Automated bookmark management

Vojtěch Knězu

May 2002

Technical report
TR-DKS-02-02

Data and Knowledge Systems group
Department of Mediamatics
Faculty of Information Technology and Systems

Abstract

In this report, we address the issue of automatization of bookmark management by means of information retrieval and machine learning techniques. We have designed a system that categorizes bookmarks in an automated way and thus facilitates adaptively their management. Organization of bookmarks is based on hierarchical categories, which are mapped to the structure of the Open Directory Project. The user's hierarchy is built dynamically according to additions and removals of bookmarks. Furthermore, the system is designed to recommend related Web pages to the recent bookmarks. This service is based on similarities observed during the categorization process. The designed system has been fully implemented and named CaterBook.

Acknowledgments

I would like to take this opportunity to express my thanks to the people at the Knowledge Based Systems Group at the Faculty of Information Technology and Systems at the Delft University of Technology. They have helped me in many ways and created a friendly atmosphere for the elaboration of my thesis. My gratitude belongs to the Nuffic organization, which financially supported my study stay in the Netherlands. Acknowledgments also go to the members of my home Department of Mathematics at the Faculty of Nuclear Sciences and Physical Engineering at Czech Technical University in Prague for their advices and support.

Special thanks belong to dr. drs. Leon Rothkrantz, who supervised my work and inspired me a lot, not only as a teacher but also as a great person with a big heart.

Last but not least, I would like to thank my parents and the whole family for their continuous support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem definition

Users of the World Wide Web often want to keep track of the valuable Web sites they have visited and where they would like to return. This is usually done by a personal list of URLs, which is commonly called as bookmarks. Well organized bookmarks facilitate a user's work and constitute his or her personal information space. Unfortunately, the current Web browsers offer very limited functionality for the bookmark management, mainly only manual sorting bookmarks into manually created folders. There is no surprise that the bookmark management was one of the main problems of work with the Web reported by users in the recent large survey reported by [Abrams98].

The research of [Kaasten01] outlines several user's problems with the bookmark management, mainly the troubles with long alphabetical bookmark lists, which make desired pages difficult to find. Keeping a list effective for searching requires ideas for organizing and a constant maintenance, and that is something what people are often unable or unwilling to do. Considering these problems, some other ways of bookmark organizing and management should be explored.

1.2 Goals

This thesis wants to address the previously mentioned issues and explore the possibilities for an automated organizing and managing of bookmarks, which would lead to better adaptability to the user's needs. The research comprises an examination of new techniques of information retrieval with respect to the Web and then their application in a prototype system as a proof of concept. Such a system should serve as a personal bookmark assistant, which organizes bookmarks in an automated way and thus saves user's time for their processing and maintenance. The way of bookmark's organizing should be adaptive and incremental, enabling smooth additions of new bookmarks to the recent structure.

In order to make the system more personal and assistant, it should be able to recommend to a user other Web hyperlinks, which are similar to the already organized bookmarks and thus facilitates a user's searching for interesting Web pages.

The work should accompany several testings, from tests of the automated system up to a test of the application by users.

1.3 Related Work

In the recent years we have been witnessing an expansion of several systems and approaches for personalization on the World Wide Web. This section briefly summarizes some of the personal assistants for the bookmark management and recommendation of Web pages.

WebTagger [Keller97] is a system which allows users to store, retrieve and share their bookmarks with others. Each authorized member of a group can deposit a bookmark in an associated group memory. Bookmarks have to be manually categorized into one or more topical categories and can be rated by users. In addition, the system enables a searching for recommendation in the categories, which is based on the ratings.

Bookmark organizer [Maarek96] provides automatic categorization mechanism that uses the clustering analysis to organize bookmarks based on their similarity. It combines hierarchical agglomerative clustering (HAC) technique and user interaction to organize collection of Web documents listed in a bookmark file. The HAC technique starts with singleton clusters, where each contains one document, and in each step merges the two most similar clusters, until there is only one top cluster. The clusters are labeled by the most frequent words and a percentage measure of intra-cluster similarity.

SiteSeer [Rucker97] is a recommendation system that uses bookmarks and their organization within folders for predicting and recommending other relevant pages. It is a collaborative system, which measures the degree of overlap of user's folders with other people's folders. The system does not derive any semantic value from the contents of the URLs.

Syskill & Webert [Pazzani96] is both a personalized recommendation and search system. It recommends interesting Web pages using explicit feedback from a user. If a user rates some links on a page, Syskill & Webert can recommend other links on the page in which he might be interested. In addition, the system can construct a query to the Lycos search engine and retrieve other interesting pages for a user.

WebMate [Chen98] is another personal assistant for Web browsing. It is based on a proxy that stores visited Web pages as weighted keyword vectors and clusters them. These clusters are automatically labeled with the most frequent words and are assumed to represent one domain of a user's interest. Then WebMate spiders Web pages with news articles the user wants to be monitored and creates their personalized presentation based on the user's profile.

Part I

Theoretical background

Chapter 2

Organizing electronic documents

With the increasing number of documents available in electronic document collections, methods for organizing these collections to allow topic-oriented browsing and orientation gain importance. However, manual organizing becomes more source demanding or even infeasible with the large amount. Several research projects have taken up the task of automated organizing document collections, which mainly followed two main approaches: clustering (non-supervised learning) and classification (supervised learning).

This short chapter describes these two main methods in order to sketch the possible ways for the bookmark organizing. Advantages and disadvantages of these techniques for our specific problem and a justification of our selected approach for the design and implementation of our system is described later in section 7.1.

2.1 Classification

In data mining area, we understand classification as a process of finding a model that describes and distinguishes data classes, for the purpose of being able to use the model for predicting classes of objects whose class label is unknown. The derived model is based on the analysis of a set of training data, i.e. objects whose class label is known and applied to a set of testing data, i.e. object which class we want to predict.

This definition is going be more specific in case of classification of electronic documents, where each document is assigned a class label from a set of predefined topic categories, based on a set of examples of preclassified documents. For example, Yahoo!'s taxonomy and its associated Web documents can be used as training and test sets in order to derive a Web document classification model. This model may then be used to classify new Web documents by assigning categories from the same taxonomy [Mladenic98].

Automatic classification is traditionally *content-based*, i.e. performed by extracting the information for representing a document from the document itself. The research area which studies content-based techniques for organizing documents considering mainly text is called automated text categorization and in details is discussed in chapter 3. However, with an explosive growth of documents available on the Web, also other media than text are considered for the classification, e.g. images, plug-ins etc.

Additionally, some complementary approaches to the content-based gain importance, for example *context-based* classification of Web documents, which extracts useful information for classifying a document from the context where a URL referring to it appears. This approach is discussed and evaluated e.g. in [Chakrabarti98], [Attardi99] and [Glover02].

2.2 Clustering

Unlike classification, which analyzes class-labeled data objects, clustering analyzes data objects without consulting a known class label. The class labels are not present in the training data and clustering can be used to generate such labels. The objects are clustered based on the principle of maximizing the intra-similarity within a cluster and minimizing the inter-similarity between clusters. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very unsimilar to objects in other clusters.

Application of clustering techniques for organizing documents has been primarily used in information retrieval for improving the effectiveness and efficiency of the retrieval process [Rasmussen92]. More recently, clustering techniques are used by many intelligent software agents in order to retrieve, filter and cluster documents available on the Web. The similarity between documents is usually measured by using the vector space model, e.g. by the cosine of angle between two document vectors (for details see section 3.3). However, other methods than document vector similarity have been used for clustering, for example neural networks as described in [Bartfai94].

Hierarchical clustering algorithms have been used in document clustering, but the long computation time of this method was always the problem when using document clustering on-line. More recently faster algorithms for clustering have been introduced, namely the method called Suffix Tree Clustering (STC) reported by [Zamir98], which creates clusters based on phrases shared between documents. Automatic labeling of document clusters by labels approproate for humans is still a difficult problem usually solved by using the most frequent or most predictive words, which is rarely satisfactory.

Chapter 3

Automated text categorization

A research area which studies techniques for content-based organizing of documents is called *automated text categorization* (ATC). It is a discipline concerned with the construction of automatic text classifiers, i.e. programs capable of assigning to a document one or more among a set of predefined categories. Building these classifiers is done automatically, by means of a general inductive process that learns the characteristics of the categories from a set of preclassified documents. ATC lies in a crossroad of Information Retrieval and Machine Learning and has a number of applications including word sense disambiguation [Roth98], Web page classification under hierarchical Web directories [Mladenic98], ontology learning [Labrou99] or detection of text genre [Kessler97].

3.1 History

Automated text categorization arose in the early 60s and until the late 80s the main approach used to the realization of automatic document classifiers consisted in their manual construction through manually building an expert system capable of taking categorization decisions. The main shortage of this approach to the construction of automatic classifiers was the existence of a knowledge acquisition bottleneck, i.e. rules for the expert system has to be defined manually by a knowledge engineer by help of a domain expert.

A new approach to the construction of automatic document classifiers based on machine learning arose in 90s and now become the dominant one. In this approach a learning process automatically builds a classifier for a category by observing the characteristics of a set of documents that have previously been classified manually under a category by a domain expert. The machine learning approach for text categorization heavily relies on the information retrieval basis. The reason is that both information retrieval and text categorization are content-based document management tasks, and therefore share many characteristics.

3.2 Task of ATC

Automated text categorization may be defined as the task of assigning a category or more categories to a given text document. It is achieved by automatic text classifiers,

that label documents with categories from a predefined set. Usually, for each category an independent classifier has been constructed, which can decide whether a given document should be classified under the category or not.

More formally, consider a set of documents to be classified $D = \{d_1, \ldots, d_m\}$, and a set of categories $C = \{c_1, \ldots, c_n\}$. Text categorization may be defined as a task of approximation of the unknown function $f : D \times C \rightarrow \{0, 1\}$ by means of a function $f' : D \times C \rightarrow \{0, 1\}$. A value of 1 indicates the decision that a given document may be classified under a category, a value 0 indicates the opposite decision. In case of construction of an independent classifier for each category, we can divide the task of ATC into n independent problems: we are looking for approximation of the total unknown functions $f_i : D \rightarrow \{0, 1\}$ by means of functions $f_i' : D \rightarrow \{0, 1\}$ for all $i = 1, \ldots, n$ [Sebastiani99].

3.3 Vector space model

The most common model used in information retrieval and text-based machine learning applications for document representation is the *vector space model*. The procedure of this model can be divided into three stages: document indexing, weighting of the indexed terms and ranking documents according the similarity measures. These stages are used also in the design and implementation of our prototype system, in more details in chapter 7.

3.3.1 Document indexing

The task of document indexing refers to an automatic processing of text documents into an internal representation such that it can be interpreted by a classifier. It represents documents in a space in which each axis corresponds to a term. Indexing of a document consists in the extraction of content bearing terms. These terms are specified by a dictionary, which can be manually or automatically defined. The dimensionality of the space is given by the number of words in the dictionary. The feature terms are assigned fixed positions in a feature vector and assigning values to these terms projects the document into a numerical space. Generally, document indexing transforms a document into a vector $\vec{d} = (w_{d,1}, \ldots, w_{d,n})$, where $w_{d,i} \in [0, 1]$ for $i = 1, \ldots, n$ are the values (weights) of contained terms in the space.

3.3.2 Term weightening

The terms of the feature vector should to be weightened according importance for the document and thus facilitate the categorization task. Different weightening methods are used, the simplest binary method assigns 1 noting the presence and 0 noting the absence of the feature term. Another method called *term frequency* weightening assigns the count of occurences of terms within the document.

The well-known and most common method for the term weightening is *Term Frequency - Inverse Document Frequency* (TFIDF) scheme. It makes explicit the notion that the feature terms that describe a document can be found among the words that occur frequently within the document, but infrequently in other documents. The TFIDF weighting not only gives weight to predictive words, but also minimizes terms that occur in many documents. The TFIDF weight of the term t in the document d is usually defined as

$$tfidf(t,d) = tf(t,d) \cdot log(\frac{N}{df(t)})$$

where $tf(t,d)$ denotes the number of times t occurs in d, N denotes total number of documents and $df(t)$ denotes the number of documents in which the term t occurs at least once [Salton88].

3.3.3 Similarity measures

The vector space model is used mainly as a way to compare documents and queries to each other. The similarity of two documents x and y with a defined common term dictionary of size n and representation by their feature vectors $\vec{x} = (w_{x,1}, \ldots, w_{x,n})$ and $\vec{y} = (w_{y,1}, \ldots, w_{y,n})$ can be measured by several distance measuring methods, the most used function is the cosine of angle between two feature vectors:

$$sim(\vec{x}, \vec{y}) = \cos(\vec{x} \angle \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \quad ||\vec{y}||} = \frac{\sum_{i=1}^{n} w_{x,i} w_{y,i}}{\sqrt{\sum_{i=1}^{n} (w_{x,i})^2} \sqrt{\sum_{i=1}^{n} (w_{y,i})^2}}$$

3.3.4 Data visualization

In order to visualize a similarity between a bigger amounts of documents, a method which projects a high-dimensional vector space onto a lower-dimensional space is a necessity. We suggest to use the Sammon's mapping for this purpose. The algorithm works in that way it finds the locations in the target space so as much as possible of the original structure of the vectors in the high-dimensional space is conserved.

More formally, consider $\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N}$ vectors in n-dimensional space, distances $d(\vec{x_i}, \vec{x_j})$ between each two vectors according to some metric and a distance matrix D with elements $d_{i,j} = d(\vec{x_i}, \vec{x_j})$. Let $\vec{y_i}$ be the image of $\vec{x_i}$ in the 2-dimensional output space. With O we denote a matrix containing the distance between images. The goal is to place the points $\vec{y_i}$ in such a way that the distance matrix O resembles as closely as possible matrix D, i.e. to optimize an error function E by following an iterative steepest descent process.

$$E = \frac{1}{\sum_{i,j=1}^{N} d_{i,j}} \left(\sum_{i,j=1}^{N} \frac{(d_{i,j} - o_{i,j})^2}{d_{i,j}} \right)$$

We use the Sammon's mapping for our testing which is described in chapter 11.

3.4 Classification methods

A growing number of classification methods and machine learning techniques have been applied to the text categorization task in recent years. We should mention probabilistic Naive Bayes classifiers, Rochio's algorithm, neural networks, decision trees and decision rules approach, support vector machines classifiers and instance-based learning methods. We are focused on the last approach, since we decided to use it in our system (arguments for it are discussed in chapter 7).

Instance-based learning methods do not have an off-line training phase, the training examples are simply stored and no generalization beyond these examples is done.

When a new instance has to be classified, the previously stored examples are examined in order to induct a classifier for the new instance. For this reason, these methods are sometimes called "lazy" learning. It includes k-nearest neighbour classification, locally weighted regression methods, case based reasoning etc. (for a comprehensive study see [Mitchell97]).

3.4.1 Training and testing data

The induction process of the classifiers requires existence of a set of documents with known, already assigned categories, often referred as a $Corpus$. It is usually splitted into two sets, a training set $Train \subset Corpus$ and a test set $Test \subset Corpus, Train \cap Test = \emptyset$. The classifier is constructed automatically by training on a set of documents from $Train$. Once a classifier has been built, its effectivness may be tested applying it to the test set $Test$ and checking the degree of correspondence between the decisions of the automatic classifier and those contained in the corpus. This approach is called the *train and test approach* [Sebastiani99].

3.4.2 Centroid-based document classification

This method, also known as Rocchio's centroid algorithm, is based on the vector space model, it means that documents are represented as vectors in a term space. The main idea of the classification consists in using category-specific centroid vectors, where for each category a centroid vector is created as a representative. This is done by counting an average from all vectors of documents which belong to the same category.

More precisely, given a category C and set of document vectors within the category, $S_C = \{\vec{d_1}, \ldots, \vec{d_n}\}$, the centroid vector $\vec{c}$ is defined as

$$\vec{c} = \frac{1}{|S_C|} \sum_{\vec{d} \in S_C} \vec{d}$$

The classification algorithm is rather simple: for each set of documents belonging to the same category, their centroid vectors are computed. Given k categories in the training set, this leads to k centroid vectors $\vec{c_1}, \vec{c_2}, \ldots, \vec{c_k}$, where each $\vec{c_i}$ is the centroid for the ith category. The category of a new document vector $\vec{x}$ is determined as follows:

$$\arg \max_{j=1,\ldots,k} (\cos(\vec{x}, \vec{c_j}))$$

The computational complexity of the training phase of this centroid classifier is linear on the number of documents and the number of terms in the training set. The amount of time required to classify a new document x is at most $O(km)$, where m is the number of terms present in x. However, this method can suffer from the heterogeneity of Web documents (e.g. various length, structuring, mixed media etc.) and simple averaging over a set of document vectors may not lead to good results (see [Chakrabarti98]).

3.4.3 Megadocument classification

This very new approach was suggested, described and tested in [Klas00]. It's a variation of the centroid-based document classification, but in order to cope with the heterogeneity, the megadocument approach is different: for each category all corresponding

document texts from the same category are concatenated to a megadocument, which is then indexed using standard methods (see figure 3.1).

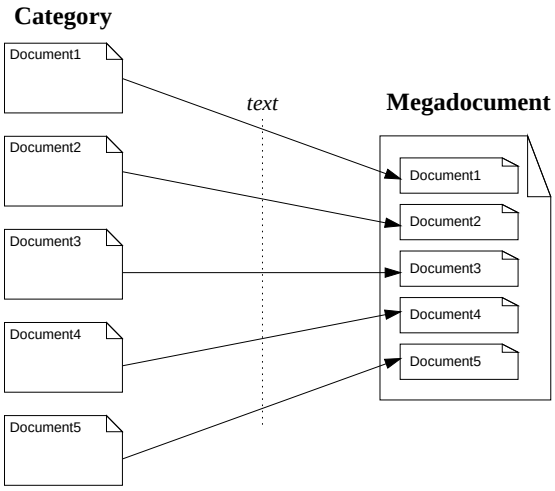

Figure 3.1: Megadocument generation

The classification algorithm is the same as in case of the centroid-based document classification: to classify a new document, the similarity between the document and all the megadocuments is calculated and the most similar megadocument determines the category to be assigned. The computational complexity of both the training phase and the testing phase is the same as for the centroid-based document classification.

The evaluations described in [Klas00] show that for Web collections, the megadocument method outperforms other methods like centroid-based or kNN classification. In contrast, for the Reuters collection, they only achieve average results.

Chapter 4

Searching and organizing on the Web

4.1 Semantics and ontology

The Web has no general classification system for Web sites, no formal indexing policy or controlled vocabulary nowadays. Web pages are represented by HTML code, which does not catch the semantics at all. There is a possibility of using meta tags, e.g. keywords and description tags, but this is not obligatory and a big portion of Web pages do not use it. However, the semantic meaning is crucial for interpreting, exchanging and another automated processing of Web pages. Nowadays we are witnessing the birth of semantic standardization activity [SemanticWeb] under the auspices of W3C organization, with a vision of having data on the Web defined and linked in a way, that it can be used by machines not only for displaying, but for using it in various applications. The eXtended Markup Language (XML) is accepted as the emerging standard for data interchange on the Web.

On the way towards semantics on the Web, an ontology for Web pages is necessary. In philosophy, an ontology is a theory about the nature of existence, of what types of things exist. Artificial intelligence researchers have assimilated the term to their own terminology, and for them an ontology is a specification that formally defines the relations among terms or objects.

The most typical kind of ontology for the Web has a taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them. Public Web directories are the typical examples of such a taxonomy for Web documents. Their categories in a hierarchy can serve as the classes and semantically characterize a content of a Web document and its relation with other documents [Labrou99].

4.2 Web directories

Web directories are organized in a hierarchical manner and compose a tree of categories (or precisely: a directed acyclic graph). Each category is denoted by keywords, describing category content, that appear on the path from the root of the hierarchy to the node representing the category. In other words, a more specific category is named by adding a keyword to the name of the more general category, for example “Science:

Computer Science: Machine Learning". Each category contains together with links to subcategories also links to Web documents corresponding to the category. The taxonomy of major Web directories (e.g. [Yahoo!] or [ODP]) is created and maintained by human indexers, who explore the content of a submitted Web page and decide the most appropriate category. This manual approach ensures a high precision of the categorized Web pages.

A user study [Chen00] showed that users often prefer navigating through such directories of pre-classified content, which provides a categorical view of retrieved documents and enables them to find more relevant information in a shorter time. The hierarchical manner, how are categories in a Web directory sorted, well describes the content of a document and its relation with other documents.

4.2.1 Open directory project

The Open Directory Project [ODP] is the largest, most comprehensive human-edited directory of the Web. It is constructed and maintained by a vast, global community of volunteer editors. Everyone can become an editor for a particular category according his/her knowledge and interest. In November 2001, ODP contained 3 millions of categorized Web sites included in 440,000 categories, which were maintained by 43,000 editors. The Open Directory powers the core directory services for the Web's largest and most popular search engines and portals, including Netscape Search, AOL Search, Google, Lycos, HotBot, DirectHit etc.

4.3 Google

Google is the largest search engine which offers a broad spectrum of services. It indexes about 1.6 billion Web pages (November 2001), mainly HTML files, but also PDF, PostScript, Word and PowerPoint documents has been recently added to the index. Furthermore, it enables searching in Usenet discussion groups and in the Google Web directory. Queries can be submitted in any language and can be limited to 15 languages including all major European languages.

Results of searching are presented with a title, URL, first two lines with search terms highlighted in boldface, and size in bites (see figure 4.1). Additional features of result presentation are a cached link, a similar pages link, a translation link (in case of a foreign language) and a category link (in case the page is categorized in the Google Web directory). The cached link leads to a snapshot of the page as Google crawls the Web. The similar link prompts a service called GoogleScout to search the Web for pages related to the resultant search. According the testing of [Ridley00], GoogleScout returns relevant pages on average at a hit rate of 65% and in general GoogleScout works well for the majority of Web pages.

The Google Web directory is powered by the Open Directory Project [ODP]. The Web search and Web directory are interlaced, so Google's regular Web search results are enhanced by information from the Google directory.

Our system uses services of Google, mainly the Web directory and GoogleScout (see section 7.3.1).

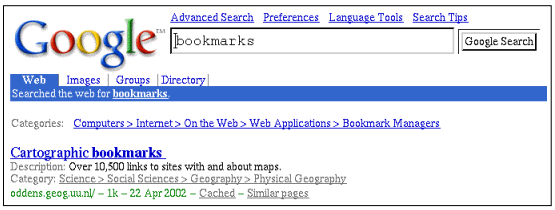

Figure 4.1: Google interface

Part II

Design

Chapter 5

System overview

This chapter gives a glimpse at the design of our bookmark system. It consists of two separate parts: the bookmark management interface and the module for the automated bookmark categorization. Figure 5.3 gives an overview of the system architecture and shows the cohesion of the two parts. With respect to section 4.1, we decided for a bookmark organization similar to the hierarchical manner used in Web directories. User's bookmarks are then sorted in categorical folders and compose a private Web directory. The taxonomy of categories for our system is taken from the Open Directory Project [ODP] (and consequently from the Google Web directory, which actually exploits ODP). Apparently, only a small subset of the ODP categories is engaged in a user's bookmark tree, depending on his interests embedded in the bookmarks (see figure 5.1).

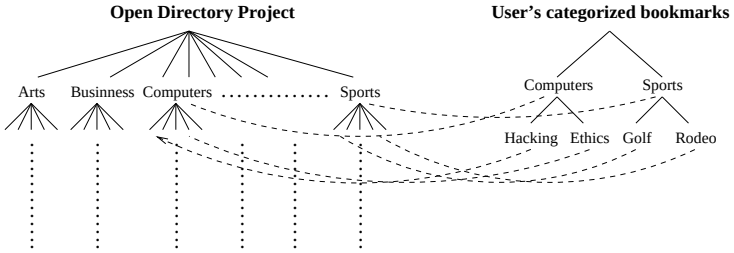

Figure 5.1: Hierarchy of categorical folders for bookmarks and their correspondence to ODP

The decision of the hierarchical organization of bookmarks influenced the design of both the bookmark management interface and the categorization module. The bookmark manager, which facilitates the interaction with a user, presents user's bookmarks in such a tree of categorical folders, in which they are automatically categorized. Figure 5.2 shows a use-case diagram in order to give an overview of possible operations with the manager. The user does not have to care about the categorization process, he only submits a bookmark for categorization and the system automatically suggests a categorical folder for the bookmark. Bookmarks can be also imported from a recent user's bookmark file used within a browser. The categorized bookmarks can be saved and loaded in XML format, which is useful for data exchange with other applications.

Based on the recent bookmarks the system can also recommend hyperlinks to Web pages with a related content. The detailed description of the design of the manager is discussed in chapter 6. The implementation with the goal of a user-friendly interface and an easy installation is described in chapter 9.

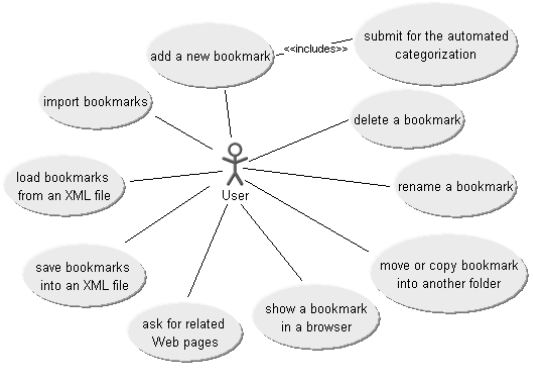

Figure 5.2: Use cases of a user's interaction with the system

The automated categorization module is the more sophisticated part of the system. It tries to find a category for a submitted bookmark. It works independently on the bookmark manager, it means that it looks for a correct category regardless on the category tree which is already existing in the manager and it is up to the manager how to place a categorized bookmark into the recent tree. The categorization process consists of a pre-classification of a bookmarked Web page including a gathering of training data from the Google Web directory, then follows the indexing of the data, selection of features and the final classification. The design of this module is described in chapter 7 and its implementation in chapter 8.

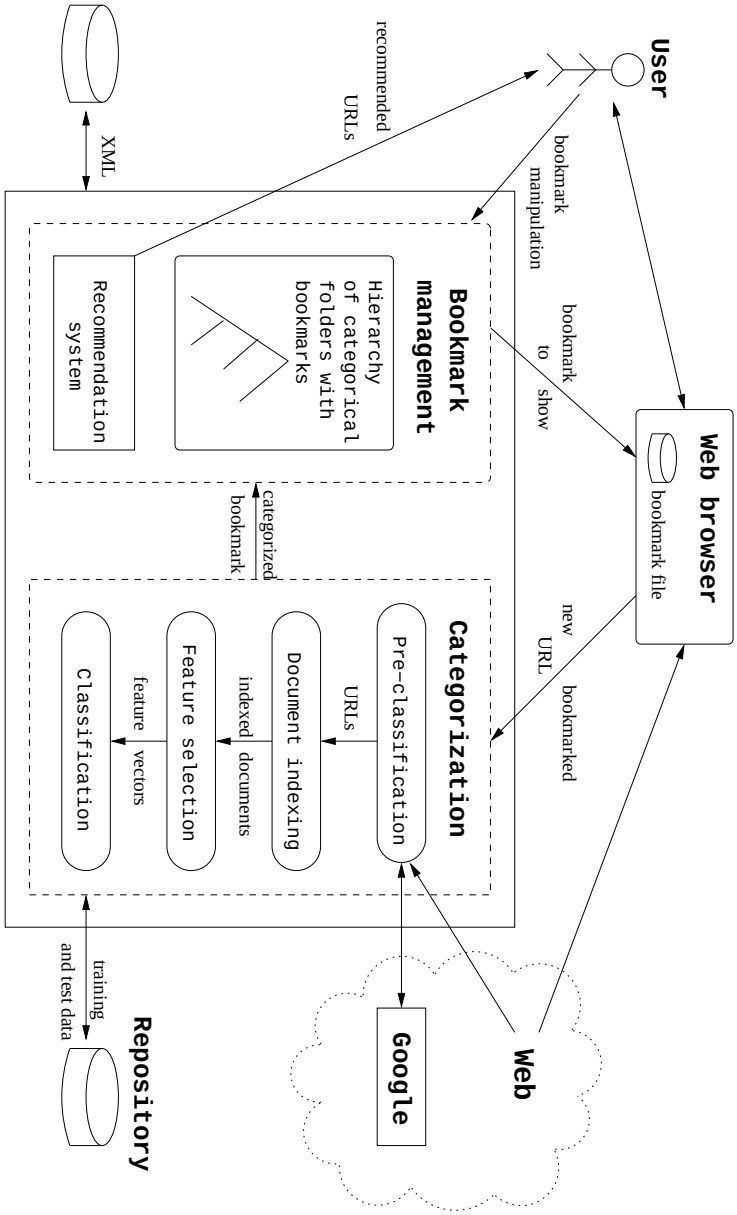

Figure 5.3: System architecture

Chapter 6

Bookmark management

This chapter describes a design of the bookmark manager, the part of the system focused on the manipulation with bookmarks and interaction with a user. Implementation details can be found in chapter 9.

6.1 Tree navigation

Navigation through a hierarchy of categories and searching for a bookmark should be as easy as possible. There are two issues about the navigation: at first, users would like to find a bookmark always at the same place and at second, users do not want to go necessarily too deep in the hierarchy of categories.

The first issue presumes a stability of the tree that can be difficult to achieve since dynamic additions and removals of bookmarks necessarily influence its structure. Our approach for bookmark organizing minimizes changes in the tree structure and ensures that during an addition only a leaf folder can be splitted and the rest of the tree remains the same. A removal of a bookmark can lead to a deletion of the folder when it is empty, but again only in case it is a leaf node. The explication of these operations which influence a structure of the tree comes in the following section 6.2.

The second issue, a deepness of the tree, is influenced only by parameters δ and Δ, which specifies the minimum and maximum number of bookmarks in a folder, respectively. The creation of subfolders can happen only when the threshold Δ is reached in a particular folder. These parameters are set by the user.

For further facilitation of searching for a bookmark, we suggest to display a bookmark not only in a folder where it has been sorted, but also in its parent folders up to the top. To avoid a confusion, such a bookmark is displayed in different way in the parent folders, e.g. by a different font. We assume that it can speed up a searching for a bookmark, because a user do not have to descend to a particular folder, but can find the bookmark also in one of the upper folders. Figure 6.1 depicts such a propagation, the bookmarked URLs displayed with the bold font are sorted precisely in that particular folder, the URLs in italic are sorted in a child folder but propagated to the upper folders.

6.2 Algorithms for the bookmark manipulation

Addition of a new bookmark

The automated categorization process tries to find a category as much as specific for a new bookmark. It means that the suggested category is often deep in the tree of ODP and the corresponding categorical folder does not have to exist in the user's tree. In that case the new bookmark is sorted to the nearest existing parental categorical folder in the user's hierarchy (figures 6.2 and 6.3). For example, consider a bookmark was assigned into a category "Agents" with the path "Computers > Artificial Intelligence > Agents" in the categorization process . However, in the user's tree only the categorical folder "Computers" exists without any children. Then, the new bookmark is sorted directly to "Computers" without creating the subfolders "Artificial intelligence" and "Agents".

Removal of a bookmark

A removal of a bookmark can lead also to a removal of the assigned folder. It happens when the folder is a leaf in the tree and the number of bookmarks is lower than δ. In other cases only the bookmark is deleted and the structure of folders remains (see figure 6.6).

6.3 Integration with a browser

A bookmark management system should fit naturally into a Web browser. We discern the following important issues for the integration:

1. Easy import of recent bookmarks to the system.
2. Automated addition of bookmarks, which were freshly bookmarked by a user within a browser.
3. Launch of a browser with a selected bookmark.

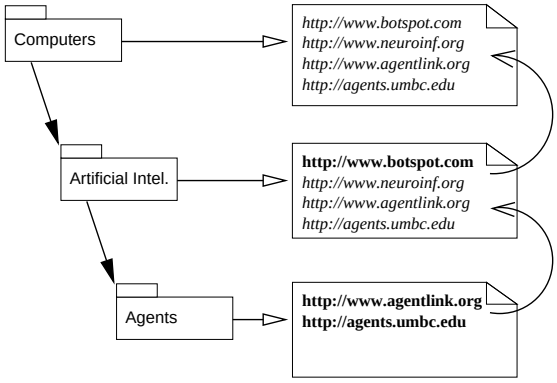

Figure 6.1: Propagation of bookmarks to parent folders.

```
addBookmark( Category C, Bookmark b )
  Folder F= findAppropriateFolder(C)
  F.bookmarks = F.bookmarks ∪ {b}
  if(|F.bookmarks| ≥ Δ) then
    split(F)
  else if( F.hasParent() ) then
    addChildBookmark(F.parent, b)
  end if
```

Figure 6.2: Algorithm for addition of a recently categorized bookmark

```
Folder findAppropriateFolder( Category C )
  Folder F = rootFolder;
  integer level = 1;
  while(C.hasLevel(level)
       AND F ≠ nil
       AND F.hasChildren()) do
    String label = C.getLabel(level);
    F = getChildFolder(label);
    level++;
  done
```

Figure 6.3: Algorithm for finding the nearest existing folder to a category

```
addChildBookmark( Folder F, Bookmark b)
  F.childBookmarks = F.childBookmarks ∪ {b}
  if( F.hasParent() ) then
    addChildBookmark(F.parent, b)
  end if
```

Figure 6.4: Algorithm for addition of a bookmark to parental folders

```
split(Folder F)
  for all Bookmark b ∈ F do
    Folder subF = findOrCreateSubfolder(b)
    subF.bookmarks = subF.bookmarks ∪ {b}
    F.bookmarks = F.bookmarks \ {b}
    F.childBookmarks = F.childBookmarks ∪
{b}
  end for
```

Figure 6.5: Algorithm for splitting of a folder

```
removeBookmark(Folder F, Bookmark b)
  F.bookmarks = F.bookmarks \ {b}
  if(F.hasParent()) then
    F.parent.removeChildBookmark(b)
  end if
  if(|F.bookmarks| < δ
       AND F.hasNoChildren()) then
    deleteFolder(F)
  end if
```

Figure 6.6: Algorithm for removal of a bookmark from a folder

4. Preservation of the recent bookmark file.

Both of the well-known browsers, Internet Explorer and Netscape Navigator, have an integrated bookmark tool, even though of the poor usability. The issue is, how our system can replace these simple bookmark organizers and fulfill the mentioned issues.

We examined the possibility of using the Webclient project ([Webclient01]), which aims to provide a browser-neutral Java API that enables generic Web browsing capability. However, the project is in an early stage and not mature enough for an effective and stable usage.

Another promising alternative was seen in an adaptation of a new version of the Netscape Communicator browser (6.x), which is based on the open source project Mozilla and thus offers more possibilities for a customization and enhancement. This is achieved especially by the XML user interface language ([XUL01]), which is a cross-platform language for describing user interfaces of applications. It is a promising approach, however it presumes an adaptation of the browser code. Unfortunately the current versions of Netscape and Mozilla browsers are also rather unstable and buggy, so this solution could not be recommendable nowadays.

The design and implementation of our bookmark manager as an applet seemed to be a feasible solution. An applet can communicate with a browser by means of Javascript and the bookmarking can be solved by a button or a keystroke within the applet. But there are some issues, which makes this approach worse applicable. Our system has to read and write to local files and that interferes with the applet security. Therefore an applet has to be trusted by a certificate authority. This is not very convenient, especially during the phase of development.

Finally, we decided to design our bookmark manager as a Java stand-alone application. The following paragraphs describe how the chosen approach challenges with the issues, the details about implementation can be found in chapter 9.

The import of bookmarks can be feasibly solved. The system supports import from several types of bookmark files. The first run of the system requires setting of such a bookmark file for the initial import and automated categorization. However, import of another bookmark file can be done at any time.

The automated addition of a new bookmark can be seamless; when a user bookmarks a new URL within a browser, it is automatically submitted for the categorization

and the result inserted into the manager. For this task we designed and implemented a monitor, which checks in intervals for changes in a bookmark file. If a new URL is found, it is automatically send to the categorization module. We argue that a delay between a Web page is bookmarked by the user within a browser and the time when the bookmark is categorized and inserted into our application is not substantial, because users rarely look for a recently added bookmark immediately after the bookmarking.

The launch of a browser with a selected bookmark is easily implemented independently on a type of browser or a platform. A user has to set his preferred browser at the first run of the system. When a user presses a button, an instance of the preferred browser is launched with the bookmark.

The preservation of the recent bookmark file offers to a user an option which bookmarks' structuring is more suitable in a given case. Our system saves the categorized bookmarks in another file and does not make any change at the bookmarks maintained by a browser. Furthermore, the system exports bookmarks in the XML Bookmark Exchange Language [XBEL99], which can be easily compiled to a browser-specific format and used instead of the former one.

6.4 Recommendation service

One of the goals of our work aims at a recommendation service of related Web pages to the recent user's bookmarks. We suggest the service can be based on similarities between a bookmark and the training data used in the categorization process (see section 7.3). The training data consist of Web pages that are categorized in the Google Web directory. The category that is after classification considered as the winner contains the most similar Web pages to the categorized bookmark. Thus when a user looks for similar Web pages to that particular bookmark, the system can recommend some of those Web pages contained in the winning category of the Google Web directory. It recommends only the top hyperlinks according to the ranking of Google, in order to offer to the user really the most relevant URLs. The whole process is displayed at figure 6.7. Implementation details about the recommendation service follow in chapter 9.

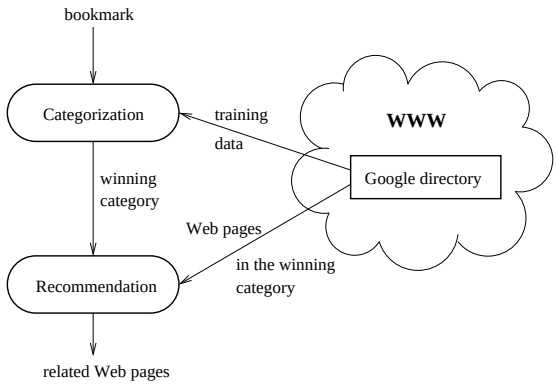

Figure 6.7: Recommendation of related Web pages to a bookmark

Chapter 7

Categorization model

7.1 Consideration of different approaches

Both classification and clustering techniques can be used for organizing documents and thus also for the organizing bookmarks. The theoretical background of those techniques is already described in chapter 2. In this section we discuss the practical issues and compile advantages and disadvantages of these approaches with objective of the best decision for the design of our system.

Clustering

Clustering for bookmark organizing was already used in work of [Maarek96]. We conclude from this and other studies at least three shortcomings of this approach. First, clustering cannot deal with incremental changes effectively, e.g. running a clustering algorithm on ten documents may result two categories. Then, adding ten more documents and rerun the clustering algorithm may yield a complete different set of categories. This dynamic nature of document organization suffers from poor usability, since the user may have harder time to need documents being frequently reclassified than if the documents are not organized in the first place. Second shortcoming lies in a possible low accuracy of document categorization because of the small number of documents (bookmarks) at the client side. Finally, choosing distinctive labels for the clusters generated by non-supervised learning algorithms is difficult. The study of [Maarek96] uses the most frequent words in a category as labels but the presented resulting labels does not seem very representative.

Classification

The first apparent disadvantage of the classification technique consist in the training phase, which has to be carried out before any bookmark is possible to classify. It can be time and space consuming, especially when there is no restriction for a topic domain of bookmarks, which is also our case. On the other hand, training data can be obtained from several Web directories. The labeling of categories used in Web directories also solve the problem which the clustering approach suffers. The automated text categorization technique (ATC), described in chapter 3, is a classification method which can fit to the classification of bookmarks.

Conclusion

The shortcomings of the clustering techniques are significant and this approach for bookmark organizing was already examined by [Maarek96]. The ATC as a classification method has been examined for several types of documents and datasets, but never applied for bookmarks. We concluded that it offers a challenging way for the bookmark organizing, which should be explored.

After the decision for the classification approach, we considered several methods based on the machine learning, which are suitable for the automated text categorization: Naive Bayes classifiers, regression models, neural networks, instance-based classifiers, support vector machines (SVM) etc. We have concluded from a literature survey that especially SVM and instance-based classifiers can outperform other methods [Yang99]. SVM is a promising approach, but as the most of the other techniques, its training phase has to be accomplished before any document can be tested. It means that the complete training dataset has to be available and the training is strictly separated from the testing phase. It's a binding limit for our system, because we cannot gather a dataset, which can train the classifier for any category of human interest in advance and even we cannot afford to save it at the client side. Therefore we suggest to use instance-based classification, where the training data are gathered at the time when new document has to be classified and the dictionary of terms for classification adapts for each testing case.

7.2 Web document indexing

The classification process is based on the similarity (or distance) measurements between documents. However, Web pages in their HTML form cannot be directly used for such a measurement and must be first transformed into another document form, where the content is represented mainly by a feature vector. This process is generally referred as a Web document indexing. For our system we designed a module for the document indexing, which is composed by submodules for the HTML fetching, parsing, tokenisation, stop word deletion and stemming, as depicted at the activity diagram in figure 7.1.

The indexing process is following: first a Web page is fetched from a Web server and parsed. Only the text without HTML tags and comments is extracted, the rest is omitted. When a Web page does not contain enough of text, also the pages which are saved at the same server and linked from the former page, are fetched and parsed. Next, the extracted text is tokenized and common words without any semantic value are deleted. Finally the words are stemmed by means of the Porter stemmer. The number of words which will represent document can be specified. These words are stored in a term frequency vector. Details about each step of the indexing process are described in the following paragraphs.

7.2.1 HTML parsing

How and what should be displayed to a user in a browser is defined by the HTML code of the Web page. Unfortunately, a content of a page is not separated from its visual appearance definition, like in XML, but everything is mixed all together. Thus, the terms which are potentially useful for the classification have to be extracted.

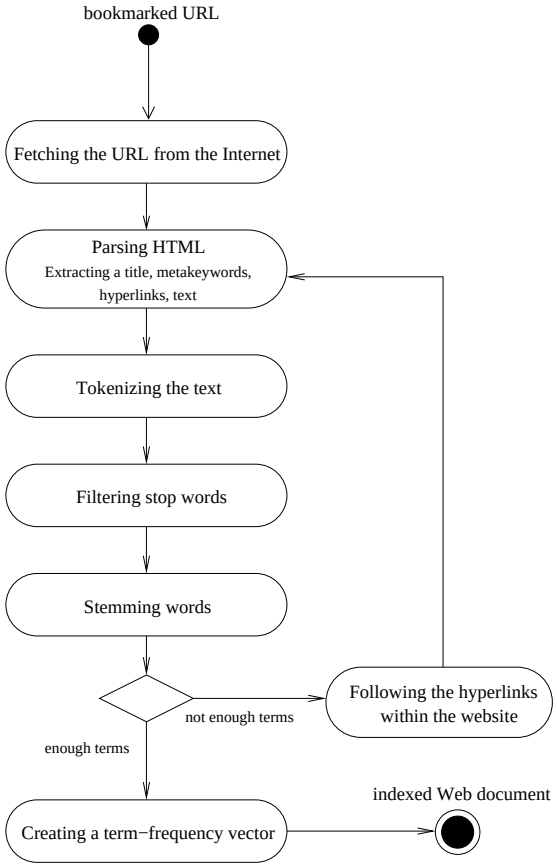

Figure 7.1: Activity diagram of a Web document indexing

We have considered a possibility of usage one of the free available HTML parsers, but finally we have decided for our own implementation. Available parsers are either too complex or cannot handle with all of the HTML features. We prefer full control over the code and a speed provision of our own parser.

Our HTML parser omits useless parts of the code and extracts only the following:

- title
- meta keyword tag
- meta description tag
- normal text
- links to other pages which are stored at the same server

7.2.2 Tokenisation

Tokenisation extracts words from the text. Our system does not use word dictionaries, because it takes more time to access them. In order to exclude as many noise words as possible, only the words which consist of characters and their length is less than 20 are considered. All capitalized letters are translated into lower case.

7.2.3 Stop word deletion

Some words appear in natural text very frequently and have very little information value . Thus they are not beneficial for information retrieval, namely for our classification task and it's useful to delete them already during the indexing phase. These noisy words are often so-called as stop words.

We define two kinds of stop word lists: general English stop words and Web specific stop words. General English stop words are independent of the domain and the set includes English words like "the", "of" etc. Web specific stop words include the words appearing in most Web pages, such as "mail" and "home". Since we did not find any standardized list of English or Web specific stopwords, we defined our lists containing 325 words and 25 words, respectively.

Besides the benefit of decreased sized of corpus, stop word deletion can improve accuracy of classification up to 8% [Mase98]. Note that the stop word deletion is done before the stemming. The stemming process is more computational demanding than stop word deletion.

7.2.4 Stemming

Stemmer is a program that reduces word forms to a canonical form, almost like a lemmatiser. The main difference is that a lemmatiser takes only inflectional endings off a word. A stemmer on the other hand tries to remove derivational suffixes as well, and the resulting string of characters might always be a real word.

Stemmers are used mainly for information retrieval. The performance of the system is improved if terms with the similar meaning are conflated into a single term and so the terms are identical for indexing purposes. This may be done by removal of the various suffixes like -ed, -ing, -ion, -ions etc. to leave the single term. For instance terms "connect", "connected", "connection", "connections" can be stemmed to the single term "connect".

In addition, the suffix stripping process apparently reduces the total number of terms in the system, and hence reduce the size and complexity of the data, which is advantageous. It can reduce the dictionary size by around 20 % [Mase98]. Thus we decided to encompass the well-known Porter's stemming algorithm [Porter88] to our system.

We considered also an additional reduction of the dictionary size by using a thesaurus, but that requires a large database on a client-side or a Web service on a server-side and therefore we did not involve it.

7.3 Classification

7.3.1 Training data and pre-classification

Most of the experimental categorization systems utilize for training some of the existing Web directories like [Yahoo!] or [ODP]. The systems use the same taxonomy of categories and train their classifiers on the included documents. However, the number of categories and documents contained in a Web directory is generally very large (see section 4.2). Because of that, the recent categorization systems are designed only for the top-level categories [Chen00], [Mase98] or another portion of a Web directory, e.g. a "Computer Sciences" subtree ([Govert99], [Mladenic99], [Frommholz01]).

Despite of the large amount of categories, we have to design our bookmark categorization system that it is suitable to categorize Web pages of any area of human interest. Therefore we suggest before the classification is carried out, a domain into which a bookmark can fall should be narrowed. A number of candidate categories for a bookmark has to be reduced and then the amount of training data would not be so extensive. Therefore we designed of an agent, which negotiates with Google about candidate categories for a bookmark and prepares training data for the classification. It can be done by additional services which Google offers, namely by GoogleScout and the Google Web directory (see section 4.3).

The agent for the pre-classification carries out the following steps for each bookmark:

1. It sends a query to GoogleScout for related Web pages to a given bookmark.
2. From results it extracts hyperlinks of associated categories, which lead to the Google Web directory.
3. From the directory pages representing each category it extracts hyperlinks to Web documents

Figure 7.2 shows an example of such a negotiation, which is actually done in automated way by the agent. Thus we obtained candidate categories for a bookmark, each with a set of categorized Web documents which can serve well as the training data for the classifier. We conclude that the utilization of Google has the following advantages:

1. The Google Web directory composes a hierarchy of categories whose labels describe comprehensibly any domain of a human's interest. The categorized Web pages in the directory are of a good quality and can serve well as a training data.
2. The service of GoogleScout enables to find candidate categories for a bookmark. This kind of pre-classification enables to design our classification system without a large training database.

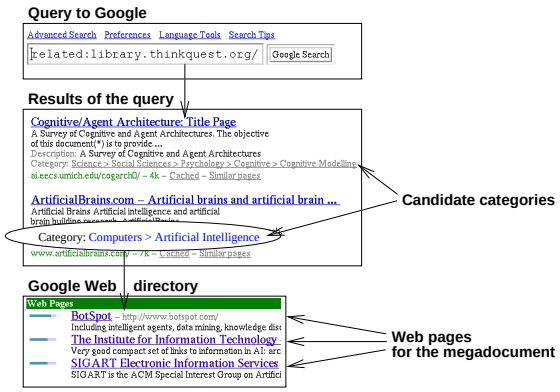

Figure 7.2: Example of pre-classification by means of Google

7.3.2 Classifier

Most of the automated text categorization methods rely on the existence of a good quality text like those in the Reuters-21578 test collection [Reuters]. However, Web documents are rather heterogeneous and their content can be based also on non-text media like pictures, plugins etc. Metadata tags are not used consistently and often does not exist in a Web page. Our testing of similarities between documents described in chapter 11 shows that it happens a document is rather dissimilar to others of the same category, because of these issues. Thus we conclude it is worth to use a classifier based on the megadocument approach (see section 3.4.3), which can handle better with the heterogeneity than a k-nearest neighbour or centroid-based document classifier [Klas00].

Figure 7.3 displays the whole classification process. After the pre-classification and the document indexing, the classifier gets on the input set of megadocuments and the test document. Each megadocument is a representative of a candidate category obtained in the pre-classification process. The term dictionary is defined by terms extracted from the megadocuments. Vectors with frequencies of the terms are created for the documents. Then the vectors are weighted by TFIDF and similarities between the vector of the tested document and the vectors of megadocuments are calculated by the cosine of angle (see section 3.3). The highest similarity is compared to a threshold t and when it is higher, the corresponding category to the winning megadocument vector is assigned to the tested bookmark. If the similarity is lower than t, the bookmark remains uncategorized.

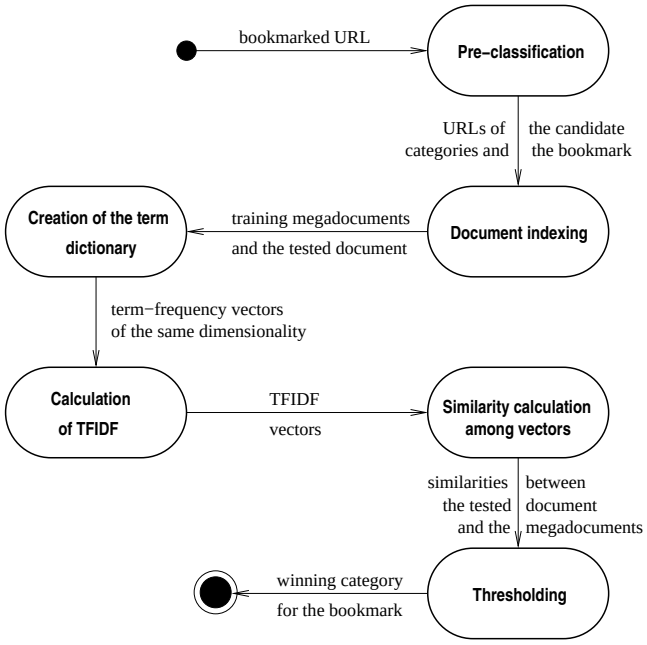

Figure 7.3: Activity diagram of the classification process

Part III

Implementation

Introduction

The system prototype was fully implemented and named CaterBook. The name is somewhat a reflection of its design concept as a bookmark manager with the automated categorization.

CaterBook was implemented in the Java programming language using the Java 2 SDK, v1.3. No other third-party packages or classes were used for CaterBook. The class files, source code and Java programming documentation are included on the enclosed CD-ROM. A brief user's manual to CaterBook can be found at the appendix A.

Details about the implementation of the categorization module and the bookmark manager are described in chapter 8 and 9, respectively. Testing and evaluation of the system by users is described in chapter 13.

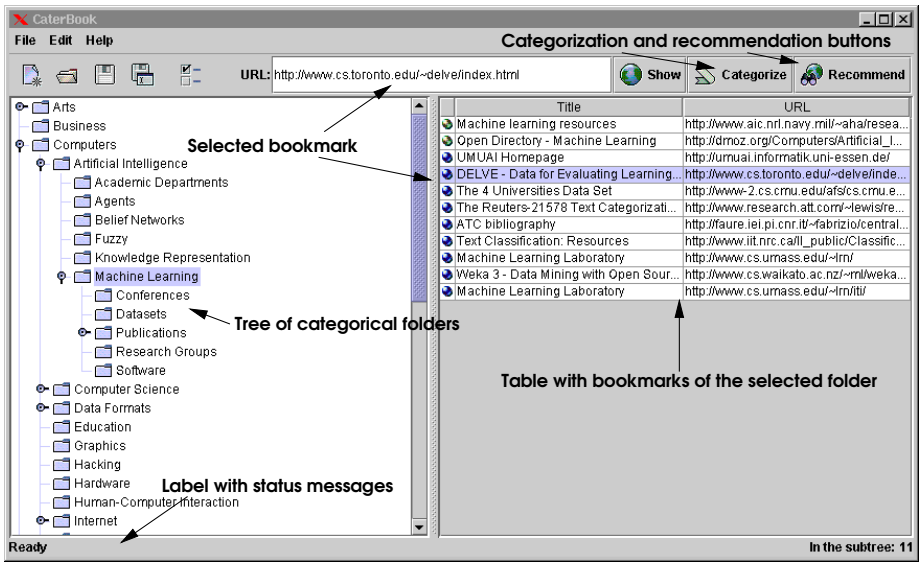

Figure: Screenshot of CaterBook

Chapter 8

Implementation of the categorization module

8.1 Overview

The categorization module is implemented as a separate package called `catr`. The implementation corresponds to the design described in chapter 7. Table 8.1 summarizes Java classes of the `catr` package. The more detailed explanation of important classes follows in the next section. The well commented source code files and the Java documentation might be useful for a deeper understanding. Each class (unless it is not an inner class) is saved in a file of the same name with the extension ".java" .

Table 8.1: Java classes of the categorization module

Class name	Task of the class
`Document`	indexed Web site
`Category`	category of Google with assigned documents
`MegaDocument`	mergence of indexed documents from a Google category
`DocIndexer`	indexer of Web documents
`MegaDocIndexer`	indexer and merger of the ODP Web documents
`GooWebSearch`	agent for interaction with the GoogleScout
`GooDirSearch`	agent for interaction with the Google Web directory
`TFVector`	term-frequency vector of a document
`HTMLParser`	HTML parser of Web pages
`ParserThread`	separate thread covering the HTTP connection
`StopWordChecker`	checker of a text for English and Web stop words
`PorterStemmer`	stemming by the algorithm of Porter
`Tokeniser`	tokeniser of text into words
`Corpus`	a corpus of training documents
`LazyClsfier`	instance-based classifier
`Repository`	repository of the data
`Catizer`	main categorization class

Figure 8.1 displays the event trace of the categorization process. It shows only the case, when neither the tested or the training documents were found in the repository of the system. When it occurs, the indexing is not carried out, but the indexed documents are taken from the repository.

8.2 Classes description

Document

An object of the `Document` class represents an indexed Web page. The content of the page is extracted into a term-frequency vector, i.e. into a `TFVector` object. Each document can be assigned to a category. The appropriate category can be found out in the categorization process or explicitly designated in case the Web site is categorized in the Google Web directory.

Besides the term-frequency vector and the category, a document contains its URL and title of a Web page.

HTMLParser

This class provides the parsing of a Web page and extraction of the contained text and hyperlinks. A Web page is specified by a URL and it is fetched by `GET` method of the HTTP protocol. The plain text, title, meta keywords, meta description and set of outgoing hyperlinks can be requested after a successful parsing.

The parser extracts only the hyperlinks, which lead to Web pages on the same server and in the same subtree. This feature is implemented on behalf of the document indexer, that utilizes it. Only HTTP hyperlinks are extracted, other protocols are omitted. The links extracted from a Web page which are in form of a relative path (e.g. “jaxb/index.html”) or an absolute path (e.g. “/xml/jaxb/index.html”) are transformed to their global representation (e.g.”http://java.sun.com/xml/jaxb/index.html”). The links with another host then the parsed page are omitted. Fragments in a URL, i.e. the parts that start with ’#’ character are deleted for avoidance of a URL ambiguity.

A robustness of the parser was tested by more than 200 Web pages and had been improving, so finally it can recognize and handle with the following issues and errors:

- recognition of the content type, only HTML is considered.
- skipping Web pages with the `404 Not Found` and `403 Forbidden` HTTP status code response.
- automated redirection in case of the `301 Moved` HTTP status code response.
- automated redirection in case of the HTML tag `<META HTTP-EQUIV= "Refresh" CONTENT="x; URL=new.URL">` in the HEAD tag of the Web page.
- managing frames: parsing of HTML code for each frame.
- handling with the usage or non-usage of quotation marks for links.
- skipping commentaries in the HTML code.
- recovery from malformations of the HTML code.

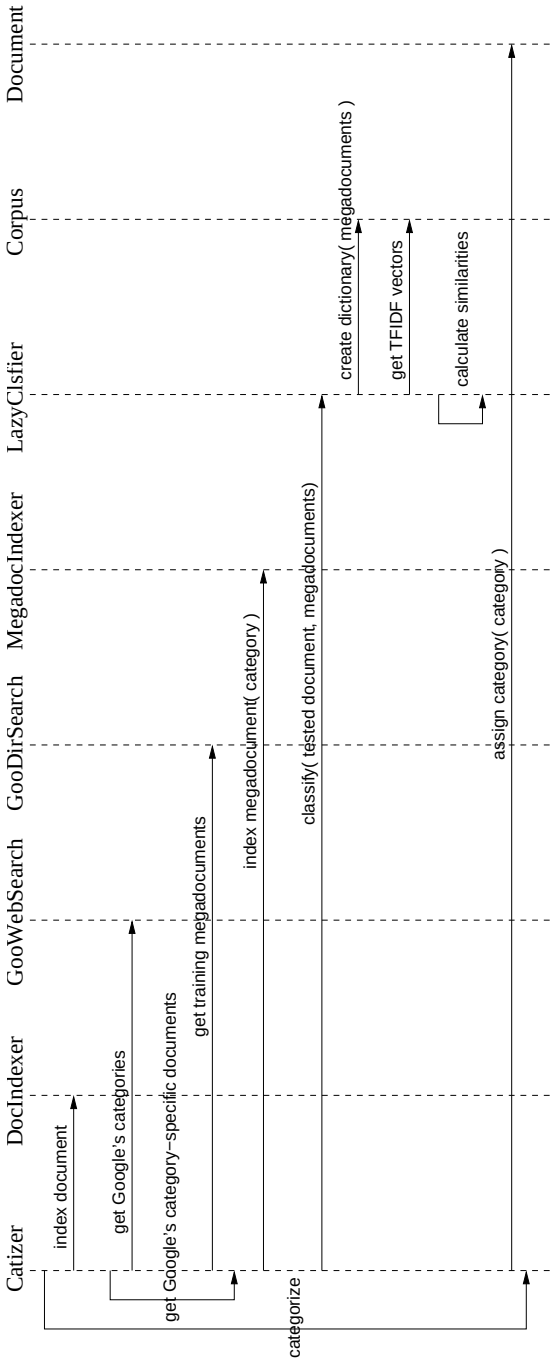

Figure 8.1: Event trace diagram of the categorization process

Furthermore, the time-out problem of unreachable Web sites is solved more efficiently than the standard `HttpURLConnection` class offers. An attempt for connection is invoked by a separate thread of class `ParserThread` and the object of `HTML-Parser` waits only a limited time (e.g. 5 seconds) and after expiration continues with processing other URLs.

PorterStemmer

For stemming purposes, we adopted and implemented the Porter's stemming algorithm [Porter88]. This algorithm has been widely used, quoted, and adapted over the past 20 years. It works only for English words.

StopWordChecker

This class provides checking of words whether they belong among the general English stop words or Web specific stop words. The lists of stop words are saved in the text files "engstopwords.txt" and "webstopwords.txt" in the directory of the application. These lists are downloaded during creation of an object of this class and they are stored in two hash tables (English and Web specific) in order to recognize a stop word in a constant time.

DocIndexer

The `DocIndexer` class provides the indexing of a Web site. It exploits services of `HTMLParser` for downloading and parsing of the main Web page and hyper-linked pages from the same server. Then, the services of `Tokeniser`, `PorterStemmer` and `StopWordChecker` are used for the creation of a document with the term-frequency vector representation. When a Web site is not reachable or it is not in English (this can be checked by means of `StopWordChecker`), the indexing method `indexDoc()` returns `null` value.

Megadocument

The `Megadocument` class is a child of the `Document` class and represents a mergence of Web sites, which are categorized in the Google Web directory and thus are specific for a certain category. Objects of this class are intended as the training data for the classifier.

MegaDocIndexer

The `MegaDocIndexer` is an indexing tool for megadocuments, i.e. objects of the `Megadocument` class. It indexes a set of given Web sites and their content merges together into one term-frequency vector. The class is a child of the `DocIndexer` class and thus inherits and uses the auxiliary methods for the indexing task.

GooWebSearch

The `GooWebSearch` class represents that part of the pre-classification agent which communicates with GoogleScout. It tries to find candidate categories for a bookmark. It makes queries to Google for related pages to the bookmarked URL, i.e. for the Web pages which are similar to the bookmark. The results are filtered and only the

Web pages, that have assigned a Google's category, are considered. It extracts not hyperlinks of the Web pages, but hyperlinks of the assigned categories, which later serve as the candidate categories for the submitted URL. This service can be used by invocation of the public method `getCatURLs()`. It can happen that a new bookmark is already categorized in the Google Web directory. In this case only the corresponding category is returned.

The subcategories of the "World" category are omitted, because they contain Web sites in different languages and our classification system is designed only for English. Also the "Regional" and "By region" subcategories are omitted, because of difficulties with the regional-specific categorization.

GooDirSearch

The `GooDirSearch` class represents the second part of the pre-classification agent, which communicates with the Google Web directory. It extracts URLs of Web sites, that are categorized in a given category. The number of Web sites can be specified. This can be done by invocation of the `getDocURLs()`method.

Corpus

The `Corpus` creates a term dictionary from the contained documents and calculates TFIDF vectors for the training of the classifier. It also calculates norms of the document vectors, however it does not divide each member of vector by the norm, but postpones the division at the classification time, where the whole sum is divided only once, which saves the computational time.

LazyClsfier

The `LazyClsfier` class provides instance-based (also known as lazy) classification. It can be invoked by the public method `classify()`, where the collection of training documents and a tested document should be passed. In case of the successful classification, it returns the most similar training document to the tested one. The classification is considered as unsuccessful, when none of the similarities between the training documents and the tested document is not higher than the threshold `SIM_THRESHOLD`. The value was set experimentally to 0.1, it means that the tested document has to be similar to one of the training documents at least in 10%, otherwise the classification failed. The classifier uses an object of the `Corpus` class for the creation of the term dictionary and the TFIDF computation. The private method `calculateSim()` calculates the similarity among the tested document vector and the training document vectors by the cosine of angle between two TFIDF vectors.

Repository

The class serves as a persistent repository for indexed Web documents, megadocuments and categories. Once a bookmark is being categorized, all the training data are saved there, in order to speed up a future categorization of a bookmark with similar training data. The repository is loaded at the launching of CaterBook and saved at the closure. The repository file is named as "repository" and is saved in the application directory.

Catizer

This class provides the main interface to the categorization module. It is a descendant of the `Thread` class and thus the whole categorization can run independently on the whole system. The categorization of a Web page, which is represented by a URL, can be invoked by one of the public methods with the same name `categorize()`. One of them waits until the whole categorization is finished and returns a `Document` object with the assigned category. The second method adds the categorization request to a queue and lets the categorization process run separately. The sequence of commands in the `categorize()` method is the following: at first it looks into the repository, whether the Web site wasn't already indexed and categorized. Otherwise it uses a `DocIndexer` object for indexing and a `GooWebSearch` object for obtaining candidate categories for the document. If there are no candidate categories, the categorization fails. When candidate categories were found, it looks again in the repository for megadocuments that represents the categories, otherwise a `GooDirSearch` object and a `MegadocIndexer` object are used for obtaining them. The megadocuments and the document being categorized are passed to the `LazyClsfr` object for the classification. If the classification finishes successfully, the most similar megadocument is returned by the classification method. The category corresponding to the megadocument is then assigned to the Web document.

Chapter 9

Implementation of the bookmark manager

9.1 Overview

The bookmark manager is implemented in a separate Java package called `manager`. Table 9.1 summarizes Java classes, which implement the management module according to the design described in chapter 6. The more detailed explanation of important classes follows in the next section. The Java documentation and the well commented source codes might be useful for a deeper understanding. Each class (unless it is not an inner class) is saved in a separate file of the same name with the extension ".java".

9.2 Classes description

BookMan

This is the main frame class of the bookmark manager. It contains the main method of the whole application and creates objects of the other manager's classes as the tree, table etc. The class is also responsible for the communication with the categorization module, more precisely, it creates an object of the `Catizer` class. And last but not least it creates and communicates with an object of the `Repository` class.

The `BookMan` class is a descendant of the `JFrame` class. The main frame is split into four parts, at the top is a tool bar of the class `JToolBar`, on the left a tree panel of the class `BookTree`, on the right a table panel of the class `BookTable` and at the bottom a status bar with labels of the class `JPanel`.

BookTree & BookTreeModel

The `BookTree` class is a descendant of the `JTree` class and is responsible for displaying the tree with categorical folders. The tree is built dynamically according a content of an object of the class `BookTreeModel`. This one is responsible for creating, removing and splitting folders and also for adding and removing bookmarks from a folder. The `BookTree` class also contains an inner class `DTListener`, that serves as a drop target for a bookmark dragged from the bookmark table.

Table 9.1: Java classes of the bookmark manager

Class name	Task of the class
`BookMan`	main class of the bookmark manager
`BookTree`	tree of categorical folders
`BookTreeModel`	model of the tree holding the content
`Folder`	folder in the tree
`BookTable`	table for records of bookmarks
`BookTableModel`	model of the table holding the content
`Preferences`	user's preferences
`BookReader`	reader of XML bookmark files
`BookWriter`	writer of XML bookmark files
`XBELFileFilter`	dialog filter for XML bookmark files
`BookFileChecker`	monitor of the browser bookmark file
`BookmarkFileFilter`	dialog filter for browser bookmark files
`ImportDlg`	dialog for a bookmark import
`TablePopupMenu`	pop-up menu for the table
`Recommender`	recommends related Web pages

Folder

The class is a descendant of the `DefaultMutableTreeNode` class and thus serves as a node of the `BookTree`. It holds a collection of bookmarked documents, the path to the root and the list of its children folders.

BookTable & BookTableModel

The `BookTable` class is a descendant of the `JTable` class and implements a table for bookmark's records. It contains an inner class called `DGListener`, that serves as the drag gesture listener for the dragging of a bookmark to another folder in the tree.

The content of the table is managed by an object of the `BookTableModel` class, which is a descendant of the `AbstractTableModel` class. It contains records of bookmarks from the currently selected folder. The records consist of a title, URL and a status icon. These items are updated every time when another folder is selected.

BookReader & BookWriter

These classes implement the reading and writing of bookmarks in and from an XML file. A saved bookmark file accomplishes requirements of the XML Bookmark Exchange Language v1.0 specification [XBEL99]. The bookmark reader can of course read the files written by the writer, but it cannot read any XBEL file. Nevertheless, this is not required, because another class called `BookmarkFileChecker` is suited for the import of different types of bookmarks of different browsers, including an import of the XBEL files.

Preferences

The class manages the user's preferences, their loading and saving from and to a preferences file (called `caterpref` and saved in the application directory). A user can set the following parameters: a type of the used Web browser, a bookmark file of the browser, a minimum and maximum number of bookmarks in a folder. Other parameters as time stamps when the bookmark file was checked for changes, the last checked URL and the last opened XML bookmark file are maintained without an interaction with a user.

The class also contains an inner class for displaying a preference dialog, which is called `PrefDialog`.

BookFileChecker

This class is a descendant of the `Thread` class and thus an object of this class runs in a separate thread. It monitors and checks a bookmark file corresponding to a Web browser for newly added bookmarks. It can handle with bookmark files of Netscape Navigator (all versions), Konqueror (version 2 and upper), Opera (all versions) and Internet Explorer (all versions). It checks a bookmark file in an interval (default is 10 minutes) for any change of the saved URLs. If a new URL is found, it is submitted for categorization and saved in a folder in CaterBook.

Recommender

This class implements the recommendation service. It can recommend related Web pages to the already categorized bookmarks. The service is based on the Web pages categorized in the Google Web directory in the same category as a bookmark. It fetches the Google Web directory page of a category and extracts top ten hyperlinks, which can be presented to the user.

Part IV

Testing & Evaluation

Chapter 10

Testing Datasets

We performed several tests in different stages of our work in order to evaluate each step of the bookmark processing. This chapter describes datasets we gathered and observations we made about the size of the term dictionary. Similarity measures among documents is evaluated and described in chapter 11. Another testing described in chapter 12 evaluates the effectiveness of our classifier and the last testing in chapter 13 focuses on user's experiences with our bookmark manager called CaterBook.

We gathered 27 different datasets of Web documents for testing purposes. They were fetched from several categories of the Open Directory Project [ODP]. The datasets differ in the number of documents, number of categories and setting of parameters for the indexing process. Table 10.1 summarizes the specifics of the datasets. The fifth column "term inclusion" refers to the method of creating the indexing dictionary, terms can be included either from all documents or only from megadocuments, which are the representatives of the involved categories. The sixth column "freq threshold" refers to a threshold of a term occurence within a document, these terms which occur less than the threshold are not omitted during the indexing. The last column refers to the number of terms which are extracted from a Web page, in case there are not enough terms, the hyperlinked Web pages from the same server are also indexed and a particular portion of terms from each page is included. Table 10.2 displays categories of ODP included in the datasets.

From the gathering of these datasets we observed the following phenomena:

1. The size of a term dictionary is much smaller (15-30%) when using the megadocument approach than the classic approach.

2. The threshold value for a term frequency set to 3 leads to a half-size term dictionary than when the threshold with value 2 is used. This phenomenon is also confirmed by our testing evaluating how often is a term repeated within a document. The result is depicted in a "pareto" graph in figure 10.1.

3. The reduction from 400 down to 300 extracted terms per document results in reduction of the dictionary size by 15 % in average.

4. The data does not allow us to make distinctive conclusions from a projection into a 2D graph by the principal component analysis, mainly because of the unnormal distribution of the data. We exploratively applied PCA to some of the datasets,

Table 10.1: Testing datasets

dataset no.	no. of documents	no. of categories	dictionary size	term inclusion	freq threshold	terms per page
1	180	18	4348	all	2	400
2	180	18	3798	all	2	300
3	180	18	2599	all	3	400
4	180	18	2201	all	3	300
5	180	18	1371	megadocs	2	400
6	180	18	1278	megadocs	2	300
7	180	18	630	megadocs	3	400
8	180	18	522	megadocs	3	300
9	100	5	2492	all	2	400
10	100	5	2183	all	2	300
11	100	5	1534	all	3	400
12	100	5	1254	all	3	300
13	100	5	418	megadocs	2	400
14	100	5	385	megadocs	2	300
15	100	5	202	megadocs	3	400
16	100	5	173	megadocs	3	300
17	60	3	1711	all	2	400
18	60	3	1418	all	2	300
19	60	3	927	all	3	400
20	60	3	726	all	3	300
21	60	3	255	megadocs	2	400
22	60	3	208	megadocs	2	300
23	60	3	118	megadocs	3	400
24	60	3	101	megadocs	3	300
25	10	2	428	all	2	400
26	10	2	474	all	2	400
27	10	2	500	all	2	400

Table 10.2: Categories included in the datasets

Category	**Datasets**
Arts>Literature>Poetry	1-8
Business>Investing>Investment Guides	1-8
Computers>Artificial Intelligence	1-16,25
Computers>Artificial Intelligence>Agents	26
Computers>Artificial Intelligence>Machine Learning	26
Computers>Artificial Intelligence>Machine Learning>Case-based reas.	27
Computers>Artificial Intelligence>Machine Learning>Datasets	27
Computers>Speech technology	25
Games>Gambling>Blackjack	1-24
Health>Fitness>Aerobics	1-16
Kids and Teens>School Time	1-16
News>Internet Broadcasts	1-8
Recreation>Climbing	1-24
Recreation>Climbing>Indoor	1-8
Recreation>Food>Drink>Wine	1-8
Recreation>Food>History	1-8
Reference>Dictionaries	1-8
Science>Astronomy	1-8
Science>Social Sciences>Psychology>Behavior Analysis	1-8
Shopping>Consumer Electronics>Audio	1-8
Society>Religion and Spirituality>Taoism	1-8
Sports>Snowboarding	1-8

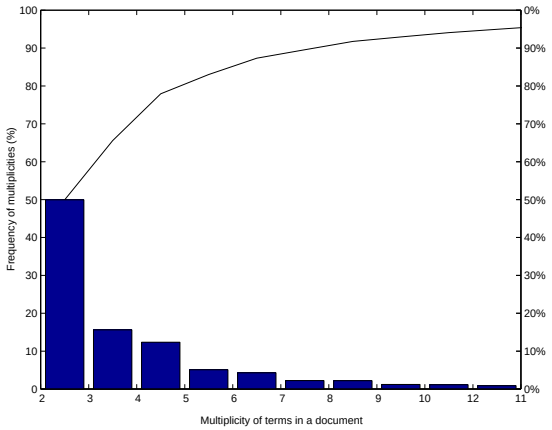

Figure 10.1: Pareto graph of term multiplicities

for the datasets no.21-24 we obtained distinctive clusters each comprising documents of same category, but in other cases it led to low variances captured by the first and second principal component and thus the results were useless.

These observation give an idea how the term dictionary can be reduced which would be a big asset for the classification computations. The influence of these changes on the similarity measures between documents is described in the following chapter 11.

Chapter 11

Document similarity testing

11.1 Similarity visualization by means of Sammon's mapping

In this chapter we discuss evaluation of similarity measurements among Web documents, after they were indexed by the process described in section 7.2. The similarity is measured by cosine of angle between document vectors, where the vector elements are weightened by TFIDF weightening. We tested 180 Web documents from 18 categories, i.e. the datasets 1-8 as described in table 10.1. For each dataset we calculated a distance matrix and applied the Sammon's mapping technique for visualization of similarities among documents in a 2D graph. Each symbol in the mappings represents a document of a category, the included categories with their symbols (the legend) are described in figure 11.9. Documents within a same category should be close together and compose separate clusters. The quality of clustering varies for different datasets, as it is observable in the following figures. Each couple of figures displays the Sammon's mappings of datasets indexed by using the classical or megadocument approach, respectively.

We observed that the results with a significantly reduced dictionary by the megadocument approach are comparable or even better than the results achieved by the classic approach. Thus we concluded there is no reason for using huge dictionaries and we decided for megadocuments in the design and implementation of our system. The best result was achieved with the dataset no.7 (figure 11.6), where the term frequency threshold is set to 2, the number of extracted terms are 400 and the megadocument approach is used for the dictionary definition. When the threshold is increased (figures 11.4 and 11.8) or the number of terms decreased (figure 11.2 and 11.4), the clustering of documents is not so remarkable. The calculation of the Sammon's mapping was carried out several times for confirmation of the presented observations.

11.2 Similarity tests on different categorical levels

In order to discover if a deepness of categories in the hierarchical tree influences similarity among Web documents within a category and between different categories, we performed three tests, each on different level of the ODP hierarchy. The datasets no.26, 27 and 28 comprise two categories on the second, third and fourth level of the ODP

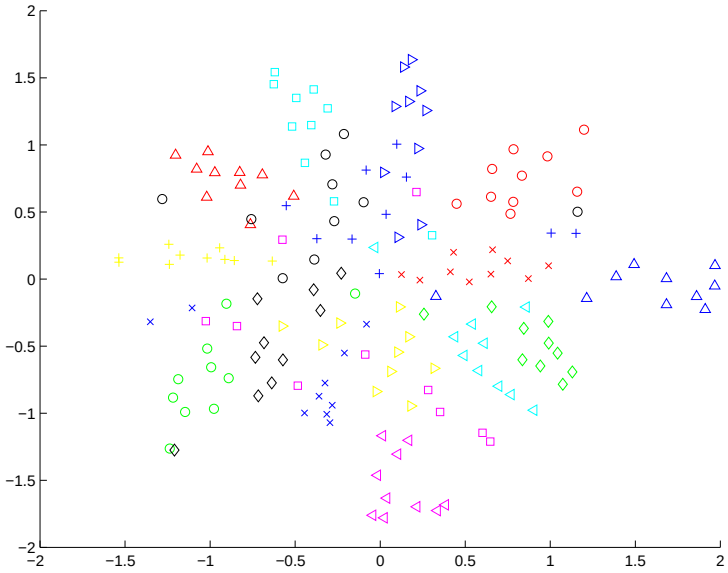

Figure 11.1: Sammon's mapping, dataset no.1

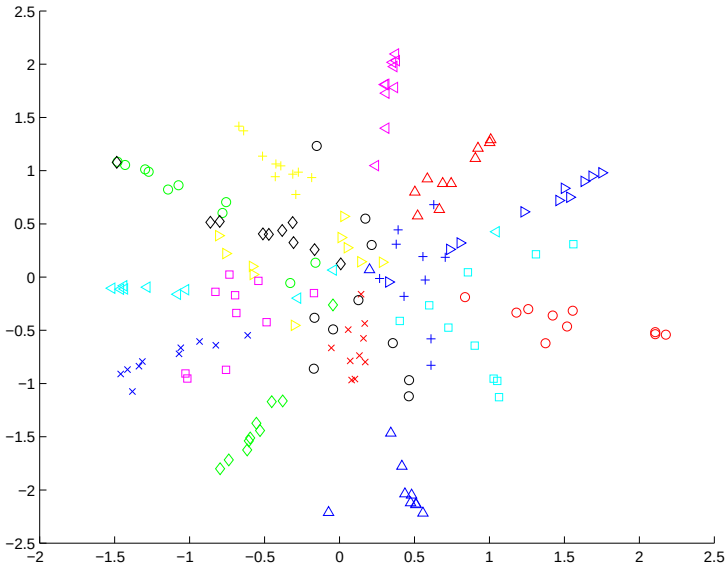

Figure 11.2: Sammon's mapping, dataset no.5

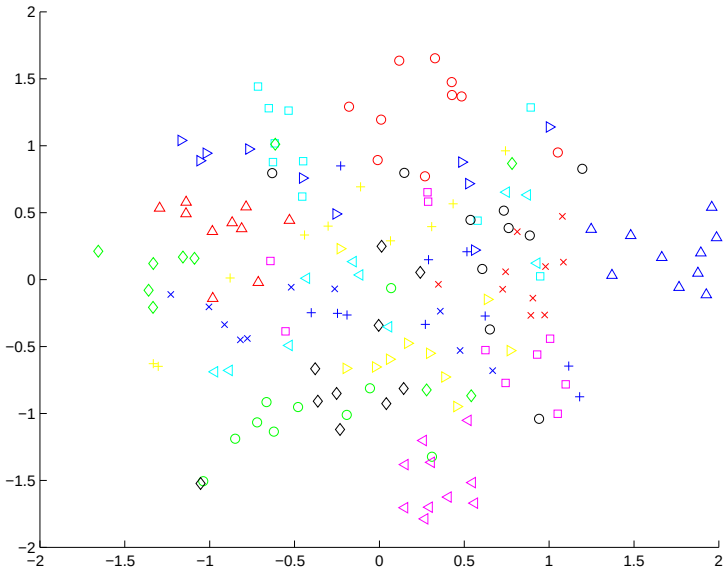

Figure 11.3: Sammon's mapping, dataset no.2

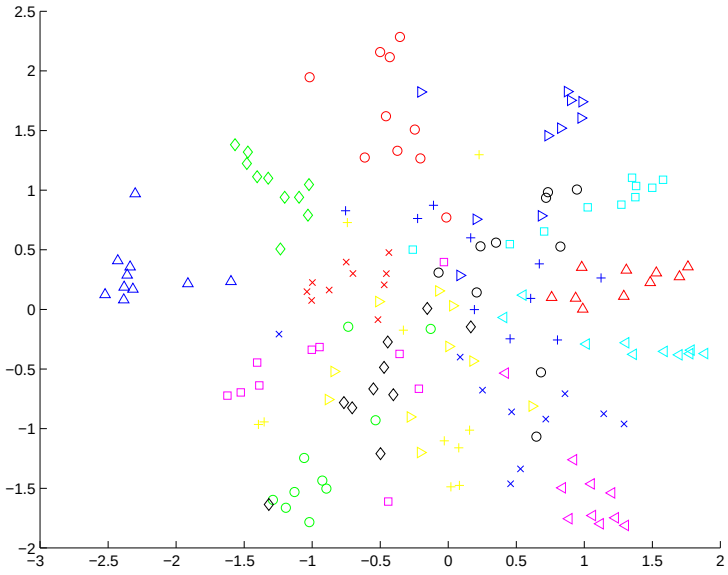

Figure 11.4: Sammon's mapping, dataset no.6

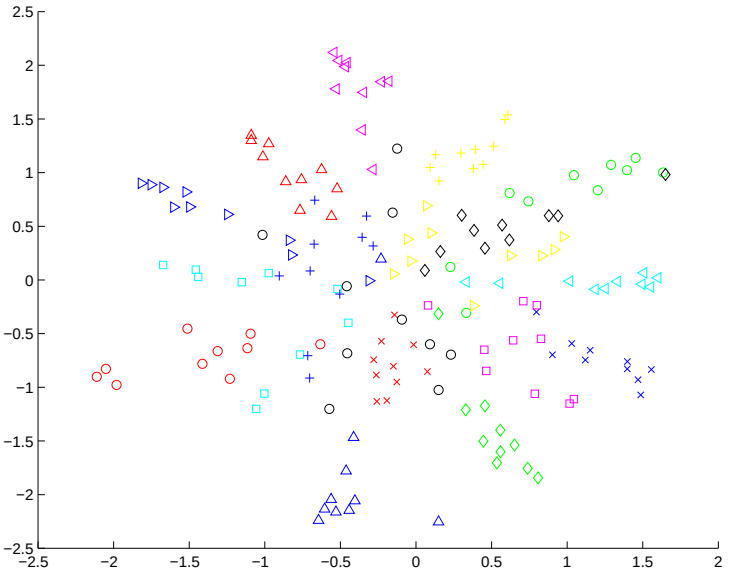

Figure 11.5: Sammon's mapping, dataset no.3

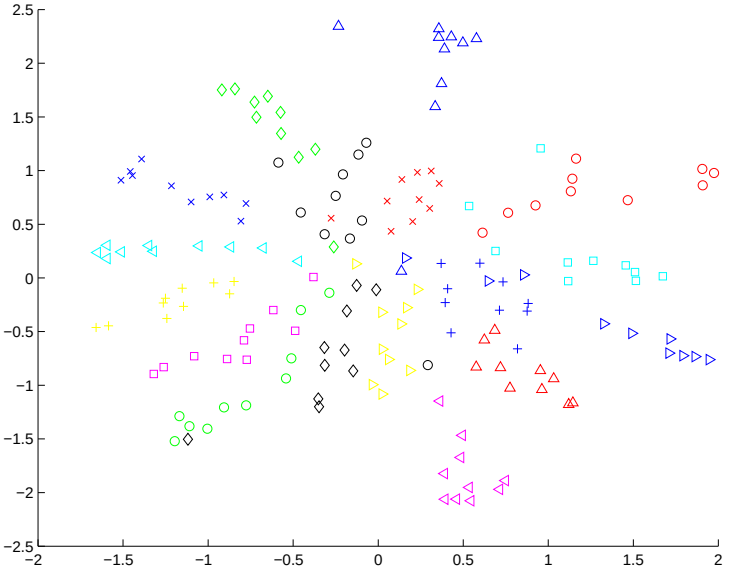

Figure 11.6: Sammon's mapping, dataset no.7

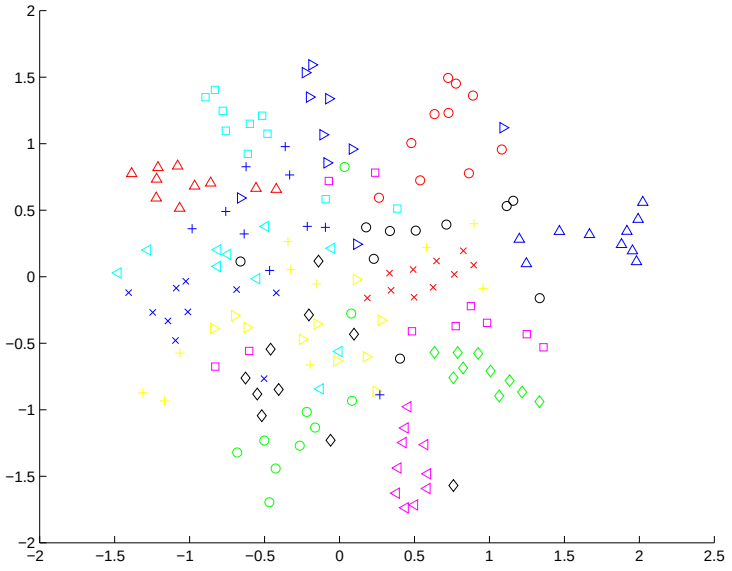

Figure 11.7: Sammon's mapping, dataset no.4

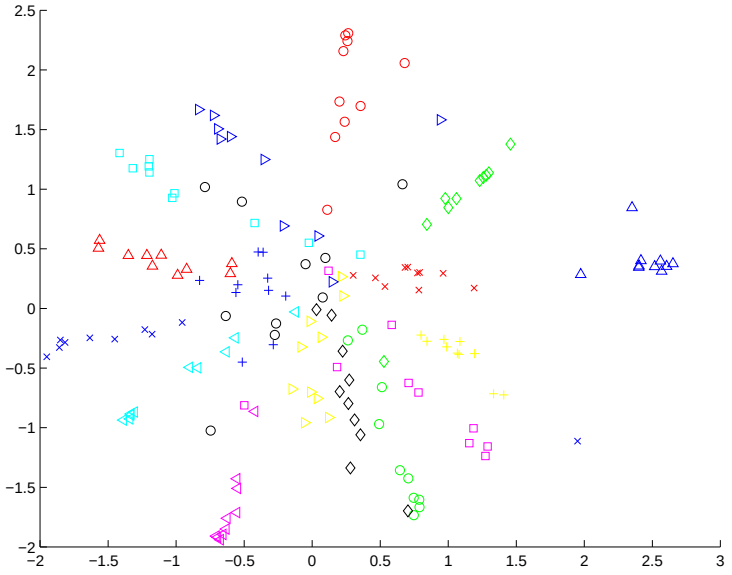

Figure 11.8: Sammon's mapping, dataset no.8

Figure 11.9: Legend to the Sammon's mapping figures

hierarchy, respectively. Similarities were measured as cosine of angle of normalized vectores and in the following sections they are presented as percentage values.

Testing on the second level

Table 11.1 summarizes the similarity among documents from the "Artificial Intelligence" ($AI_1, \ldots, AI_5$) and "Speech technology" ($ST_1, \ldots, ST_5$) categories, which are children of the category "Computers". The mean of similarities of documents within the first and within the second category is 14% and 7%, respectively, thus the first category is much more coherent than the second one. The mean of similarities between documents among those two categories is 1%.

Table 11.1: Similarity between the "Artificial intelligence" (AI) and "Speech technology" (ST) categories

	AI_1	AI_2	AI_3	AI_4	AI_5	ST_1	ST_2	ST_3	ST_4	ST_5
AI_1	x	38%	19%	11%	12%	3%	0%	3%	1%	2%
AI_2	38%	x	18%	13%	13%	1%	0%	1%	1%	0%
AI_3	19%	18%	x	5%	4%	0%	1%	1%	2%	0%
AI_4	11%	13%	5%	x	9%	0%	0%	0%	2%	1%
AI_5	12%	13%	4%	9%	x	1%	1%	2%	2%	1%
ST_1	3%	1%	0%	0%	1%	x	3%	14%	7%	3%
ST_2	0%	0%	1%	0%	1%	3%	x	3%	2%	5%
ST_3	3%	1%	1%	0%	2%	14%	3%	x	6%	5%
ST_4	1%	1%	2%	2%	2%	7%	2%	6%	x	26%
ST_5	2%	0%	0%	1%	1%	3%	5%	5%	26%	x

Testing on the third level

Table 11.2 summarizes the similarity among documents from the "Machine learning" ($ML_1, \ldots, ML_5$) and "Agents" ($Ag_1, \ldots, Ag_5$) categories, that are children of the category "Artificial Intelligence". The mean of the similarites of documents within the

first and within the second category is 5% and 11%, respectively. Thus, the "Machine learning" category seems rather incoherent. The mean of similarities among those two categories is 2%.

Table 11.2: Similarity between the "Machine learning" (ML) and "Agents" (Ag) categories

	ML_1	ML_2	ML_3	ML_4	ML_5	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
ML_1	x	8%	2%	3%	2%	0%	0%	0%	1%	1%
ML_2	8%	x	6%	8%	8%	4%	2%	3%	3%	2%
ML_3	2%	6%	x	3%	3%	1%	0%	0%	1%	1%
ML_4	3%	9%	3%	x	6%	4%	4%	4%	4%	4%
ML_5	2%	8%	3%	6%	x	1%	1%	1%	2%	5%
Ag_1	0%	4%	1%	4%	1%	x	24%	10%	30%	18%
Ag_2	0%	2%	0%	4%	1%	23%	x	4%	19%	7%
Ag_3	0%	3%	0%	4%	1%	10%	4%	x	9%	5%
Ag_4	1%	3%	1%	4%	2%	30%	19%	9%	x	7%
Ag_5	1%	2%	1%	4%	5%	18%	7%	5%	7%	x

Testing on the fourth level

Table 11.3 summarizes the similarity among documents from the “Case-based reasoning” ($CR_1, \ldots, CR_5$) and “Datasets” ($DS_1, \ldots, DS_5$) categories, that are children of the category "Machine Learning".

The mean of similarites of documents within the first and within the second category is 20% and 4%, respectively. The mean of similarities among those two categories is 2%. The category "Case-based reasoning" seems to be very coherent except the CR_2 document, which is a Web page containing only personal names of people interested in case-based reasoning, so there is no wonder why the similarity of this document with the others of the same category is so low. The "Datasets" category is rather incoherent, mainly because of the lack of the common terms.

Table 11.3: Similarity between the "Case-based reasoning" (CR) and "Datasets" (DS) categories

	CR_1	CR_2	CR_3	CR_4	CR_5	DS_1	DS_2	DS_3	DS_4	DS_5
CR_1	x	3%	21%	26%	27%	1%	0%	0%	0%	0%
CR_2	3%	x	6%	3%	3%	3%	1%	0%	0%	0%
CR_3	21%	6%	x	38	24%	3%	1%	1%	0%	1%
CR_4	26%	3%	38%	x	39%	0%	1%	0%	2%	0%
CR_5	27%	3%	24%	39%	x	0%	1%	0%	2%	1%
DS_1	1%	3%	3%	0%	0%	x	5%	2%	3%	4%
DS_2	0%	1%	1%	1%	1%	5%	x	3%	2%	5%
DS_3	0%	0%	1%	0%	0%	2%	3%	x	2%	5%
DS_4	0%	0%	0%	2%	2%	3%	2%	2%	x	4%
DS_5	0%	0%	1%	0%	1%	4%	5%	5%	4%	x

Conclusion of the level testing

The assumption that documents in a deeper level have more specific vocabulary and thus the similarity between documents of a category would be higher was not approved. It varies from category to category and a level in the hierarchy has no impact on it.

There are some categories which are very incoherent in the hierarchy, and thus we conclude that it is not advisable to use e.g. k nearest neighbour or centroid-based document classification method, but rather the megadocument approach, which clears noisy words and creates a better category-specific dictionary.

Chapter 12

Testing of the categorization effectiveness

12.1 Training and test data

The megadocument classifier was trained and tested on 18 categories of ODP (see table 12.1). The training of the classifier was carried out with data obtained by means of Google, i.e. by indexing megadocuments from the top ranked Web pages present in the Google Web directory. Maximum of 100 terms per page was extracted until the whole megadocument reached the number of 500 terms. It means approx. 5 Web pages from Google were used.

The test data were gathered from corresponding categories of ODP, with omitting the pages which were used for the training megadocuments. Additionally, there was another portion of Web pages, which was also deleted form the test dataset. It was those Web pages that were categorized in the Google Web directory and Google gave in response to a query for related pages the same ones as a top result with already assigned category. Thus, there would be no task for our classifier and for this reason also these Web pages were not used for the testing. It would be rather discreditable and the results of our testing would be strongly biased, if we test these simple cases. Only the Web pages, for which the classification was considerable including more candidate categories, were presented in the test-bed.

The indexing of test pages was carried out as follows: the number of 400 terms was extracted from each test document, all the terms in the initial Web page and the rest from linked pages within the same server. Further details about the indexing, training and classification process are described in chapters 7 and 8 (design and implementation).

12.2 Evaluation method

The evaluation is based on the classic notion of precision with adaptation to the text categorization, as described in [Govert99]. Precision with respect to a category C_i is defined as the conditional probability, that if a random document d is classified under C_i, this decision is correct. This probability may be estimated as

$$\hat{Pr}_i = \frac{TP_i}{TP_i + FP_i}$$

where TP_i denotes the number of documents correctly classified under C_i (true positives with respect to C_i) and FP_i denotes the number of incorrectly classified documents (false positives). For the whole category set of size m, we use the micro-averaging evaluation, which is obtained by global summing over all individual decisions, i.e.

$$\hat{Pr} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m} (TP_i + FP_i)}$$

12.3 Results

Table 12.1 shows the results of the testing. The first column displays the names of categories in ODP, the second column the number of correctly categorized Web documents and the third column the total number of the tested Web documents in a category. From this we calculated the micro-averaged precision, considering the 18 categories together with other categories which Google suggested as "false" candidates in the pre-classification. A decision of the classifier to classify a document to one of these “false” categories was naturally evaluated as incorrect, thus as a "false positive" (FP) for a category. A document which was categorized into its training category (or into its more specific subcategory) was evaluated as a "true positive" for a category.

We achieved a value of 73 % for the micro-averaged precision. This is comparable to the results obtained in text categorization applied to the standard text collections like Reuters, which varies between 70 and 80 % (see [Yang99]). When we compare our system to the recent systems for Web document categorization, our result is much better than those reported by e.g. [Govert99] and [Klas00], who achieved only max. 37% and 51% precision, respectively. Their evaluation was carried out with a testbed created from Yahoo!, however there should not be a big difference to ODP.

We accredit our favorable results to the pre-classification process which significantly narrows the number of candidate categories, i.e. the amount of training data for a tested bookmark. This approach brought significant improvement to our system.

Table 12.1: Effectiveness of the classifier

Category	Correct	Total
Arts>Literature>Poetry	21	23
Business>Investing>Investment Guides	25	60
Computers>Artificial Intelligence	20	24
Computers>Artificial Intelligence>Machine Learning	10	16
Games>Gambling>Blackjack	21	21
Health>Fitness>Aerobics	37	48
Kids and Teens>School Time	43	67
News>Internet Broadcasts	11	19
Recreation>Climbing	25	29
Recreation>Climbing>Indoor	7	10
Recreation>Food>Drink>Wine	27	31
Recreation>Food>History	16	22
Reference>Dictionaries	5	15
Science>Astronomy	16	16
Science>Social Sciences>Psychology>Behav. Analysis	12	16
Shopping>Consumer Electronics>Audio	25	31
Society>Religion and Spirituality>Taoism	25	26
Sports>Snowboarding	20	25
In total	**366**	**499**

Chapter 13

Testing of the system by users

We have asked some of our colleagues to test the system, that is to install CaterBook, import their recent bookmarks, let them automatically categorize, evaluate the results and fill a questionary. We have addressed 15 people and we have got 9 responses. The users were asked about the following in the questionary :

1. The total number of bookmarks a user imported to Caterbook.
2. The number of bookmarks in the folder "Others".
3. The number of non-English bookmarks in the folder "Others".
4. The number of unreachable bookmarked Web pages in the folder "Others".
5. The number of bookmarks which were according to user's opinion wrongly categorized.
6. The number of those wrongly categorized bookmarks, which have even the top category wrongly assigned.
7. How were the user satisfied with CaterBook, with options: very frustrated, unsatisfied, neutral, contented, very keen.
8. Whether the user will use CaterBook in future, if some additional features are implemented, with options: absolutely not, probably not, maybe, likely, surely.

Table 13.1 summarizes users' responses, where each column refers to one question. We can see that number of bookmarks per user differs a lot, some people we asked for testing even do not use bookmarks, however some of them use bookmarking very heavily.

Figure 13.1 shows categorization results evaluated by users, that is six ratios for each user. Meaning of abbreviations in the legend is the following: tot refers to the total number of user's bookmarks, val to the number of valid (English and not obsolete) bookmarks, $class$ to the number of classified (the pre-classification by means of Google suggested candidate categories and the classification was performed), cor to the number of correctly categorized bookmarks (according to a user) and $topcor$ to the number of correctly assigned to a top category.

Figure 13.2 shows an example of a part of user's categorized bookmarks concerning the artifical intelligence area. Icons on the right side represent the user's evaluation about the correct and incorrect categorization.

Table 13.1: Results of the user's testing

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
user	**total**	**others**	**non-eng.**	**unreach.**	**wrong**	**abs.wr.**	**satisf.**	**future**
1	166	92	10	33	28	11	neutral	maybe
2	147	65	23	5	21	11	content	maybe
3	72	36	14	10	8	7	content	maybe
4	880	445	101	54	40	13	content	maybe
5	53	21	10	8	3	1	content	maybe
6	180	70	25	37	29	10	neutral	likely
7	20	7	3	4	2	1	content	maybe
8	89	30	2	15	12	7	content	maybe
9	359	192	58	20	19	10	content	likely

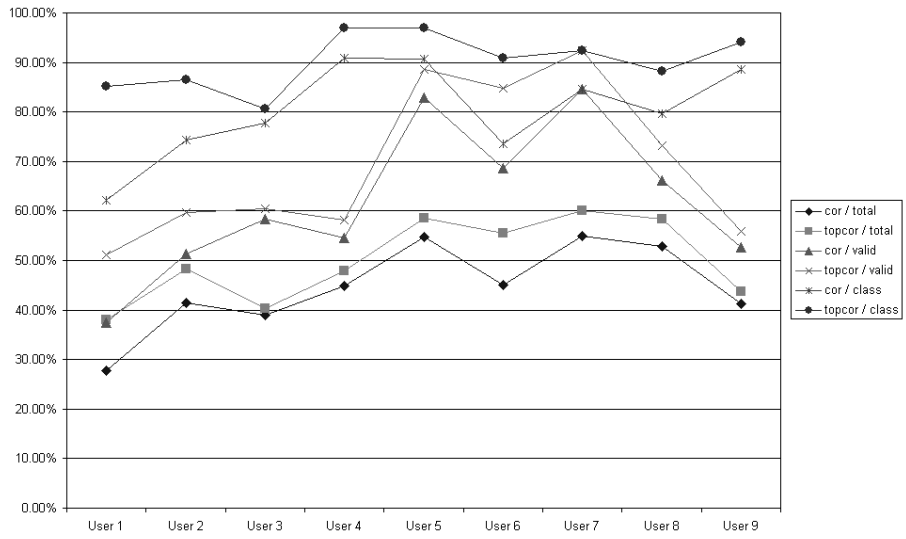

Figure 13.1: Categorization precision evaluated by users

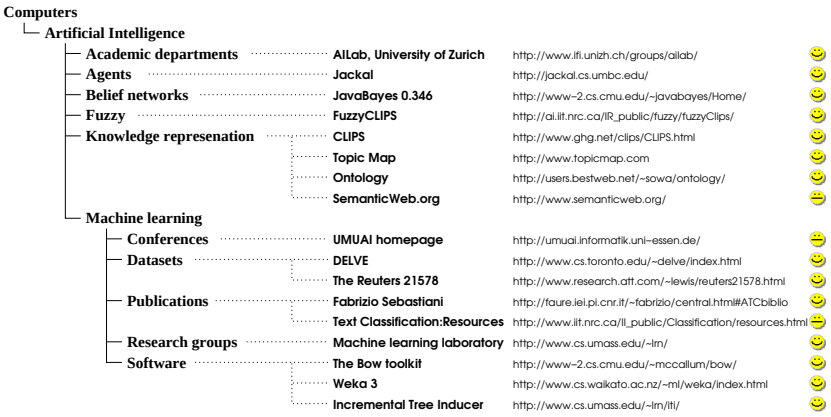

Figure 13.2: Example of categorization of user's bookmarks concerning the artificial intelligence

Since the testing has been carried out with such a small sample of users, we cannot make any significant conclusion. However, we consider the users' evaluations as a valuable preview of the functionality of our system. Additionally, the users gave us some more comments about their experiences with the system. They appreciated the recommendation service and found it beneficial. On the other hand, they judged negatively that the categorization system does not take into account the original structure of a bookmark file during its import.

Chapter 14

Conclusions

In this report, we have addressed the issue of adaptive and automated bookmark management. We have investigated the usage of information retrieval and machine learning techniques for the organizing bookmarks in an automated way and we have concluded the automated text categorization can be applied to our problem. We have had to overcome difficulties with a heterogeneity of Web documents, that influences performance of a classification. We have solved it by the megadocument approach for the training data processing, which wipes out noisy words and preserves the significant feature terms. Another problem has emerged with the training of our classification system for such a broad area of topics that bookmarks can comprise. We have solved this issue by a "pre-classification", where our agent negotiates with the Google search engine for candidate categories for a particular bookmark. After obtaining these candidates, the classifier can be trained on the data contained in these categories and can classify the bookmark. The negotiation has shown to be a crucial and volatile point of the categorization process, because if a correct category is not among the candidates, the following classification cannot result in a right decision. However, such a pre-classification has shown to be necessary because of the classification computational costs.

We have suggested the organization of bookmarks in a hierarchy of categories, based on the structure of ODP. In order of adaptability the user's hierarchy is built dynamically according additions and removals of bookmarks. The number of bookmarks in a categorical folder can be constrained by maximum and minimum thresholds, which can lead to the splitting and merging of folders. The idea of propagation of bookmarks to the parent categorical folder has been presented.

Bookmarking is not the only way, how to manage a user's area of interesting Web pages. We have concluded that our system can be more than just a bookmark manager and we have integrated an assistant service for recommendation of similar Web pages to the bookmarked ones. This feature has been often positively judged by users during their evaluation of the system.

The testings, which have been carried out, have helped us in every stage of the work and revealed several interesting aspects about documents on the Web with respect to their automated categorization and application for bookmarks. Then, the designed system has been fully implemented as a client-side application and named CaterBook. It has been coded in Java programming language and thus it is a platform independent. The prototype version is freely available and we hope it will facilitate a user's work with the Web.

14.1 Future work

There is a lot of space for a future research in the area of adaptive and automated bookmark management. We see some issues which can be improved in the categorization module of our system. Several different techniques can be used for the feature extraction from Web pages, considering their structuring and hyperlink cohesion. It is also desirable to explore other techniques for the representation of Web documents, e.g. the latent semantic indexing.

The bookmark management and adaptability to a user can be improved in several aspects. Our system does not take in account the recent structure of a user's bookmark file, namely the folders, which were already created by a user. Combination of a user's structuring and the categorical hierarchy of ODP can bring a new quality to the bookmark management. Other issue, which was not yet addressed, is the placement of a bookmark to more than one category. It is considerable for many Web pages, that more than one category can fit and thus it would facilitate a searching when a bookmark is placed in more categories. The very essential improvement we see in a searching interface for bookmarks, not only searching in titles and URLs, but also in the content of bookmarked Web pages. Because the feature keywords are extracted by the system during the categorization process, it should be straightforward to implement a simple search engine, which would facilitate searching for bookmarks. Other visualization of bookmarks than the hierarchy of categories can be investigated, e.g. tree-map visualization, and thus the system could offer multiple views to the user's bookmarks. There is also a small step towards user modeling, i.e. the evaluation of user's areas of interest, their specifity, intensity and variability in time.

We are strongly encouraged to continue with our research in order to facilitate bookmark management and personalization on the World Wide Web.

Bibliography

[Abrams98] D. Abrams, R. Baecker, M. Chignell. Information Archiving with Bookmarks: Personal Web Space Construction and Organization. In *Proceedings CHI-98, Conference on Human Factors in Computing Systems*, pp 41-48, Los Angeles, USA, 1998.

[Attardi99] G. Attardi, A. Gullí and F. Sebastiani. Automatic Web Page Categorization by Link and Context Analysis. In *Proceedings of THAI-99, 1st European Symposium on Telematics, Hypermedia and Artificial Intelligence*, pp. 105-119, 1999.

[Bartfai94] G. Bartfai. Hierarchical Clustering with ART Neural Networks. In *Proceedings of the IEEE International Conference on Neural Networks, volume 2*, pp. 940-944. 1994.

[Boley99] D. Boley, M. Gini, R. Gross, E. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, J. Moore. Partitioning-Based Clustering for Web Document Categorization. *Journal of Decision Support Systems. Vol 27, no. 3*, pp. 329-341, 1999.

[Belew00] R.K. Belew. Finding Out About: a cognitive perspective on search engine technology and the WWW. Cambridge University Press, 2000.

[Chakrabarti98] S. Chakrabarti and B. E. Dom and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, pp. 307-318, ACM Press, New York, US, 1998.

[Chekuri97] C. Chekuri and P. Raghavan. Web Search Using Automatic Classification. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara, CA, USA, 1997.

[Chen00] H. Chen and S.T. Dumais. Bringing order to the Web: Automatically categorizing search results. In *Proceedings of CHI'00, Conference on Human Factors in Computing Systems*, The Hague, NL, 2000.

[Chen98] L. Chen and K. Sycara. Webmate : A personal agent for browsing and searching. In *Proceedings of 2nd International Conference on Autonomous Agents*, Minneaoplois, USA, 1998.

[Chuang00] W. Chuang, A.Tiyyagura, J.Yang, and G.Giuffrida. A Fast Algorithm for Hierarchical Text Classification. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK'00)*, pp. 409-418. London-Greenwich, UK, 2000.

[Cutting92] D.R. Cutting, D.R. Karger, J.O. Pedersen and J.W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *Proceedings of the 15th Annual International Conference on Research and Development in Information Retrieval (SIGIR'98*), Copenhagen, 1992.

[Deerwester90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science, 41(6)*. pp. 391-407, September 1990.

[Frommholz01] I. Frommholz. Categorizing Web Documents in Hierarchical Catalogues. In *Proceedings of ECIR-01, 23rd European Colloquium on Information Retrieval Research*, Darmstadt, Germany, 2001.

[Glover02] E. J. Glover, K. Tsioutsiouliklis, S. Lawrence, D. M. Pennock and G. W. Flake. Using Web structure for classifying and describing Web pages. In *Proceedings of WWW-02, International Conference on the World Wide Web*, Honolulu, Hawai, 2002.

[Google] L.Page and S.Brin. Google. 1997. `http://www.google.com`

[Govert99] N. Govert, M. Lalmas, and N. Fuhr. A probabilistic description-oriented approach for categorising Web documents. In *Proceedings of the 8th International Conference on Information and Knowledge Management*, pp. 475-482, ACM, New York, 1999.

[Han01] J. Han. Data mining:concepts and techniques. Morgan Kaufmann, San Francisco, 2001.

[Kaasten01] S. Kaasten and S. Greenberg. Integrating Back, History and Bookmarks in Web Browsers. In *Extended Abstracts of the ACM Conference of Human Factors in Computing Systems (CHI'01)*, pp.379-380, ACM Press. 2001.

[Keller97] R.M. Keller, S.R. Wolfe, J.R. Chen, J.L. Rabinowitz and N. Mathe. Bookmarking Service for Organizing and Sharing URLs. In *Proceedings of the 6th World-Wide Web Conference*, Santa Clara, USA, 1997.

[Kessler97] B. Kessler, G. Nunberg and H. Schütze. Automatic detection of text genre. In *Proceedings of ACL-97, 35th Annual Meeting of the Association for Computational Linguistics*, pp. 32-38, Morgan Kaufmann Publishers, San Francisco, US, 1997.

[Klas00] C. Klas and N. Fuhr. A new Effective Approach for Categorizing Web Documents. Proceedings of BCSIRSG-00, the 22nd Annual Colloquium of the British Computer Society Information Retrieval Specialist Group, 2000.

[Labrou99] Y. Labrou and T. Finin, Yahoo! As an Ontology: Using Yahoo! Categories to Describe Documents. In *Proceedings of 8th International Conference on Knowledge and Information Management (CIKM-99)*, Kansas City, MO. 1999.

[Maarek96] Y.S. Maarek I.Z.B. Shaul. Automatically Organizing Bookmarks per Contents. In *Proceedings of the 5th International World Wide Web Conference*. Paris, France, 1996.

[Mase98] H. Mase. Experiments on Automatic Web Page Categorization for IR system. Technical report, Stanford University, 1998.

[Mladenic98] D. Mladenic. Turning Yahoo! into an Automatic Web-Page Classifier. *Proceedings of the 13th European Conference on Aritficial Intelligence ECAI'98*, pp. 473-474. 1998.

[Mladenic99] D. Mladenic. Machine learning used by Personal WebWatcher. *Proceedings of ACAI-99, Workshop on Machine Learning and Intelligent Agents*, Chania, Crete, 1999.

[Mitchell97] T. Mitchell. Machine Learning. McGraw Hill, 1997.

[ODP] Open Directory Project. 1998. `http://www.dmoz.org`

[Pazzani96] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: identifying interesting Web sites. In *Proceedings of 13th National Conference on Artificial Intelligence*, pp.54-61. Portland, USA, 1996.

[Porter88] M.F. Porter. An algorithm for suffix stripping. In *Program, 14 no. 3*, pp. 130-137, July 1980.

[Rasmussen92] E. Rasmussen, editorials W. B. Frakes and R. Baeza-Yates. Information Retrieval: Data Structure and Algorithms. Chapter 16: Clustering Algorithms, pp.419-442. Prentice Hall, USA, 1992.

[Reuters] Reuters text categorization test collection.
`http://www.research.att.com/~lewis/reuters21578.html`

[Ridley00] J.S. Ridley.A summative evaluation of the Google search engine. 2000.
`http://www.atomicrevs.net/jaki/google/test.html`

[Rijsbergen79] C. J. van Rijsbergen, Information Retrieval. Butterworths, London. 1979.

[Roth98] D. Roth. Learning to resolve natural language ambiguities: a unified approach. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pp. 806-813, AAAI Press, Menlo Park, US, 1998.

[Rucker97] J. Rucker and J.P. Marcos. Siteseer: Personalized Navigation for the Web, *Communications of the ACM*, pp. 73-75, Vol. 40, no. 3, March 1997.

[Rumehart86] D. E. Rumehart, G. E. Hinton, and J. L. McClelland. Parallel Distributed Processing, chapter A General Framework for Parallel Distributed Processing, pp. 45-76. The MIT Press, Cambridge, MA, 1986.

[Rusell95] S. Russell and P. Norwig. Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.

[Salton88] G. Salton and C. Buckley. Term-weightining approaches in automatic text retrieval. *Information Processing and Management 24, 5*, pp. 513-523.1998.

[Sebastiani99] F. Sebastiani. A tutorial on automated text categorisation. In *Proceedings of ASAI-99, 1st Argentinian Symposium on Artificial Intelligence*, pp. 7-35. Buenos Aires, AR, 1999.

[SemanticWeb] W3C Semantic Web. `http://www.w3.org/2001/sw/`

[Webclient01] Webclient v1.1 . 2001. `http://www.mozilla.org/projects/blackwood/webclient/`

[XBEL99] The XML Bookmark Exchange Language (XBEL), version 1.0. 1999. `http://www.oasis-open.org/cover/xbel.html`

[XUL01] XML User Interface Language (XUL) 1.0 . 2001. `http://www.mozilla.org/projects/xul/xul.html`

[Yahoo!] D. Filo and J.Yang. Yahoo! 1994. `http://www.yahoo.com`

[Yang99] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22th Annual International Conference on Research and Development in Information Retrieval (SIGIR'99)*, pp. 42-49, 1999.

[Zamir98] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21th Annual International Conference on Research and Development in Information Retrieval (SIGIR'98*), 1998.

Note: all the Internet addresses were checked and operational on $April 12^{\text{th}}, 2002$.

Appendix A

User's manual to CaterBook

Overview

CaterBook is a bookmark management tool, which automatically organizes user's bookmarks in a Yahoo-like hierarchy of categories. At the first execution it imports user's bookmark file and categorizes the bookmarked Web pages. The hierarchy of categories is dynamically built during the categorization process. After the categorization finishes, the user can easily browse through the hierarchy, manipulate with the bookmarks and add new bookmarks. A user can also use CaterBook without importing his recent bookmarks, just by testing it with arbitrary URLs.

Features

- automated classification of English written Web pages and categorization into appropriate folders.
- dynamical building and maintaining of the hierarchy of categorical folders.
- monitoring of a browser's bookmark file and checking for newly added bookmarks. When a new bookmark is detected, it is automatically submitted for classification and sorted into a categorical folder in the hierarchy. The bookmark file import and monitoring currently works with Mozilla , Netscape Navigator, Konqueror, Opera and Internet Explorer in all their versions.
- loading and saving of bookmarks in the XML format. The format of a saved file fulfill the XML Bookmark Exchange Language (XBEL) standard (`http://www.oasis-open.org/cover/xbel.html`).

System Requirements

The application is written in 100% pure Java and it can run on any Java 2 platform. For a complete list of platforms that support Java, please check `http://www.javasoft.com/cgi-bin/java-ports.cgi`

System requirement for hardware is basically the same as that of any Java 2 environment, 128 MB RAM is recommendable. Additionally it requires a fast and stable Internet connection (T1/T3/DSL) and approximately 1 MB of free space on a hard disc (200 kB for the application and 800 kB for the working data).

Platforms which were tested for running CaterBook:

- Sun Ultra 5, 128 MB RAM, Solaris SunOS 5.7
- PC Pentium II, 128 MB RAM, Linux 2.4.18, Debian distribution
- PC Pentium II, 128 MB RAM, Windows NT 4.0, service pack 6a
- PC Pentium III, 256 MB RAM, Windows 2000 Professional edition

Download

The newest version of CaterBook, packed in a zipfile can be downloaded at http://caterbook.can.com .

Installation

1. Make sure that you have installed Java 2 Runtime Environment (1.2 or later), which is free for download at Sun's Java site http://java.sun.com/ j2se/. If not, please download a free copy at this site and install it according to the installation instruction. If you have installed JDK that already include Java virtual machine, you don't need to install Java Runtime any more.

2. Choose a directory of your choice and unzip the file you downloaded to the directory. A subdirectory called *caterbook* with the application is automatically created. There is no additional installation.

Uninstallation

Simply delete the whole directory, where you have installed CaterBook.

Starting CaterBook in Windows

1. Go to the CaterBook's directory and find the *caterbook.jar* file. Try double clicking on it and CaterBook should start up. If this fails you need to change the application which Windows uses for opening a .jar file. Right-click on the *caterbook.jar* file and choose: "Open With... > Choose the program... >". Browse the list with programmes you will see and choose *javaw* and if applicable, set it with an option as *javaw -jar*. Don't forget to mark "Always use this program to open these files".

2. The simplest way to start-up CaterBook in future times is to make a shortcut to your desktop from the *caterbook.jar* file (just right-click again to make the shortcut). Please, do not move the file itselft from the directory, otherwise the application will not work properly.

3. When you start CaterBook for the first time, a window with the user's preferences will appear (see figure A.1). If you want to import your recent bookmarks, please choose there which browser you use and the location of your bookmark file or "favorites" directory. The usual location of MS Internet Explorer "Favorites" directory is in your profile directory, e.g. "C:\Documents and Settings*username*\Favorites" or
"c:\Windows\Profiles*username*\Favorites".

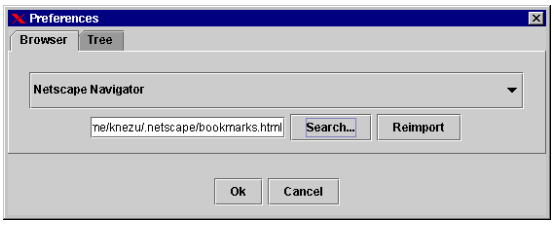

Figure A.1: Preferences dialog

4. The first import of your bookmark file can take a really long time (for 100 bookmarks approximately half an hour). Don't worry about the application and let it run in the background. If you skip the initial categorization process or the system crashes (which should never happen!), restart the application and click on the button "Reimport" in the preferences dialog.

Starting CaterBook on Unix systems

1. Go to the caterbook directory and find the *caterbook.jar* file. Start the application by *java -jar caterbook.jar*. If you want to launch the application from another directory, start it by *java -jar PATH/caterbook.jar*, where *PATH* is the path to the location of *caterbook.jar*.
2. When you start CaterBook for the first time, a window with the user's preferences will appear (see figure A.1). Please choose there which browser you use and the location of your bookmark file. Finding of the location might be difficult, try the following paths:
   (a) Netscape Navigator: *$HOME/.netscape/bookmarks.html*
   (b) Mozilla: *$HOME/mozilla/bookmarks.html*
   (c) Konqueror: *$HOME/kde/bookmarks.xml*
3. The first import of your bookmark file can take a really long time (for 100 bookmarks approximately half an hour). Don't worry about the application and let it run in the background. If you skip the initial categorization process or the system crashes (which should never happen!), restart the application and click on the button "Reimport" in the preferences dialog.

Categorization of bookmarks

The categorization process is based on finding of similarity between the bookmarked Web page and Web pages already categorized in the Google Web directory (`http://directory.google.com`) . It happens CaterBook fails in the categorization of bookmarks, because no similarity was found or the bookmarked URL is obsolete. In this case the bookmark is sorted into a top folder called "Others". CaterBook supports in the current version only categorization of English written Web pages. Web pages written in another language are categorized in the folder "Others". Note, that the default preset bookmarks of Netscape Navigator and Internet Explorer are skipped during the import of a bookmark file. They are usually considered as useless.

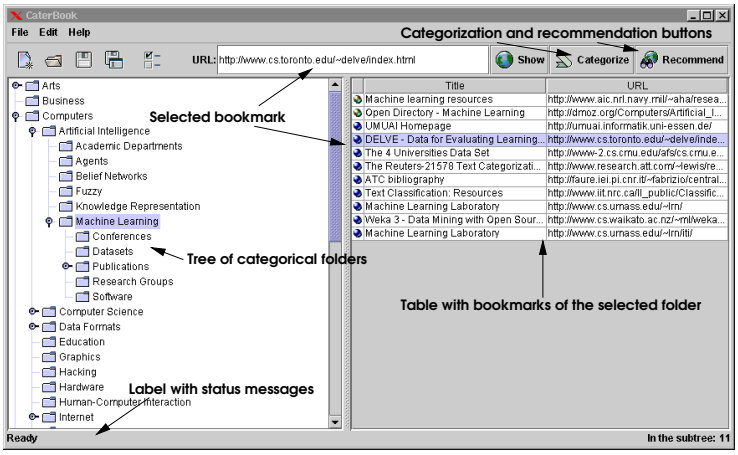

Figure A.2: Screenshot of CaterBook

Working with the CaterBook

The application frame is splitted into two panels, on the left is the hierarchy tree of categories and on the right are the bookmark records of the currently selected category (see figure A).

The bullet of each bookmark record shows if a bookmark is categorized precisely in that folder (a green bullet) or if it is categorized in a subfolder (a blue bullet). The displaying of bookmarks even in the upper categories than they are really categorized facilitates the searching. You can affect the maximum and minimum number of bookmarks in a category by setting the values in the Tree tab pane of the preferences dialog. After the maximum of bookmarks in a category folder is reached, the application will offer splitting of the folder into subfolders and resorting of the contained bookmarks. The bookmarks can be manipulated in several ways:

- show a bookmark in the browser - press the right mouse button and choose the Open in a browser item from the pop-up menu. Or select the bookmark and click on the Open button in the toolbar.
- categorize a new Web page - type the new URL into the text field in the toolbar and click on the Categorize button.
- get recommendation for related Web pages to a bookmark - select a bookmark and click on the button “Recommend” at the right top corner. In the table you will get in few seconds a list of related Web pages, displayed with a red bullet (see figure A.3).
- move a bookmark - press the left mouse button, drag the bookmark record from the table and drop it to another folder
- copy a bookmark - press the left mouse button and Control key, drag the bookmark record from the table and drop it to another folder
- delete a bookmark - press the right mouse button and choose the Delete item from the pop-up menu.

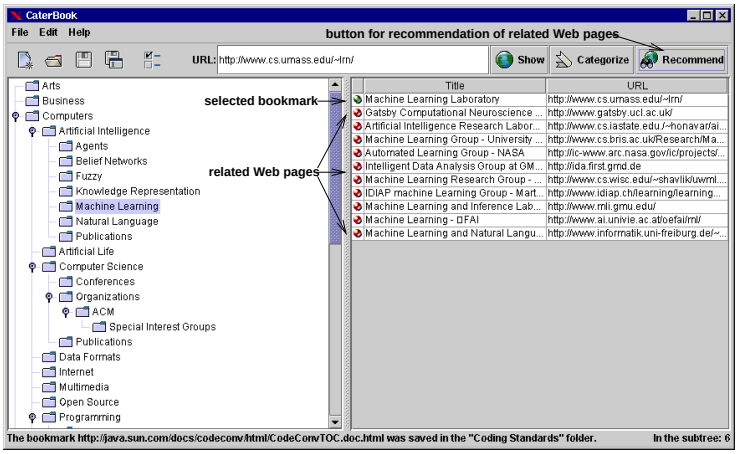

Figure A.3: Recommendation of related Web pages

- rename a bookmark - press the right mouse button and choose the Rename item from the pop-up menu.

Appendix B

Paper

The following paper called “A System for Automated Bookmark Management” was submitted to the 14th Belgian-Dutch Conference on Artificial Intelligence (BNAIC’02).

Appendix C

Enclosed CD-ROM

The CD-ROM enclosed to the thesis contains the following:

- The installation package of the prototype system CaterBook.
- The source code of the system including the Java documentation.
- The thesis text in the PDF and HTML format.
- The conference paper presented in Appendix B in the PDF format.
- The conference and journal papers used in the bibliography in the PDF format.

When using the CD-ROM, start a browser with the file *index.html* placed in the root directory, which contains the interface to the content.