
     String id = "";
     Markup markup = null;

     for (Content textElement : article.getLinks("Articles
        ")){
      if (textElement.getType().getName().equals("Text")){
        List<Content> languages = textElement.getLinks("
            Language");

        if (languages.size()>0 &&
                languages.get(0).getString("Name").
                    equalsIgnoreCase("en")){
          markup = textElement.getMarkup("Content");
          id = textElement.getId();
          serialize(id, markup);
        }
      }
    }
  }

   private void serialize(String id, Markup markup){
     StringBuffer sb = new  StringBuffer();
     sb.append(id.substring(id.lastIndexOf("/")+1));
     sb.append(extension);

     File outputPath = new File(path) ;
     if(!outputPath.exists()){
       outputPath.mkdir();
     }

     File outputFile = new File(sb.toString());
     if (!outputFile.exists()){
       outputFile = new File(outputPath,sb.toString());
     }

     try {
       BufferedWriter writer = new BufferedWriter(new
           FileWriter(outputFile));
       String plainText = asPlainText(markup);
       writer.write(plainText);
       writer.close();

     } catch (IOException ex){
       log.error(ex);
     }
     log.info("*** File created: "+sb.toString());
   }

  private String asPlainText(Markup markup){
    if (markup ==  null) return "";

    StringWriter writer = new StringWriter();
    PlaintextSerializer serializer = new
        PlaintextSerializer(writer);
    markup.writeOn(serializer);

    return writer.toString();
  }
}
```

Listing B.2: Latent Semantic Analysis

```java
public class LSA {
  private static Logger log = Logger.getLogger(LSA.class);

  public void runLSA() {
      Properties props = new PropertiesLoader().
          loadProperties();
      IteratorFactory.setProperties(lsaprops);
      int noOfThreads = Runtime.getRuntime().
          availableProcessors();

      long start = System.currentTimeMillis();
      LatentSemanticAnalysis sspace = null;
      try {
        // initialize the semantic space
        sspace = new LatentSemanticAnalysis();
        Iterator<Document> iter = new
            OneLinePerDocumentIterator(props.getProperty("
            docFile"));

        // dimensionality reduction and SVD
        processDocumentsAndSpace(sspace, iter, noOfThreads
            , props);

        // save the constructed term space - after SVD
            these are U * Sigma matrices.
        File output = initOutputFile(props, "termSpace.
            sspace");
        SemanticSpaceIO.save(sspace, output,
            SemanticSpaceIO.SSpaceFormat.TEXT);
        log.info("Semantic space is saved after SVD
            reduction.");
      } catch (IOException e) {
        e.printStackTrace();
      } catch (InterruptedException ex) {
        ex.printStackTrace();
      }
      saveDocumentSpace(sspace, props);

      long end = System.currentTimeMillis();
      log.info("LSA took " + (end - start) + "ms to index
          the document collection.");
    }

  protected void processDocumentsAndSpace(SemanticSpace
      space,
                                         Iterator<
                                             Document>
                                             iter,
                                         int noOfThreads,
                                          Properties props
                                             )
          throws IOException, InterruptedException {

    parseDocsMultiThreaded(space, iter, noOfThreads);
    space.processSpace(props);
  }

  protected void parseDocsMultiThreaded(final
     SemanticSpace space,
                                        final Iterator<
                                           Document> iter,
                                        int noThreads)
          throws IOException, InterruptedException {
    Collection<Thread> threads = new LinkedList<Thread>();
    for (int i = 0; i < noThreads; ++i) {
      Thread t = new Thread() {
        public void run() {
          log.info("Process document: " + iter.next().
             toString());
          while (iter.hasNext()) {
            Document document = iter.next();
            try {
              space.processDocument(document.reader());
            } catch (Throwable t) {
              t.printStackTrace();
            }
          }
        }
      };
      threads.add(t);
    }

    for (Thread t : threads)
      t.start();

    for (Thread t : threads)
      t.join();
  }

  /** .... */
}
```

Listing B.3: Querying the semantic space constructed by LSA

```
public class Query {
  private SemanticSpace sspace;
  private DoubleVector queryVector;
  private DocumentVectorBuilder docBuilder;
  private WordComparator wordCompare;
  private Matrix docMatrix;

  public Query() {
    sspace = LSAUtils.getTermsSpace();
    docBuilder = new DocumentVectorBuilder(sspace);
    wordCompare = new WordComparator();
    queryVector = new DenseVector(sspace.getVectorLength()
        );
    docMatrix = LSAUtils.getDocsMatrix();
  }

  /**
   * @param query
   * @return
   */
  protected DoubleVector getQueryAsVector(String query) {
    queryVector = docBuilder.buildVector(new
        BufferedReader(new StringReader(query)),
        queryVector);
    return queryVector;
  }

  public List<SearchResult> searchDocSpace(String query) {
    DoubleVector queryVector = getQueryAsVector(query);
    final Map<Integer, Double> similarityMap =
            new HashMap<Integer, Double>();
    for (int i = 0; i < docMatrix.rows(); i++) {
      double sim = SimilarityUtil.getCosineSimilarity(
          docMatrix.getRowVector(i), queryVector);
      if (sim > 0.0D) {
        similarityMap.put(i, sim);
      }
    }
    return sortByScore(similarityMap);
  }

  public List<SearchResult> searchTermSpace(String query,
      int maxResult) {
    DoubleVector queryVector = getQueryAsVector(query);

    MultiMap<Double, String> similarityMap = wordCompare.
        getMostSimilarToVector(queryVector, sspace,
            maxResult, Similarity.SimType.COSINE);

    List<SearchResult> results = new ArrayList<SearchResult>();
    for (Map.Entry entry: similarityMap.entrySet()){
      double score = (Double)entry.getKey();
      if (score < 0.001D){
        continue;
      }
      results.add(new SearchResult(0, (String) entry.getValue(), score));
    }

    return results;
  }

  protected MultiMap getSimilarWords(SemanticSpace sspace, String word, int maxResult) {
    MultiMap results = wordCompare.getMostSimilar(word, sspace, maxResult, Similarity.SimType.COSINE);
    return results;
  }

  private List<SearchResult> sortByScore(
          final Map<Integer, Double> similarityMap) {
    List<SearchResult> results = new ArrayList<SearchResult>();
    List<Integer> docIndexes = new ArrayList<Integer>();
    docIndexes.addAll(similarityMap.keySet());
    Collections.sort(docIndexes, new Comparator<Integer>() {
      public int compare(Integer s1, Integer s2) {
        return similarityMap.get(s2).compareTo(similarityMap.get(s1));
      }
    });
    for (Integer index : docIndexes) {
      double score = similarityMap.get(index);
      if (score < 0.001D) {
        continue;
      }
      results.add(new SearchResult(index, "", score));
    }
    return results;
  }

  public class SearchResult {
    public int index;
    public String word;
    public double score;

    public SearchResult(Integer index, String word, double score) {
      this.index = index;
      this.word = word;
      this.score = score;
    }
  }
}
```

Listing B.4: Tag cloud generation

```java
public class TagCloud {
  private Cloud cloud;
  private DictionaryFilter blacklist;
  private Properties props;
  private static Logger log = Logger.getLogger(TagCloud.
      class);

  protected TagCloud(double weight, int maxTags) {
    props = new PropertiesLoader().loadProperties();
    String DEFAULT_LINK = props.getProperty("DEFAULT_LINK
        ");
    String STOPWORDS = props.getProperty("STOPWORDS");
    cloud = new Cloud();
    cloud.setMaxWeight(weight);
    cloud.setMaxTagsToDisplay(maxTags);
    cloud.addInputFilter(getStopWords(STOPWORDS));
    cloud.addOutputFilter(getStopWords(STOPWORDS));
    cloud.setDefaultLink(DEFAULT_LINK);
  }

  protected Filter<Tag> getStopWords(String STOPWORDS) {
    Filter<Tag> stopwords = null;
    try {
      FileReader reader = new FileReader(new File(
          STOPWORDS));
      stopwords = new DictionaryFilter(reader);
    } catch (IOException e) {
      e.printStackTrace();
    }
    return stopwords;
  }

  public Cloud generateCloud(double maxWeight, int maxTags
      , Map<String, Double> tags, double threshold) {
    TagCloud tc = new TagCloud(maxWeight, maxTags);
    tc.populateCloud(tags);
    tc.orderCloud(threshold);
    return tc.cloud;
  }

  @SuppressWarnings("unchecked")
  private void populateCloud(Map<String, Double> tags) {
    double weight;
    for (Object o : tags.entrySet()) {
      Map.Entry<String, Double> entry = (Map.Entry<String,
          Double>) o;
      String word = entry.getKey();
      weight = entry.getValue();
      Tag tag = new Tag(word, weight);
      cloud.addTag(tag);
    }
  }

  protected void orderCloud(double threshold) {
    cloud.tags(new Tag.ScoreComparatorDesc());
    cloud.setThreshold(threshold);
  }

  /** ... */
}
```

Listing B.5: WCC cluster labeling algorithm

```
/**
 * The algorithm nominates terms as cluster labels based on term occurrences.
 *
 * Algorithm is based on the paper "Topic Identification:
 * Framework and Application" by Stein and Meyer zu Eissen
 * http://i-know.tugraz.at/wp-content/uploads/2008/11/40_topic-identification.pdf
 */
public class TagsIdentification {
  List<Tuple> topOccurrences;

  /**
   * @param tags - the set of terms in a document set
   * @param l - how many terms make up the label of a document
   * @param k - how often the same word may occur in the label of different documents
   */
  public void identifyTags(Set<Document> docs, Set<Tag> tags, int l, int k){

    topOccurrences = new ArrayList<Tuple>();

    for(Tag tag : tags){

     /** for each term, get its k most frequent occurrences in documents
      * and save them in list topOccurrences
      */
      List<Tuple> tuples = kMostFrequent(docs, tag, k);

      for (Tuple tuple:tuples){
          topOccurrences.add(tuple);
      }
    }
    sortOccurences();

    for(int labelcount = 1; labelcount < l; labelcount ++){
      int assigned = 0;
      int j = 1;
      Iterator iter = topOccurrences.iterator();
      while((assigned < docs.size()) && (j <= topOccurrences.size()) && iter.hasNext()){
        Tuple tj = (Tuple) iter.next();
        Document doc = tj.getDocument();

        if (doc.getLabelSize() < labelcount){
          doc.addLabel(tj.getTag());
          assigned++;
        }
        j++;
      }
    }
  }

  private List<Tuple> kMostFrequent(Set<Document> docs,
      Tag tag, int k){

    Map<Double, Document> topDocs = getTopK(tag, docs, k);
    List<Tuple> tuples = new ArrayList<Tuple>();

    for (Map.Entry<Double, Document> entry : topDocs.
        entrySet()){
      Double tf = entry.getKey();
      Document doc = entry.getValue();
      Tuple tuple = new Tuple(doc, tag, tf);
      tuples.add(tuple);
    }
    return tuples;
  }

  /**
   * Returns the k documents in which a term occurs most
      frequently.
   *
   * @param tag
   * @param docs
   * @param k
   * @return
   */
  private Map<Double, Document> getTopK(Tag tag, Set<
      Document> docs, int k) {
    Map<Double, Document> documents = new HashMap<Double,
        Document>();
    for(Document doc : docs){
      int idx = doc.getId();
      Vector frequencies = tag.getFrequency();
      Double frequency = (Double) frequencies.getValue(idx
          );
      documents.put(frequency, doc);
    }

    // sort the documents in descending order based on
        term occurrences
    documents = sortMap(documents);

    Map<Double,Document> result = new HashMap<Double,
        Document>();
    int i=0;
    for (Map.Entry entry : documents.entrySet()){
      i++;
      if (i==k){
        return result;
      }
      result.put((Double)entry.getKey(), (Document)entry.
         getValue());
    }
    return result;
  }

  /**
   * Sort a Map in reverse order by its keys.
   *
   * @param docs
   * @return
   */
  public Map<Double, Document> sortMap(Map<Double,
     Document> docs){
    List documents = new LinkedList(docs.entrySet());
    Collections.sort(documents, new Comparator(){
      public int compare(Object o1, Object o2){
        return ((Double)((Map.Entry) o2).getKey()).
           compareTo ((Double)((Map.Entry)o1).getKey());
      }
    });

    Map result = new LinkedHashMap();
    for (Iterator it = documents.iterator(); it.hasNext();
        ){
      Map.Entry entry = (Map.Entry) it.next();
      result.put(entry.getKey(), entry.getValue());
    }
    return result;
  }

  private void sortOccurences(){
    sort(topOccurrences);
  }

  public void sort(List<Tuple> tuples){
    Collections.sort(tuples, Collections.reverseOrder());
  }
}
```

Listing B.6: Lightweight domain ontology for CoreMedia CMS domain

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY protege "http://protege.stanford.edu/plugins/owl/protege#" >
    <!ENTITY xsp "http://www.owl-ontologies.com/2005/08/07/xsp.owl#" >
]>
<rdf:RDF xmlns="http://www.coremedia.com/CoreMediaCMS.owl#"
     xml:base="http://www.coremedia.com/CoreMediaCMS.owl"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://www.coremedia.com/CoreMediaCMS.owl">
        <rdfs:label>CoreMedia CMS Ontology</rdfs:label>
        <rdfs:comment>Ontology for CoreMedia CMS domain.
Developed by Angelina Velinska for Master Thesis project at TUHH, Hamburg</rdfs:comment>
        <dc:creator>Angelina Velinska</dc:creator>
    </owl:Ontology>

    <!-- ///////////////////////////////////////
    // Annotation properties
    ///////////////////////////////////////-->
    <owl:AnnotationProperty rdf:about="&dc;creator"/>
    <owl:AnnotationProperty rdf:about="&rdfs;label"/>
    <owl:AnnotationProperty rdf:about="&rdfs;comment"/>

```

```
<!-- //////////////////////////////////////
// Datatypes
//////////////////////////////////////////-->

<!-- http://www.w3.org/2001/XMLSchema#ID -->
<rdfs:Datatype rdf:about="&xsd;ID"/>

<!-- ///////////////////////////////////////////
// Object Properties
///////////////////////////////////     -->

<!-- http://www.coremedia.com/CoreMediaCMS.owl#
   configures -->
<owl:ObjectProperty rdf:about="http://www.coremedia.
   com/CoreMediaCMS.owl#configures">
    <rdfs:range rdf:resource="http://www.coremedia.com
       /CoreMediaCMS.owl#Component"/>
    <rdfs:domain rdf:resource="http://www.coremedia.
       com/CoreMediaCMS.owl#ConfigurationFile"/>
    <owl:inverseOf rdf:resource="http://www.coremedia.
       com/CoreMediaCMS.owl#isConfiguredBy"/>
    <rdfs:subPropertyOf rdf:resource="&owl;
       topObjectProperty"/>
</owl:ObjectProperty>

<!-- http://www.coremedia.com/CoreMediaCMS.owl#
   contains -->
<owl:ObjectProperty rdf:about="http://www.coremedia.
   com/CoreMediaCMS.owl#contains">
    <rdfs:range rdf:resource="http://www.coremedia.com
       /CoreMediaCMS.owl#Component"/>
    <rdfs:domain rdf:resource="http://www.coremedia.
       com/CoreMediaCMS.owl#Environment"/>
    <rdfs:subPropertyOf rdf:resource="&owl;
       topObjectProperty"/>
</owl:ObjectProperty>

<!-- http://www.coremedia.com/CoreMediaCMS.owl#deploys
    -->
<owl:ObjectProperty rdf:about="http://www.coremedia.
   com/CoreMediaCMS.owl#deploys">
    <rdfs:range rdf:resource="http://www.coremedia.com
       /CoreMediaCMS.owl#Client"/>
    <rdfs:domain rdf:resource="http://www.coremedia.
       com/CoreMediaCMS.owl#Server"/>
    <owl:inverseOf rdf:resource="http://www.coremedia.
       com/CoreMediaCMS.owl#isDeployedOn"/>
    <rdfs:subPropertyOf rdf:resource="&owl;
       topObjectProperty"/>
</owl:ObjectProperty>

```

```
    <!-- http://www.coremedia.com/CoreMediaCMS.owl#describes -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#describes">
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Component"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#hasPort -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#hasPort">
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Port"/>
        <rdfs:domain rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Server"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#isConfiguredBy -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#isConfiguredBy">
        <rdfs:domain rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Component"/>
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#ConfigurationFile"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#isConnectedTo -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#isConnectedTo">
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#isDeployedOn -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#isDeployedOn">
        <rdfs:domain rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Client"/>
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Server"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#isPartOf -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#isPartOf">
        <rdfs:domain rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Component"/>
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Environment"/>
        <owl:inverseOf rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#contains"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#isReplicatedOn -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#isReplicatedOn">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#MasterLiveServer"/>
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#ReplicationServer"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#isStoredIn -->
    <owl:ObjectProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#isStoredIn">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdf:type rdf:resource="&owl;IrreflexiveProperty"/>
        <rdfs:range rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Repository"/>
        <rdfs:domain rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Resource"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>

    <!-- http://www.w3.org/2002/07/owl#topObjectProperty -->
    <owl:ObjectProperty rdf:about="&owl;topObjectProperty"/>

    <!-- ///////////////////////////////////////////
    // Data properties
    /////////////////////////////////////////     -->

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#hasId -->
    <owl:DatatypeProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#hasId">
        <rdfs:range rdf:resource="&xsd;ID"/>
    </owl:DatatypeProperty>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#hasVersion -->
    <owl:DatatypeProperty rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#hasVersion">
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>

    <!-- ///////////////////////////////////////////////
    // Classes
    ////////////////////////////////////////////     -->

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#API -->
    <owl:Class rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#API">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Action -->
    <owl:Class rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#Action">
        <rdfs:subClassOf rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Task"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#AdministrationGuide -->
    <owl:Class rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#AdministrationGuide">
        <rdfs:subClassOf rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Documentation"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Application -->
    <owl:Class rdf:about="http://www.coremedia.com/CoreMediaCMS.owl#Application">
        <rdfs:subClassOf rdf:resource="http://www.coremedia.com/CoreMediaCMS.owl#Client"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Client -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Client">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Component"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Communication -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Communication"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Component -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Component"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       ConfigurationFile -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#ConfigurationFile">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#File"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       ContentManagementServer -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#ContentManagementServer">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#ContentServer"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       ContentServer -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#ContentServer">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Server"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Dataaggregator -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Dataaggregator">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#WorkflowServer
           "/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Database -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Database">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Component"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Delivery -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Delivery">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Environment"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       DeveloperGuide -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#DeveloperGuide">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Documentation"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Document -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Document">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Resource"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Documentation -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Documentation"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Environment -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Environment"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#File
       -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#File"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Folder
       -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Folder">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Resource"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       LicenseFile -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#LicenseFile">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#File"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Management -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Management">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Environment"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       MasterLiveServer -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#MasterLiveServer">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#ContentServer"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Port
       -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Port">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Communication"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       PropertyFile -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#PropertyFile">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#
           ConfigurationFile"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Protocol -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Protocol">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Communication"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       ReplicationServer -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#ReplicationServer">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#ContentServer"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Repository -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Repository">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Component"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Resource -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Resource"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Right
       -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Right">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Task"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Server
       -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Server">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Component"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       SpringConfigFile -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#SpringConfigFile">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#
           ConfigurationFile"/>
    </owl:Class>


    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Task
       -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Task">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       UserGuide -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#UserGuide">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Documentation"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Variable -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Variable">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Task"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Version
        -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Version">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Document"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       WebApplication -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#WebApplication">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Client"/>
    </owl:Class>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       Workflow -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#Workflow"/>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       WorkflowServer -->
    <owl:Class rdf:about="http://www.coremedia.com/
       CoreMediaCMS.owl#WorkflowServer">
        <rdfs:subClassOf rdf:resource="http://www.
           coremedia.com/CoreMediaCMS.owl#Server"/>
    </owl:Class>

    <!-- http://www.w3.org/2002/07/owl#Thing -->
    <owl:Class rdf:about="&owl;Thing"/>

    <!-- ///////////////////////////////////////////////
    // Individuals
    ////////////////////////////////////////     -->

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#HTTP
       -->
    <owl:NamedIndividual rdf:about="http://www.coremedia.
       com/CoreMediaCMS.owl#HTTP">
        <rdf:type rdf:resource="http://www.coremedia.com/
           CoreMediaCMS.owl#Protocol"/>
    </owl:NamedIndividual>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#Hessian
        -->
    <owl:NamedIndividual rdf:about="http://www.coremedia.
       com/CoreMediaCMS.owl#Hessian">
        <rdf:type rdf:resource="http://www.coremedia.com/
           CoreMediaCMS.owl#Protocol"/>
    </owl:NamedIndividual>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#IOR -->
    <owl:NamedIndividual rdf:about="http://www.coremedia.
       com/CoreMediaCMS.owl#IOR">
        <rdf:type rdf:resource="http://www.coremedia.com/
           CoreMediaCMS.owl#Protocol"/>
    </owl:NamedIndividual>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#JDBC
       -->
    <owl:NamedIndividual rdf:about="http://www.coremedia.
       com/CoreMediaCMS.owl#JDBC">
        <rdf:type rdf:resource="http://www.coremedia.com/
           CoreMediaCMS.owl#API"/>
    </owl:NamedIndividual>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#JMXMP
       -->
    <owl:NamedIndividual rdf:about="http://www.coremedia.
       com/CoreMediaCMS.owl#JMXMP">
        <rdf:type rdf:resource="http://www.coremedia.com/
           CoreMediaCMS.owl#Protocol"/>
    </owl:NamedIndividual>

    <!-- http://www.coremedia.com/CoreMediaCMS.owl#
       UnifiedAPI -->
    <owl:NamedIndividual rdf:about="http://www.coremedia.
       com/CoreMediaCMS.owl#UnifiedAPI">
        <rdf:type rdf:resource="http://www.coremedia.com/
           CoreMediaCMS.owl#API"/>
    </owl:NamedIndividual>
</rdf:RDF>
<!-- Generated by the OWL API (version 3.0.0.1451) http://
    owlapi.sourceforge.net -->
```

Appendix C

Document set used for evaluation

Document set overview		
Category Id	Category	Document #
1	session, connection	5
2	server	5
3	publication, workflow	5

Table C.1: Overview of the constructed document set

Doc. Id	Cat. Id	Document
1	1	The session that is created while the connection is opened is also known as the connection session.
2	1	The sessions of the connected clients will be closed and no more content changes are possible.
3	1	The previous code fragment shows how a second session is created from an existing connection.
4	1	Multiple sessions show their greatest potential in trusted applications which receive help in restricting user views while maintaining a shared cache.
5	1	Having opened connection, all actions are executed on behalf of the single user whose credentials where provided when logging in.
6	2	The state of the Master Live Server must be younger than the state of the Slave Live Server.
7	2	The rewrite module checks the availability of the requested file and, in the negative case, passes the request on to the Active Delivery Server.
8	2	The directory layout of the Active Delivery Server has changed as well as the format of the configuration file.
9	2	The CoreMedia Content Server is a central component that manages the content repository and the user authorization.
10	2	The CoreMedia Content Management Server is the production system used to create and administrate content.
11	3	If the database does not allow to take online-backups ensure that all publications are finished and that the last publication was successful.
12	3	The third task in the workflow aims to check if the change set is empty. Then, no publication is necessary and the workflow can be finished.
13	3	This element is used to define which information should be shown in the columns of the workflow list at the left side of the workflow window.
14	3	Publication will be executed when finishing the task after all resources in the change set have been approved.
15	3	The CoreMedia Workflow installation comes with four predefined workflows which cover the publication of resources.

Table C.2: Document set used for evaluation of the implementation

Queries used for evaluation	
Query Id	Query #
1	session, connection
2	server
3	publication, workflow

Table C.3: Queries used for evaluation

Bibliography

[1] A. Spink, J. Bateman, and B. J. Jansen, "Users' searching behavior on the excite web search engine," in *WebNet*, 1998.

[2] B. Stein and S. M. Z. Eissen, "Topic identification: Framework and application," in *Proc of International Conference on Knowledge Management (I-KNOW)*, 2004.

[3] B. Stein and S. M. zu Eissen, "Topic-identifikation. formalisierung, analyse und neue verfahren," *KI*, vol. 21, no. 3, pp. 16–22, 2007.

[4] D. Hiemstra, "Information retrieval models," in *Information Retrieval: Searching in the 21st Century* (A. Göker and J. Davies, eds.), UK: Wiley, 2009.

[5] S. Robertson, "The probability ranking principle in IR," vol. 33, pp. 294–304, 1977.

[6] G. Salton, "Automatic text processing: The transformation, analysis, and retrieval of information by computer," AddisonWesley, 1989.

[7] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* Cambridge, UK: Cambridge University Press, 2008.

[8] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, "Using Latent Semantic Analysis to improve access to textual information," in *Sigchi Conference on Human Factors in Computing Systems*, pp. 281–285, ACM, 1988.

[9] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American Society for Information Science*, vol. 41, pp. 391–407, 1990.

[10] S. Dumais, "LSA and Information Retrieval: Getting Back to Basics," pp. 293–321, 2007.

[11] G. H. Golub and C. F. Van Loan, *Matrix computations (3rd ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[12] M. W. Berry, S. Dumais, G. O'Brien, M. W. Berry, S. T. Dumais, and Gavin, "Using linear algebra for intelligent information retrieval," *SIAM Review*, vol. 37, pp. 573–595, 1995.

[13] S. T. Dumais, "Improving the retrieval of information from external sources," *Behavior Research Methods, Instruments, & Computers*, vol. 23, pp. 229–236, 1991.

[14] P. Nakov, A. Popova, and P. Mateev, "Weight functions impact on LSA performance," in *EuroConference RANLP'2001 (Recent Advances in NLP*, pp. 187–193, 2001.

[15] P. Nakov, "Getting better results with Latent Semantic Indexing," in *In Proceedings of the Students Prenetations at ESSLLI-2000*, pp. 156–166, 2000.

[16] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," 1999.

[17] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[18] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram, "A hierarchical monothetic document clustering algorithm for summarization and browsing search results," in *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pp. 658–665, ACM, 2004.

[19] S. Lloyd, "Least squares quantization in pcm," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[20] C. Carpineto, S. Osinski, G. Romano, and D. Weiss, "A survey of web clustering engines," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–38, 2009.

[21] O. Zamir and O. Etzioni, "Grouper: A dynamic clustering interface to web search results," pp. 1361–1374, 1999.

[22] R. Krishnapuram and K. Kummamuru, "Automatic taxonomy generation: issues and possibilities," in *Proceedings of the 10th international fuzzy systems association World Congress conference on Fuzzy sets and systems*, IFSA'03, pp. 52–63, Springer-Verlag, 2003.

[23] D. Weiss, *Descriptive Clustering as a Method for Exploring Text Collections.* PhD thesis, Poznań University of Technology, Poznań, Poland, 2006.

[24] D. Carmel, H. Roitman, and N. Zwerdling, "Enhancing cluster labeling using wikipedia," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pp. 139–146, ACM, 2009.

[25] D. Mugo, "Connecting people using Latent Semantic Analysis for knowledge sharing," Master's thesis, Hamburg University of Technology, Hamburg, Germany, 2010.

[26] J. Davies, A. Duke, and A. Kiryakov, "Semantic search," in *Information Retrieval: Searching in the 21st Century* (A. Göker and J. Davies, eds.), UK: Wiley, 2009.

[27] T. Gruber, "Ontology of folksonomy," 2005.

[28] I. Peters and P. Becker, *Folksonomies : indexing and retrieval in Web 2.0.* Berlin: De Gruyter/Saur, 2009.

[29] G. Smith, *Tagging: People-powered Metadata for the Social Web.* Thousand Oaks, CA, USA: New Riders Publishing, 2008.

[30] P. Heymann, G. Koutrika, and H. Garcia-Molina, "Fighting spam on social web sites: A survey of approaches and future challenges," *IEEE Internet Computing*, vol. 11, pp. 36–45, November 2007.

[31] D. Jurgens and K. Stevens, "The S-Space package: an open source package for word space models," in *ACL '10: Proceedings of the ACL 2010 System Demonstrations*, (Morristown, NJ, USA), pp. 30–35, Association for Computational Linguistics, 2010.

[32] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008.

[33] M. Brand, "Fast low-rank modifications of the thin singular value decomposition," *Linear Algebra and Its Applications*, vol. 415, no. 1, pp. 20–30, 2006.