Multilabel Learning for personalized Bookmark Classification

Bachelor-Thesis von Bastian Christoph aus Fritzlar
April 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

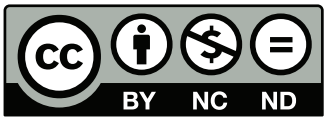
Fachbereich Informatik
Fachgebiet Knowledge Engineering

Multilabel Learning for personalized Bookmark Classification

Vorgelegte Bachelor-Thesis von Bastian Christoph aus Fritzlar

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Eneldo Loza Mencia

Tag der Einreichung:



Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 20. April 2014

(B. Christoph)

Zusammenfassung

In dieser Ausarbeitung werden wir die Anwendung von Support-Vector-Machines für Multilabel hierarchical datasets zeigen, die über den allgemein bekannten Bookmark-Klassifikation-Prozess angelegt wurden. Dazu werden wir zwei Ansätze für die Multilabel-Klassifikation vorschlagen.
Abschließend werden wir zeigen, dass die vorhergesagten Multilabel-Rankings für verschiedene levels von der Hierarchy evaluiert werden können.

Abstract

In this thesis we will show the application of support-vector-machines to multilabel hierarchical datasets, built in the commonly known process of bookmark classification. For this purpose we will propose two approaches for the multilabel classification.
Finally we will show that the predicted multilabel rankings can be evaluated for the different levels of the hierarchy.

Contents

1 Introduction

Over multiple years, a large size of web-portals developed, providing their content to different groups of users.

As each content-provider usually focuses on specialized fields of knowledge, they developed schemes of resource-classification in terms of content-management-systems, underlying the portal to manage the delivery of new published items, which are often inspired by the traditional news-publishing-schemes known from the field of journalism.

As users usually read across the internet over multiple sources, they develop over browsing- and reading-sessions an own view of which content-providers share the most relevant content, such that users revisit the portals again when looking for new up-to-date content.

When managing their known and read resources, users apply the technique of storing shortcut-descriptors of the resources as bookmarks to their bookmark-knowledge-library, organized usually flat but partially hierarchical in bookmark folders, building an own library-catalogue-scheme.

Over sessions, it can be observed, that users build their own organization and classification scheme for their shortcut-library, mostly build out of the processed content. It is also partially influenced by the content-management-metadata, explicitly shared by the content providers. This can be distinguished headlines, breadcrumbs or expanded url-paths including natural language words, used in direction for a good orientation in content and a good usability-experience, such that users can easily point to that over techniques like bookmarks and have an orientation to come back on information needs and search over widely known web-search-interfaces, using the remembered focused vocabulary.

In this thesis, we will show the application of approaches from the field of machine learning, to support the user with a multi-label ranking, based on the user-defined methodology-scheme of organizing bookmarks into a hierachical folder-class structure.

First we will show how the process of bookmark classification during tabbed-browsing on user- interaction can be modelled and where we can take advantage of our classification approach to support the user.

There we will describe, how webresources from different content-providers can be represented in a library-catalogue over indexing.
In that phase of preprocessing, the basic elements are registered to later build vectors of features for the training of a classifier.
For that we will take a short look at the organization-scheme of content-providers, named theme-pages or dossiers, where items are grouped by an identified theme-category under a specific name to cover in abstract the content-topics of the item-subsets.

Second we will discuss the training of a linear support vector machine classifier to build a basis for binary-relevance (br) predictions of items coming additionally in the testing / prediction step.
There we build for each folder-class a br classifier, which classifies in a one-against-all fashion the class-instances against the set of instances over all of the remaining folder-classes.

In the third section we will use the results from the set of br-classifiers to calculate for each new incoming item-instance a personalized (personal library based) ranking of the best suitable classes for

the instance, where we want to support a user in assigning one new instance to more than just one class (multi-label), since often users find themselves in the situation, that content-items cover from their own point of personal qualification, evolved chronological knowledge and target interest on information needs, multiple subtopics, that will be found in multiple folders, building an intersection of different fields of knowledge, that can later be a basis, to split or merge/cluster the folders, leading to a more distinguished classification-scheme.
There, users are mapping topics of interest over item-sets, best to their recent professional and chronological knowledge, to a hierarchically organized folder-structure with an assigned focused, controlled vocabulary behind it.
As goal of distinguishing the personal classification-scheme over sessions, it can be observed that users reach a better focused vocabulary, leading to more precise navigation in the web, on interactions with search-interfaces, as feedback results do better cover the goal of information seeking, the more precise the typed keywords were selected.

In Chapter 3 we show our multilabel datasets and which issues should be considered in contrast to the widely known social bookmark classification techniques.

In Chapter 4 we show the important aspect of focusing towards the article-content directly and how this is used to build our indexing of the datasets in the database.

In Chapter 5 we formulate the connection to the field of machine learning, how the imported datasets can be decomposed for the learning of classifiers and how classifiers are used for multilabel classification and label ranking of the predicted results.

In Chapter 6 the experiments running on the two datasets are proposed, especially how flat single-level bookmarks and hierarchical multi-level bookmarks can be covered using our approach. There we will basically address the class imbalance for a binary classification task.

In Chapter 7 we will show the general setup for the evaluation and the configurations for the experiments.

In Chapter 8 we evaluate the results of our experiments.

In Chapter 9 we finally summarize with our conclusions and show some suggestions for further work.

2 Bookmark Classification and the System as Knowledge-Library

When users are exploring the web over browsing tasks and browsing sessions, they get experience with the websites to judge about the content and functionality. As some of the explored websites often fit the information needs of the users, they are revisiting these websites.

Service Library Catalogue

The websites can be treated as services with specific functionality. All the websites, that one user is exploring regularly, can be treated as his '**Service Library Catalogue**'.

Content Library Catalogue

When a user is opening an URL in the browser, he retrieves mostly an HTML-document as answer for a GET-request. This can be treated as the content-output of the invocation of the URL as service-function. To build an index over these content, we need to apply wrappers, that select the needed parts of the HTML-document and store these as reduced, distinguished representation in the database.

The distinguished datasets in the database are building the '**Content Library Catalogue**' and are needed to define the features for the training of algorithms to calculate automatic recommendations for building a personalized classification scheme.

This type of focusing on the content-reading is known from browsing tools like Readability[1], the Safari-Reading-Mode[2] or the iReader-Plugin[3] for Firefox and Chrome. These are usually invoking the function over a Bookmarklet.

Reference Library Catalogue

As third point, the user is usually reminding important or useful HTML-documents over storing the URL together with some parts of metadata-text as shortcut-descriptor (**Bookmark**) and organizes these references in a hierarchical tree structure, as bookmarks are assigned to bookmark-folders and bookmark-folders can have other bookmark-folders as parent.

This tree-structure does not contain the HTML-documents or content inside the documents itself and so we will call it the '**Reference Library**'.

Since one bookmark can be added multiple times to the tree for different folders, we can consider the whole bookmarks-tree as multilabel dataset, where the descriptor of a folder-node is called '**Label**' and the sparse feature vector generated out of the content-output is called '**Instance**'.

[1] http://www.readability.com/
[2] http://support.apple.com/kb/PH5068?viewlocale=en_US
[3] http://samabox.com/extensions/iReader/

In terms of browsing tools, currently read-later and aggregation tools like Instapaper[4], Pocket (Read It Later)[5], Readability[6], Evernote Clearly[7], Storify[8], Pinboard[9] are available.

Personal read-only View of the Web, Labeling Vocabulary, Focused Vocabulary

The user is usually storing only shortcuts to web-pages, where the HTML-document itself behind the invocation of the URL is almost static, since the idea of bookmark classification is to map the process of finding and reminding [1] large sets of web pages into the bookmark organization tool. This has also been considered in [2], following the concept of associative linking [3].
So we want to assume, that the user is building a personal view of the web [4] over browsing sessions and encodes his knowledge partially over organizing the shortcut-references in the bookmarks-tree.
In the bookmark-folder-descriptors the user is defining personal words (labeling words) that should be a super-concept in relation to the descriptor-words of the contained bookmark-set. There the focused vocabulary, that is selected by the user to make the assignment, usually not formulated explicitly.
In that case, it can be assumed, that these modeling of knowledge organization does apply to a read-only type of the web. The only type of writing is when the user types special values as invocation-parameters of services on HTTP-GET, which can be stored to logfiles on the server side.

As the user is naming the bookmark-folders based on his knowledge about the bookmarked items and properties shared across sets of bookmarks, the used words for naming can be treated as labeling vocabulary. The knowledge here is about selecting a focused vocabulary out of the webresource words and find words from the folder-labeling-vocabulary to make the mapping.

If two bookmark-folders share a common set of bookmarks it can be considered, that the bookmark-folder-concepts are related / associated with each other in terms of co-occurence. This has been examined in the Associative Read-Out Model (AROM) in [5], using an associative layer [6].
The associative relation between words has early been discussed in [7].

It can be observed as a goal, that the user is learning (building) a structured scheme in type of an ontology, related to a library catalogue [8] of a digital library [9].

[4] http://www.instapaper.com/
[5] http://getpocket.com/
[6] http://www.readability.com/
[7] http://evernote.com/intl/de/
[8] http://storify.com/
[9] http://pinboard.in/

2.1 The TODO-List: Tab-based Browsing in Web Navigation

Each tab in the browser can be treated as an entry on the TODO-list of the user in a web navigation session. With a growing amount of opened tabs, the user has a growing TODO-list. For a growing TODO-list, the user needs to define priorities for the entries, where the number of priorities can be defined with a range, such that multiple entries can have the same priority. Entries with an identical priority are defining an entry-group. Each group of tabs in tabbed browsing can be moved to a separate browser-window.
In organization of the TODO-list the user can assign self-defined concepts / labels to the priorities in order to remind the groups over natural-language words instead of the priority-numbers. Additionally the natural-language words are usually formulated related to the meta-data of the tabs in the tab-group, that the user can see in organisation, without going into the document-content itself.
This gives us a first labeling-intention for the later formulation of bookmark-folders for a group of bookmarks.

2.2 Lifecycle: Different Phases of a Resource in the Knowledge Library

When the user draws resources out of the web, usually different steps are done in order to do a bookmark classification process. Especially in browser-interaction links are first opened in new tabs, later added as bookmark and get further classified in hierarchical organization. Here we want to show the lifecycle first to give a basic understanding for our modelling in this thesis.
After that we will show the browser-interaction process to point out at which step we want to take advantage for our approach to support the user there.

1. Discoverable:

   Initially the article is not published and not available on the web.

   a) Adressable:

      In the publishing phase, the each page of the article is assigned as HTML document to an URL. Therefore the article gets adressable over the URL or set of URLs in type of a '**Web Resource**'. In Section 4.1 we will further describe how a multi-document article can be handled.

   b) Accessible:

      When the article was assigned to an URL, it is still only reachable for those, who know the URL. Therefore it needs to be integrated into the website navigation structure over placing links to the URL on different documents that are still accessible. With that the web resource is accessible over a navigation path. In the case of search engines, a crawler can retrieve the resource over a navigation path now and adds it then with the document-text-unit (token)-representation to an index. Therefore the resource gets accessible over the resultset of the invocation of a search query, related to the indexed documents.

2. Observed:

   When the user navigates on browsing through the websites or applies search queries, he can discover resources along the path. When the user now decides to open the dicovered resource in a new browser tab, we define, the resource was actively '**observed**'.
   Here, the user read only the metadata (url, page-title, headline, date) of the resource without further investigation.

3. Structurable:

   During browsing sessions users often want to remind some of the observed resources when they are clearly 'important' or maybe 'needed' with respect to a specific context. Therefore users can

simply add them as bookmarks to the root of the bookmarks-tree or there needs to exist a specific branch (structure) on a growing tree, such that the user can easily remind, to search on which path of the tree to reach these reminded bookmarks.

For that structure we can identify two prototypes. At first the structure is formulated before the bookmark is added. In the second case some bookmarks are collected under the root node and when the user has a specific need for organization, a branch is created. We will later discuss these prototypes more in detail.

4. Collected:

   When the user decides to add the observed article at least to the root node of the bookmarks-tree the article gets '**collected**'.
   This is comparable to the user sending an email with links to himself to collect it in the `'TO_READ'`-named email-folder.
   Here the user decided to include the article to his library without explaination. The content of the article may get important, when a larger set of items has been collected.

5. Organized:

   With a growing set of collected bookmarks the user gets a need to further distinguish between subsets inside a growing collection. Therefore the subsets are assigned in organization to a prepared structure-branch. Now the user can easier remind the distinguished subsets over the tree-structure. Here, additionally to the metadata, the user was reading parts of the content of the article. This can be summary / first paragraph, some paragraphs starting at the beginning or the first page of a multi-page article. With that action, the article was further classified through adding additional context information (folder description in the browser profile). We define the article was set into a personal context by the user. Here the user needs to read at least parts of the content to judge about which folder should be selected. When no specified folder can be determined, the root node of the current browser profile (e.g. 'Home', 'Work', 'University', 'Development') is used as basic context. We will later discuss an approach, how the user can be supported in finding a suitable folder in his environment, based on his personally built tree.

2.3 Bookmark-Classification Process

In this section we want to show the general process of bookmark classification to give a basic understanding at which steps we can take advantage of our approach to support the user in the process.

At first we will show the steps in the process.

1. Define a browsing-task as browsing-context:

   Usually the user finds himself in a specific context as starting point, why a browsing session should be done. This context can be e.g. location, project, task group, task.

   Here we can especially observe, that a single person is usually having a user-account on multiple client-computers (e.g.: at home, at work, mobile-devices), which means that the user-profile can be distributed over multiple clients and one machine-user can have multiple web browser profiles, each assignable (usable) for a special context.

   As the user can be in different and changing working contexts when building the bookmarks-tree we can assume, that he is building it in a '**self-collaborative**' way.

   For a general location-context, a browsing-task can be formulated like 'Which articles are new for my topics of interest at my sources of interest?'

   This gives the constraints for the initial task-context:

   - retrieve articles as HTML documents
   - the articles are most recent to the current date
   - the topics of interest are mostly formulated abstract and are to be specified more precisely during browsing over formulating words that the user knows from his general labeling-vocabulary
   - the sources of interest are personally selected / acceptable / trusted websites

   When working on a project and solving groups of tasks or a single task the formulation what to be searched is usually more specific.

2. Start the browsing-session:

   Start the browser and open the first window over selecting a web browser profile.

3. Observe: Open resources, that seem to be relevant for the problem-solving of the browsing-task:

   Navigate through the known websites and open pages in new tabs, that contain words, relevant to the topics of interest.

   Here it can often be observed, that as current websites have an underlying content management system, approaches are used to recommend also pages to the user, that are in any type similar with or related to the opened one, relevant for the task.

   There users often open pages from these clustered recommendation, that do not match the browsing task directly and are relevant to these, but are related to that task over the organization scheme of the content management system.

An example for this are side-recommendations like 'new in the ressort / category', 'most read', 'most commented' or 'users who read this, also read ...'.

This effect, also known from browsing (walking) through a library, where users find books not by direct search, but by the order of the bibliographic organization scheme in position next to the searched one, is often called '**Serendipitious Information Encountering**' [10].

In this case it needs to be considered, that the user is in fact unconsciously extending his formulation of the browsing task, when opening pages, that where initially not thought to be opened and not matchable over the initial formulation.

Here a type of noise is introduced, compared with the initial task. This often leads the user into losing increasingly the overview over the several adressed topics [11] across the growing collection of pages when switching often between them. It is often considered as '**Information Overload**' [12] [13] and sometimes as '**Organization Underload**' [14] as the user loses increasingly the ability to distinguish between the contained subsets caused by their dissimilarity and missing own organization structures and adapts unconsciously to the organization scheme of the website [15] without explicitly accepting this by reformulating the personal scheme.

For the case, that the user wants to build a personal organization scheme that is applicable across multiple websites and not only the current one, the user needs to get aware of his personal labeling-vocabulary for an abstract formulation of his topics in order to learn a strategy for a focused navigation.

As approach to solve this, the user needs to explicitly formulate an organizational structure of groups of pages, where each group is assigned a focused vocabulary to get the ability back to formulate explicitly central assumptions about the topics, covered by each group of pages, underlying the growing collection.

With the increasing availability of content with the growing web, it becomes a need for the users to develop a personal formulation, which recommendations of the available websites should be included and which ones are not relevant, related to that personal organizational scheme.

This should give a method for a more structured, focused navigation with a continously growing web, considered in abstraction as a type of digital library. Thereby a personal catalogue should help to do the classification and filtering.

It can be thought to include the personal scheme for reducing the views of the websites to pages, that are matchable with the vocabulary behind the personal scheme, reducing negative effects of serendipitious information encountering and leading to a more focused navigation in the web.

In terms of web crawling, the personal catalogue can also be used in personalization as reference for a schema-based focused crawler [16].

4. Pre-collect: Group the opened resources over tabbed browsing workspace

   Here the goal can be observed to formulate groups of tabs to find suitable folders for sets of bookmarks in a session of parallel browsing [17].

a) Build groups of resources (implicit group vocabulary):

Create an additional window, that should represent the group. The window is initially assigned the known, defined browsing-task. As the group-window should cover a subset of the resources, we can formulate additional words as constraints in order to reach the reduction.

Now we move resources from the first window to the new one, that contains words for a shared specific topic. These words are first assumed to be added as constraints to the browsing-task-formulation and can be formulated temporally explicit in the window head.

This should give the user a strategy to build a frame for finding a formulation for the unconscious knowledge, applied for deciding to open the pages in the session of parallel browsing. There, the user wants to formulate a label-concept, such that all label-instances can be treated as grouped by these label-concept.

b) Identify the group-vocabulary temporally explicit:

Switch to the new window, that contains the extracted group of resources. Here we find now a reduced set of pages, compared to the initial task, as specific words were selected as additional task constraints in order to identify the shared topic. Read the resources and identify from each resource, which words should be assigned to the group temporally.

This can easily be done with a Tabbed-Browsing-Plugin like 'Tab Mix Plus'[10], where words can be added to the head of the Tab-Group-Window in the workspace view.

The words added to the tab-group-window are basically needed to distinguish the groups in the overall workspace-view. After the resources are added later as bookmarks to the folders, these words can be discarded since the folder-label-words can be reminded.

5. Collect and organize the grouped resources

a) Structure: Create a bookmark-folder:

Create an empty folder-container and an empty folder-descriptor. Add the identified words from the group-vocabulary or related labeling-words to the descriptor.

b) Classify: Collect and organize the bookmarks:

Add all resources from the group as bookmark-shortcuts to the folder-container. This means assigning each bookmark to the bookmark-folder.

Thereby the bookmark is the instance and the bookmark-folder is the instance-class, when speaking with the terms from machine learning. The user is here learning (training) a '**instance-of**'- relation (assignment), that should later be included as training data for the classifier algorithm.
In Chapter 5 we will further explain our modelling for multilabel-learning.

In the recent integrated browser-tools for assigning a bookmark to a bookmark-folder, e.g. the firefox bookmarks, a ranking is presented to the user, that is based on which folder was used by the user most recently and therefore sorted related to that. We define the ranking is for that case sorted by the attribute 'recently used'.

[10] https://addons.mozilla.org/de/firefox/addon/tab-mix-plus/

In this thesis we want to show a personalized approach, how a ranking can be generated, that takes the webresource-content into account, which is delivered with the bookmark-HTML-document-sequence. Now an algorithm is trained with the user-bookmarks-tree based on the content and for a new bookmark as test-instance a ranking should be generated to support the user in the instance-assignment step. Additionally the ranking should recommend the first few folders as preselected to deliver a possible personalized multi-label assignment.

When the user is running browsing-sessions over time, it can be observed, that a resource, used for solving a previous browsing-task, can occur again for solving a different task.

In that case, a bookmark is assigned to multiple folders and can be considered as shared across multiple classes and shared across multiple tasks. In terms of machine learning, this step generates the bookmarks as '**Multi-Label Dataset**'.

Hierarchical relation:
When two bookmark-folders have a sufficiently large common set of bookmarks, the user can decide to move that subset to a common parent. Therefore a new bookmark-folder is created and with it the empty folder-container and folder-descriptor. Then the common subset is moved from the child- containers to the parent-container. The common words from the child-descriptors are moved in reformulation to the parent-descriptor. Finally the new bookmark-folder is set as new parent to the child bookmark-folders and the previous parent of the child-folders is set as parent to the new folder, introducing a hierarchical relation for the tree. This hierarchical subconcept-to-superconcept relation was considered in [18].
Additionally this type of semantic relation was discussed as extension possibility for connectionist models [19], [7], [20].

2.4 Bookmarks-Tree

With the formulation of the process of bookmark classification, we have a structure for the organization of references towards HTML-documents. As initially proposed in Section 2, we are considering the bookmarks-tree as '**Reference Library**', since these HTML-documents are the result, when invoking the service when opening the URL, which is the representation of the service-function together with its parameters for HTTP-GET. To give a basic unterstanding about how this reference library can be build, we want to show two prototypes of different organization strategies of the hierarchical structure, related to the lifecycle of the item.

Structure-Case: Early Hierarchy Formulation (Top Down)

When the user adds bookmarks, at first a suiting branch in the hierarchy must exist to assign bookmarks there. If the branch does not exist, it needs to be build first. The created branches are usually structured related to the users professional experience with organizing documents in the general personal 'Offline' information management processes. This was especially examined in works about the psychological issues [21] and the management of emails [12]. Additionally they are inspired by often found organization schemes when browsing the web, for example ressort-categories or theme-pages from newspaper-websites or sitemaps in general. There the user does not know the underlying assumptions (rules) for the structure of the hierarchy-branches, but judges in imitation to agree with the structure and copy it at least partially to the own knowledge-organization-hierarchy, extending his labeling-vocabulary.

With that, we can identify the sequence of operations from the lifecycle:

1. Structure:
   Build a suiting branch in the tree
2. Collect:
   Include the observed resource as bookmark to the reference library
3. Organize:
   Assign the collected bookmark to the bookmark-folder in the build branch

Structure-Case: Late Hierarchy Formulation (Bottom Up)

Here the user does initially not set hierarchy-rules and collects the bookmarks directly under the root-node in a type of '**pile**' [22], keeping in mind how to distinguish between different sets in the growing collection. This is especially observed, when users only have a few bookmarks with no need for a more complex hierarchical organization. With a growing collection, the user needs to keep in mind increasing numbers of subsets to distinguish between items and remind a sort order. For an increasing number of subsets, the user needs to formulate labeling-words explicitly, that are commonly shared across all items inside a subset of the collection. When each item is now assigned to a parent-folder in type of a 'prefix' using the formulated words, the user gets back a sort order over the collection, such that the different subsets can be distinguished easily. A hierarchy over multiple levels is built, when the prefixes are again prefixed with additional words. The hierarchy is usually visualized using a tree structure.

With that, we can identify the sequence of operations from the lifecycle:

1. Collect:
   Include the observed resource as bookmark to the reference library. It is assumed that it is assigned to the root-node.

2. Structure:
   If there are too much subsets in the collection under the root-node, identify one subset, define a common attribute for that and build a folder (branch-structure) using that attribute.

3. Organize:
   Now the items from the identified subset have to be assigned there (moved from the root-node towards these new node). If the newly collected bookmark is not contained inside these item- set, it stays assigned under the root node.

Here we can observe, that this strategy is more directed to exclude easy formalizable subsets from the root-node-collection than organizing the new bookmark directly to a folder in order to build a sorted tree-organization-structure. This gives us an explicit formulation of prior knowledge about growing collections, just build out of the contained data and less oriented toward generally experienced variants of possible hierachical organization. With respect to email-programs we would state here, that items are collected in the root-inbox until the user is able to identify topics, that can be moved to a common folder, labeled with word-sets of each topic.

3 Multilabel Datasets

3.1 Newspaper: Chronologic Theme-Pages and Dossiers from the Meta-Catalogue

In the web, the availablility of user-generated hierachical bookmarks is usually quite low. Mostly, users share their bookmarks on folksonomy-like social bookmark classification services like delicious[1], which gives a flat one-level hierarchy without further explicit relations between tags.

There, conceptually users assign possibly multiple '**tags**' to a bookmark. When multiple users assign different sets of tags to one resource in collaboration, only the resulting overall set of tags is shown. There the context-information about the used subset of tags is lost.
In the definition of the tags it can usually be observed that a user is free to attach any word as tag to a bookmark and one tag is used by different users with varying background knowledge about a word, such that this polysemous usage of words leads to a type of noise [23].

To solve this, we will use an own created dataset, that is basically not build in a socially collaborative but more self-collaborative way as we described in Section 2.2. In our case, we asssume that one prototype user was building the dataset over multiple browsing sessions and possibly many client-machines. For the bootstrapping, we select so called '**Theme**'-pages and '**Dossier**'-pages, that will be imported by a crawler to some depth starting at the current date. Since one resource at a news portal can be assigned by the journalists in organization to multiple themes, this gives us a multi-label dataset.

In contrast to the concept of tagging we first allow, that one bookmark-folder-descriptor can have multiple words. Second we allow that the user can formulate a hierarchy of folders, such that explicit parent-to-child relations can be defined.

As the user is usually reading across multiple news sites, we will pick possibly themes with the same name, available at different sites. In this point we abstract from the fact, that one special theme can be handled quite different, related to the journalistic main orientation of the editor-in-chief strategy behind the site. There we ignore these possibly different fine-grained interpretation of themes and topics and treat the overall merged theme as one bookmark folder.

These themes and dossiers are usually available in the meta-catalogue sections of the portals, which gives the user easier and extended deeper access into the archive for a potentially increasing revisitation. The meta-catalogue feature can be found in news sites as Tagesschau Online[2][3] , Spiegel Online[4], FAZ Online[5], ZEIT Online[6], Golem Online[7], Heise Online[8], Computerwoche Online[9] .

We can observe that similar to this meta-catalogue feature used in this newspaper websites, the bookmarks-tree is also based on the descriptive metadata of the web documents. In contrast the bookmarks-tree has an explicitly formulated hierarchical ordering relation for the folders, instead of an alphabethical ordering relation in the newspaper websites.

1 https://delicious.com/
2 http://meta.tagesschau.de/
3 http://www.tagesschau.de/archiv/dossiers/
4 http://www.spiegel.de/thema/
5 http://www.faz.net/themen/
6 http://www.zeit.de/schlagworte/themen/
7 http://www.golem.de/specials/
8 http://www.heise.de/thema/
9 http://www.computerwoche.de/schwerpunkt/

We can observe, that the interpretation of special theme-names and topic-names changes dependent on at which chronological point in time it is used.
This is often called '**Concept Drift**' [24]. According to the incremental formulation of the bookmark-folder-descriptor we can observe, that a user introduces or reuses a word or set of words for each set of added bookmarks. Chronologically this sets give each a different meaning of the concept, such that subsets inside the folder define the drift in the concept.

For further investigation on concept drift it may get important to handle that overall single-label property inside the one folder as a multi-label property, as one single word can be used in multiple sets of words that are usually finally merged.

The concept drift is usually driven by the events in the calendar, where journalists have to report about the current facts, happening in context with the theme at the selected calendar-dates. This is often used in market-driven journalism [25] and online news production [26].

3.2 Reuters RCV1 V2

For our experiments we will use the benchmark dataset Reuters-RCV1-V2 [27] in its form available on the website[10].

This dataset was commonly used in lots of experiments inside the Information Retrieval (IR) and Natural Language Processing (NLP) Community, such that it is a good reference dataset to run our experiments for multilabel text classification.

[10] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

4 Data Preprocessing, Indexing

4.1 Wrappers for Service-Invocation-Outputs

In recent observations about the structure of websites, it can be observed, that developers are increasingly following guidelines of good-case-practices to reach a better maintenance of the pages inside the website and to make it more machine-readable for search-engine-optimization.

Content Management Systems, Abstract Template

It can be observed that with the increasing availability of open-source content-mangement-systems[1] such as Joomla, Drupal, TYPO3 and WordPress users can easily create personal websites or blogs. In general, developers are building some examples for a central, unified layout-template shared across all pages inside a website-section. This will then be used to present all pages of the section in that layout to the user, just filling in the template with the content, stored in the database-catalogue-tables. In many cases it can be observed, that developers do not always build the layout-templates themselves, but do more rely on the available frameworks for the management of content, increasingly available on the web, reducing the effort for an own distinguished webdesign.

Multi-Document Sequence, Central Content Filtering, Wrapper

When article-documents that are spanning over multiple pages are presented in the web, often one separate HTML-document with an own URL for each document-page is created. Therefore using the first URL only it can not be determined, if more HTML-documents exist in a sequence. If one user opens the URL and navigates through the document, a 'next'-link can sometimes be found that references to the next document-page. Since each content-provider can encode that link in a different way, we have to build a special module called '**Wrapper**'.

The wrapper should first implement a technique to read all URLs in the multi-document sequence. Second, each HTML-document is usually build out of multiple conceptionally different parts [15], such as website-navigation, context information like breadcrumb, recommendations, central content and more. In this thesis we want to show an approach to train a classifier-algorithm based on the plain central content inside the sequence of HTML-documents. Here we can especially observe that if we would use all words of the document sequence, then words from e.g. the navigation structure would be counted many times if there are multiple documents in the sequence, resulting in misleading high word counters for classifier-training. Therefore we have to implement a filter-approach [28] in the wrapper, that selects the special content-part out of the overall document.

Since we focus in this thesis on a manually essembled dataset, we do not implement wrappers for all websites in the web. In the case of adding a bookmark for a new website, a prototype wrapper can take just the first document in the sequence and read all words from it. There we expect a type of training noise, since if the user adds some documents of that website, the classifier would possibly get trained on the often reoccurring words of the navigation structure.

Finally the resulting unified representation of the document generated by the wrapper should be registered (stored) to the database. We want to call that unified representation '**Web Resource**'.

[1] http://de.wikipedia.org/wiki/Content-Management-System

When looking at the DOM-representation of the pages, we can identify properties about the underlying templates over looking at the (key, value) - pairs, delivered as attributes with the DOM-nodes.

There we can make use of three types of (key, value) - pairs:

- '**class**' attribute
  Developers often make use of the '**class**'-attribute to outsource reoccurring CSS- layout and design information (**style**) to a separate CSS file and then encode the style over a special selected and partially semantic meaningful name, put as value to the attribute to make the mapping. When using this attribute, developers have the opportunity to format multiple areas of one page in the same style and give identifying names to simplify the lookup.

- '**id**' attribute
  Some areas inside the template should be uniquely identifiable, such that developers make use of the '**id**'-attribute, which can also be connected with a named CSS-style from an external file. With the increasing usage of that attribute, it becomes easier to select special elements (DOM-nodes) out of the DOM-tree representation. As these elements have an unique identifier, we can make a mapping of the element-content over the '**getElementById**'-method, widely supported in todays DOM-reading-frameworks, to an element-variable in our abstract page-instance-representation.

- **HTML Microdata**
  With the upcoming **HTML Microdata** specification of the W3C[2] and WHATWG[3], it can be observed as a third point, that there is a standardization about attaching special names to the DOM-nodes as additional property-attributes, which are drawn from a specified general schema, used by the content provider to give a semantic enhancement, how the content inside the node should be interpreted.

  As an example, the german news website SpiegelOnline[4] adds for the article-area inside the news-item-page the attribute '**itemtype**' with the value Article[5] to refer to the used schema in the section. Then the usage of the attribute-values ('datePublished', 'headline', 'author', 'description') can be observed for the attribute-key '**itemprop**', which leads to the assumption, that the content found there should be mapped to the article-fields (datePublished, headline, author, description/summary) in the abstract representation as metadata, giving descriptive information about the article-instance.

As the formulation of node-microdata is analog to the formulation of class- and id-attribute, but more directed to attach an explicit formulation over an additional schema-document, we will treat the values of the two last named property-attributes as '**Implicit Microdata**'.

The technique of attaching a schema document is also currently known from the Semantic-Web Resource-Description-Framework (RDF). Initiatives such as the Dublin Core Metadata Initiative (DCMI)[6] are following the concept of building standardized vocabularies for such schemas.

[2] http://www.w3.org/TR/2011/WD-microdata-20110525/
[3] http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html
[4] http://www.spiegel.de
[5] http://schema.org/Article
[6] http://dublincore.org/

For the generation of a webresource-instance, given the first URL, we need to build a wrapper for each content-area (group of pages) for each website. This is analog to building a single wrapper or a group of wrappers for each content-management-system.

For the encoding of each wrapper we need to make a static mapping-formulation that determines, which microdata- values should be used to map the text of the DOM-node to the related field of the abstract representation of the webresource.

Here we show a short example, how the summary-field of an article of the german news-website ZeitOnline can be mapped (class='excerpt') using the Java-DOM-processing framework jsoup[7]:

```
/* (non-Javadoc)
 * @see wrappers.IWrapper#extractSummary()
 */
@Override
public void extractSummary() {
String ressourceSummary = this.doc.getElementsByClass("excerpt").get(0).text();

this.ressource.setSummary(ressourceSummary);
}
```

Applying these node-processing scheme, we are reading the following fields of the webresource-instance:

- url
- topic
- title
- date published
- description / summary / intro / teaser

As these fields do not contain the content-text itself, we treat it as webresource-descriptor (webresource-metadata) and register it in one database-table-row, for each webresource.

After storing the descriptor, we read next the content-text of the resource, again applying our scheme. Over the static encoded mappings, the wrapper reads the DOM-node, covering the encoded id-name, removes all parts, not related to the text directly and not used in our experiments-modeling (e.g. imagelinks, copyright, authorname, article-references), and then treats the remaining text as content-text-field. Since it is often observable, that content-text is organized in paragraphs, we exploit this inside-document organization-scheme and store the content-text as a list of content-text- paragraphs.

Finally on extraction, we write the webresource in XML-format with descriptor-fields and list of content-text-paragraphs to a file on the filesystem and attach the path of the file to the descriptor of the webresource-instance in the table.

[7] http://jsoup.org/

Now we tokenize the text, remove stopwords, apply stemming and register each token-stem to a token-table in the database as indexing-step, such that we can later build in a simple feature subset filtering process a sparse feature-vector-representation [29] of the webresource as training-input for our algorithm.

4.2 Reuters RCV1 V2

Multilabel Dataset

Here still all stopwords were removed and the tokens were stemmed. As the articles there are assigned to multiple hierarchy nodes, that have a common parent, we can consider this dataset to be a multi-label dataset, too. In terms of data preprocessing we can directly load the dataset into a database schema, identically to the schema which we use for our bookmarks representation.

As both datasets will be in an identical representation, we consider that the experiments, running on the datasets, will be comparable.

5 Multilabel Learning: Basic Modelling

5.1 Problem Formulation for Multilabel Classification

For our imported multi-label datasets, we need a problem formulation, such that the datasets can be handled to train a classifier algorithm.

First we define the elements of the instance-label relation:

- Label λ_{ID} := Label for the bookmark-folder (BMF) with the descriptor-ID
- Instance ib_{ID} := Sparse Feature-vector-representation of the bookmark with the descriptor-ID

As we described in Chapter 2 our bookmarks-datasets cover basically two aspects. First, since one bookmark can be used in different browsing-contexts it can be added to multiple different bookmark-folders, resulting in a multi-label dataset. Second, the user is free to create subfolders for each folder, resulting in a possible multi-level hierarchical tree structure.

These two types of link-associations are registered each to a different database-table:

- **Leaf-set**
  When bookmarks are associated with a folder, the resulting set of child-to-parent association-datasets should be called the '**Leaf**'-set. As we want to support multi-label associations, multiple datasets can have the same child-field (bookmark-ID) and different parent-fields (folder-ID). Since bookmarks cover the content of a folder, we define that the set of bookmarks for one folder is of the type 'Leaf'.
- **Inner-set**
  When bookmark-folders are associated with a parent bookmark-folder, the resulting set of child-to-parent association-datasets should be called the '**Inner**'-set. As we want to support hierarchical associations, a parent-folder can have a set of child-folders (subfolders), such that this builds a subtree-structure, rooted at the parent-folder. These associations between folders are building the structural 'Inner'-type sets of the subtrees.

For the structure of the tree we distinguish two cases:

- **Flat Bookmarks Tree**:
  If each BMF-node has the root-node as parent, this gives us a 1-level-hierarchy and we will call this tree structure '**Flat Bookmarks Tree**'.
- **Hierarchical Bookmarks Tree**:
  If one BMF-node has not the root-node as parent but a different BMF-node, then we have a multi-level-hierarchy and we will call this tree structure '**Hierarchical Bookmarks Tree**'.

When the user collected bookmarks under the root node and decided in organization to build separate folders for different subsets, we define that the bookmarks were moved from the leaf-set of the root to the leaf-set of the new folder-node in the inner-set of the root-node.

Therefore all leaf-sets of the subtree, rooted at the root-node, are building the multi-label dataset.

5.2 Considerations, Problem Solving Strategy

For our modelling we consider the work about multilabel ranking and hierarchical binary relevance as proposed in [30], [31].

Therefore we have to consider three aspects:

- Multilabel Dataset (Classification Tasks)
  We have to decompose each multilabel association into multiple single-label associations in order to build one binary classification task for each single-label association. This technique is widely known as '**Binary Relevance Decomposition**'.

- Hierarchical Dataset (Distribution of Task Instances)
  To train a classifier for a binary task, we have to find the set of instances relevant for the binary task. The instances are distributed over the folders of the hierarchy. We consider the instances of the subtrees, rooted at the parent of the instance-labels. The instances of each binary single-label classification task are covered by different subtrees of the overall tree, if the label-parents are different. Therefore, not all tree-instances have to be included for each binary classification task. When hierarchy-associations are not considered for a flat tree, we can define, that the parent of the instance-label is the root-node for each instance-label. Including the hierarchy, we have a more balanced distribution of instances. For one-vs-rest binary classification we have to select a parent instance-set to determine the 'rest'-set. Since we will consider each subtree of the hierarchy separately, this should be called '**Hierarchy Decomposition**'.

- Bipartition Prediction Strategy (Multi-Label Ranking of Task results)
  To determine a ranking over all labels, where the first part is assumed to be 'preselected', each label should be assigned to one of the two parts of the bipartition ('Relevant', 'Irrelevant').
  This is often denoted by $(Z_i, \overline{Z}_i)$ for a query-instance x_i. $(Y_i, \overline{Y}_i)$ denotes the reference-bipartition, which labels are initially classified {true, false}.
  The rank-position of a label $\lambda \in L$ is denoted by $r_i(\lambda)$. Therefore, first a probability-value as score needs to be computed for the rank-position.

  – Case: Flat Tree
    For a flat tree, the parent of each label is set to the root node, resulting in an overall 1-level flat tree. To train the classifiers, we first built an index for the subtree rooted at the root-node. We will show this technique for an aggregated subtree representation in Section 5.3. After the index is built, the SVM models are generated. Since each of these models cover as positive class an aggregated meta-label, which is for the flat case trivially identical to the leaf-type label, we can directly assign the label to one of the parts of the bipartition. In prediction of a new query instance all trained binary classifiers are asked. If the positive class is predicted with a probability higher than 0.5, the class-label is assigned to the bipartition-part '**Relevant**'. In the other case, it is assigned to the bipartition-part '**Irrelevant**'. The label ranking is finally sorted by the prediction probability of the positive class for each classifier.

  – Case: Hierarchical Tree
    For a hierarchical tree, the parents of two different labels can be set to different folders, such that we have to consider multiple subtrees in the general tree, resulting in an overall multi-level hierarchical tree. Analog to the flat case, we apply the same technique to build an index for a subtree and generate binary training models using the index. In contrast we have at first multiple subtrees to be considered in the ranking technique and secondly the meta-label, if the positive class is predicted, needs to be further resolved in order to decide which folder should be predicted as non-aggregated leaf-label. In this case we have to develop a more

advanced strategy to build the bipartition for the label ranking to determine the final ordering of all labels in the hierarchy. In Section 5.6 we will discuss this technique more in detail.

When training a model for a specific folder, the hierarchy for that folder needs to be aggregated (folded). Therefore, it can be considered that each bookmark has two labels assigned for its parent-folder:

- Label $\lambda_{ID,Aggregated}$:= Label of the aggregated general folder λ_{ID}. With respect to the HOMER-technique proposed in [32], this can be treated as '**meta-label**', representing the subtree.
- Label $\lambda_{ID,Leaf}$:= Label of the concrete folder-leaf-set

When the folder has only bookmarks in its leaf-set and no nodes in the inner-set, we can define:

$$\lambda_{ID,Aggregated} = \lambda_{ID,Leaf}$$

This will especially be considered later when ranking the labels, where we have to decide if the ranking should further be expanded.

We first notice, that for each leaf-set a binary base classifier should be build in a one-against-rest fashion. All bookmarks in the set are added in their feature-vector-representation as positive examples to the classifier model. Then we consider the overall bookmarks of the aggregated parent-label, exclude all actually used items and treat the resulting items in their sparse feature-vector-representation as negative examples for the classifier.

We will later see in concrete steps, how the training of these models will be done.

5.3 Hierarchy Decomposition, Binary Relevance Decomposition

In general we want to decompose the hierarchy of folders into subtrees in order to train multi-label classifiers for the folders in a sequential processing.

Looking forward to the used operations, we can identify that some operations will often be re-applied in the calculations with static, not changing results (e.g. number of child folders for one parent folder). This will be used to find the folders for that each a set of models should be trained.

Bookmark Folder Descriptors, Counters, Decomposition Decision Variables

We want to pre-calculate the values in an initial step and store it on the folder-descriptor. In order to prevent growing recursion-stacks for trees with a high depth, the values should be computed iteratively instead of recursively.

The following additional attributes will be calculated in initialization for each folder-descriptor:

- **leafItemCount**:
  Number of items, directly assigned to the folder as leaf-set-items. This will especially be needed to determine the Apriori-Probability of the folder-label. It will later be used on label-ranking, when we need to calculate the probabilities for labels in truncated subtrees.
- **innerItemCount**:
  Number of items, directly assigned to the folder as inner-set-items
- **innerLeafItemCount**:
  Sum of items, assigned to all inner-set folders in their leaf-set or inner-set
- **innerLeafBranches**:
  Sum of inner-set folders with: $leafItemCount + innerLeafItemCount > 0$
- **leafOnly**:
  If $(innerLeafBranches = 0)$, then this is set to 'true'
- **innerOnly**:
  If $(leafItemCount = 0 \wedge innerLeafBranches > 0)$, this is set to 'true'
- **counterPropDone**:
  In the initialization, the values (leafItemCount, innerLeafItemCount) should be propagated to the parent. After this is done, the variable is set to 'true'. Since we are iteratively computing the values, this propagation-step is important.
- **innerCountDone**:
  When the values of all children of the node are propagated to its parent, this is set to 'true'. To determine which folder-computation is finished, this variable is used.

Decomposition of the Hierarchy: MLC-Model Cases for the Subtrees

Finally we have to introduce the decision variable '**useMLC**' in order to determine, if a multilabel classifier as set of binary classifiers should be trained for the folder.
Therefore we have to distinguish four cases:

1. $leafItemCount > 0 \wedge innerLeafBranches = 0$
   Here we only have to consider one branch. Therefore we do not need to train a classifer and the variable is set to 'false'

2. $leafItemCount = 0 \wedge innerleafBranches = 1$
   Here we only have to consider one branch. Therefore we do not need to train a classifer and the variable is set to 'false'

3. $leafItemCount > 0 \wedge innerLeafBranches > 0$
   Here we have to consider at least two branches. Therefore we need to train a classifer and the variable is set to 'true'

4. $leafItemCount = 0 \wedge innerLeafBranches > 1$
   Here we have to consider at least two branches. Therefore we need to train a classifer and the variable is set to 'true'

Now we determined, that only for folders with $useMLC = true$ a multilabel classifier has to be trained for the subtrees rooted at these folders.

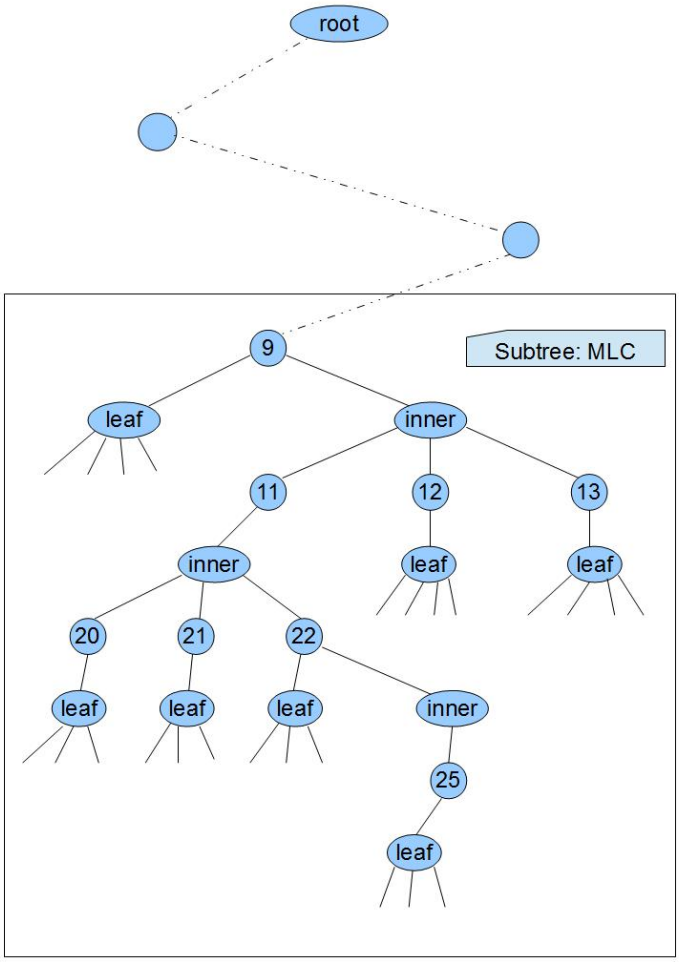


Figure 5.1: Subtree of Multilabel Classifier

As shown in Figure 5.1, we have now multiple subtrees in the hierarchy, each rooted at a trainable folder. For each of these subtrees, we will train a set of binary classifiers as multilabel-classifier.

We can identify that the depth of these subtrees can be quite high, such that the search time can be high and memory requirements might increase.

Therefore we want to build an index structure as flattened representation when exploring a subtree, represented by the relation (Folder-ID, ParentBR-ID, ParentMLC-ID, visited).
First we know, that we have to train a binary classifier for the leaf-set of the subtree-root, if it is not empty. There we just set the ParentBR-ID and ParentMLC-ID in the index equal to the Folder-ID. This means that the ParentBR-ID is treated as aggregated meta-label for the instances in the leaf-set. Second for each inner-set folder that contains leaf-items in the aggregated form, a binary model should be trained. To solve this, for each of that folders a model should be trained. In the leaf-form it is added with the equal ParentBR-ID to the index. All subfolders there are also assigned to that ParentBR-ID in the index as shown in Figure 5.2.

The index is basically build through Breadth-First-Search and the resulting datasets are stored in the index-relation as type of a 'Queue', such that the assigned bookmarks in the subtree-collection can later be easily collected in a linear sequential order, reducing memory requirements.

Since in this index structure the ParentBR-ID is generally treated as aggregated meta-label, we can use it independently of which tree structure is used.

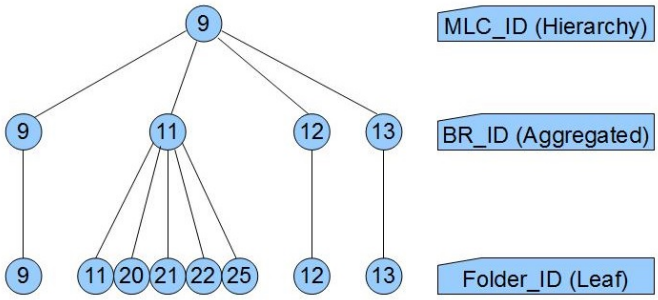


Figure 5.2: Index of the Subtree

5.4 Multilabel Classifier Training

Feature Subset Filtering, TF-IDF, Sparse Instance, Binary Base-Models using SVM

In general we have to identify the vocabulary that should be used in the training for each multilabel classifier. Therefore we calculate for each token in the hierarchy the normalized-document-frequency (DF) value with respect to the training documents. For the Reuters RCV1 V2 Dataset the {train, test}-split is used as proposed with the dataset in Section 3.2 In terms of feature subset filtering [28] we will take the tokens with the highest 5000 DF values for each classifier.

For each binary model we load all assigned instances using the index-structure and build sparse feature-vector-representations, reduced to the training vocabulary and using the TFIDF-value for each token. We can observe that this reduced vocabulary covers each instance by approx. 92%.

For the binary models we will use linear Support-Vector-Machines (SVM) [33] since it was investigated, that text classification tasks [34] [35] [36] are usually linear seperable. Therefore we do not need to apply more advanced kernel-transformation techniques. Additionally SVMs are well suited for high dimensional input spaces [29].

Generation of the binary SVM Models of the Subtree

When the index is built for a subtree rooted at a trainable folder, we can generate the SVM model files. These files should be named in the form 'MLC_<ParentMLC_ID>_BR_<ParentBR_ID>', encoding the index. Going through the index, all folder-bookmarks are collected and appended in their sparse feature-vector-representation to the model-file in ARFF-format[1] such that the set of subtree-models can now be used to calculate predictions and determine rankings. Here the ParentBR-ID is basically representing a meta-label. Since we have to decide later, if we have to further explore a subtree rooted at the folder with that ID, we will register a dataset with the model-file-name to a database-table, if the folder with that ParentBR-ID has no more subfolders. This database-table will simplify the lookup on exploration.

[1] http://www.cs.waikato.ac.nz/ml/weka/arff.html

5.5 Classifier Predictions, Bipartition, Label Ranking

When all binary SVM models down the hierarchy are built, we can calculate predictions for the models. As we initially proposed in Section 2.2 about the bookmark classification process, we want to calculate a personalized ranking of the folders for a new query-instance, where multiple folders are assumed to be preselected. Here we will give a short overview, how the ranking is basically built. In the next section we will go more into detail, how the trained hierarchy is explored on prediction of a query-instance.

In general we have to build a '**Bipartition**' (Relevant, Irrelevant) of the set of all available bookmark folder descriptors. In our modelling we have two types of binary models for one folder.
First, when the folder is predicted from its parent as binary model with more subfolders attached, this is the aggregated meta-label $\lambda_{ID,Aggregated}$, as we introduced in Section 5.1. Since we only want to rank leaf-labels, here only the subtree rooted at that folder was predicted and needs to be further resolved.
Second, when the folder is predicted from its parent as binary model with no more subfolders attached, the aggregated meta-label is identical to the leaf-label $\lambda_{ID,Leaf}$ and can be added to one of the sets of the bipartition. The probability of the prediction of the leaf-label, relative to the parent will be called 'positive class prediction probability' (**PCPP**).
Third we have to determine a sort-order for the items inside the sets of the bipartition. Therefore down the path from the root-node to the leaf-label, the prediction-probabilities will be multiplied, building the overall probability for the path of each leaf-label.

In our implementation of binary models we enable the SVM option that the prediction of the positive class is returned as prediction probability (PCPP) related to L2-regularized logistic regression.
Here we can make the assignment:

- $PCPP > 0.5$
  Here λ_{ID} is added to the set 'Relevant'
- $PCPP \leq 0.5$
  Here λ_{ID} is added to the set 'Irrelevant'

When all labels are added to one of the two sets, the final label ranking can be determined. Therefore we take the absolute path-probability, with that the leaf-label was predicted from the root-node of the tree. Since the leaf-label is also a meta-label with respect to the root-node, we have a child-to-parent relation here.

The ranking is finally built over concatenation of the folder-descriptor-titles of the two ordered sets.

5.6 Test Instance Prediction, Exploration of the trained Hierarchy

When a new bookmark should be added to the bookmarks-tree, we first have to import the referenced document-sequence using our wrappers. When the tokens are indexed we can build the sparse feature-vector representation of the instance for testing using the training vocabulary. On testing of the query-instance we first create a prediction-dataset, where the root-node is assumed to be predicted with a probability of 1.0 (100%), since the user wants to decide definitely to include the bookmark to the subtree, rooted at the root- node. Now we can explore the trained hierarchical models to determine which labels should be added to one part of the bipartition.
Additionally to the basic threshold 0.5 a dynamic threshold should be used in order to further expand the predictions.

Predictions: Relevant: Basic

Here we can explore the models in a type of Breadth-First-Search.

1. Load all prediction-datasets, that were not visited and where the positive class was predicted with higher than 0.5
2. For each prediction (parent-prediction) in list
   a) Get the Parent-BR-ID
   b) Set the ID as root-node of the next subtree
   c) Load all models, assigned to that subtree, using the ID as Parent-MLC-ID
   d) For each model in list
      i. Generate a prediction of the test-instance for the model
      ii. Register the prediction as dataset to the database-table
   e) Set for the current parent-prediction: visited := true

When the search is finished, all predictions are visited, which predict the child from the parent with higher than 0.5.

Predictions: Relevant: Threshold

In our experiments we want to analyse, how the results change, when the remaining unvited models are explored with respect to a specific threshold. Therefore we just set a threshold $0.0 \leq t < 0.5$ and run the step, explained before for model-expansion. This will give us the possibility, that leaf-labels that were not found for the relevant-set before, can be added now if on a lower level in the tree a leaf-label is positively predicted from its parent with higher than 0.5. Trivially, if the threshold is set to 0.5, this step is finished.

Predictions: Irrelevant

When for a model the positive class was predicted with not more than the value of the threshold, the subtree rooted at the class of this model was not further explored. For that case we define, that this subtree was pruned for the exporation of the labels for the relevant-part of the bipartition. Here we want to show, how these pruned subtrees can now be included to determine the second part of the bi-partition, since we want to build a label-ranking over all leaf-labels in the overall tree. First we have to

calculate for each pruned subtree, how much overall leaf-items are contained there. Therefore we take the parent-BR-ID for each of these models and load the folder-descriptors. As described in Section 5.3, the total number of leaf-items in the inner-set inside the subtree, rooted at these folder is covered by the fields 'leafItemCount' and 'innerLeafItemCount'. Next all leaf-labels of the subtree are loaded. For each of these folder-descriptors, we take the value of the 'leafItemCount' and 'innerLeafItemCount'- field and calculate the apriori-probability:

- $apriori(Leaf|Meta) := \frac{leafItemCount(Leaf)}{leafItemCount(Meta)+innerLeafItemCount(Meta)}$

6 Multilabel Learning: Approaches

For each approach we consider the two phases training and prediction.

6.1 Approach 1: 1-Level, flat Tree (Binary Relevance)

Here no hierarchy is considered, such that we assume that the parent of each folder is the root node.

Phase: Training

In a flat tree, the only trainable folder is the root-node, since the other folders have no subtree. Therefore we build the mlc index structure only for that node and generate the svm-training-models. This scheme is often called 'Binary Relevance'. For our formulation we generally treat the BR_ID as an aggregated meta-label to simplify the formulation for the hierarchical case. In the flat case trivially, this meta-label is identical to the leaf-label. We can observe, that in each model-file each distinct instance of the overall tree is used, such that we have with a growing dataset a large size of the negative-set compared to the positive-set for the binary task. This is often called 'class imbalance'.

Phase: Prediction

For a new query instance, we can just make a prediction for each model. Since the aggregated meta-label is here identical to the leaf-label, we can add it to the bipartition depending on the prediction probability. Finally both sets are sorted by the probability and the concatenation of these two ordered sets determines the ranking.

6.2 Approach 2,3: Multi-Level, hierarchical Tree (HOMER)

Here we take a possible hierarchical structure into account. When for two different folders not both have the root-node as parent, we need a more distinguished approach.

Phase: Training

In a hierarchical tree, we have multiple subtrees through the hierarchical decomposition. Therefore we build the mlc index structure for each subtree and generate the svm-training-models. Here we are applying an aggregated binary relevance technique on each subtree. Since each subtree covers only a subset of the instances of the overall tree, the ratio between the size of the positive-set and negative-set for each binary model is more balanced.

Phase: Prediction

Here, for a new query instance the ranking is determined over the hierarchical exploration as explained in Section 5.6.

7 Experiments

7.1 Evaluation Levels

For each test-instance a bipartition of the set of all leaf-labels is determined.
For the evaluation we take different levels into account:

- L_0:
  In the default evaluation the hierarchical relation between the labels is not considered. Therefore we assume that all labels are on the same level.

- $L_1 - L_5$:
  For the hierarchical evaluation we want to take the hierarchical relations between the labels into account. In the Reuters-RCV1-V2 dataset, we can observe that the labels from the root-node down the hierarchy can be assigned to the levels 1-5.

For the evaluation measures of a specific level L_i we define the base-set of labels:

- L $:= \{\lambda | \lambda \in L_i\}$

Therefore the bipartition of level 0 can be distinguished into 5 bipartitions, one for each hierarchy-level.

This is denoted by: $(P_0, N_0) = (P_1, N_1), (P_2, N_2), (P_3, N_3), (P_4, N_4), (P_5, N_5)$

For each hierarchy-level now the bipartition-sets are ordered by the score of each item, building the ranking. Each hierarchy-level can be evaluated using the evaluation-measures.

7.2 Evaluation Measures

When a multi-label ranking for a test-instance x_i in form of a ordered bipartition is predicted, we can evaluate the prediction using commonly known evaluation measures.

For the evaluation of the bipartition we will use the following label-based measures:

- Confusion Matrix
  Here we calculate for the prediction the values (tp, fp, fn, tn) of a Confusion Matrix C.
  tp denotes the number of labels, that were predicted positive and classified positive.
  fp denotes the number of labels, that were predicted positive and classified negative.
  fn denotes the number of labels, that were predicted negative and classified positive.
  tn denotes the number of labels, that were predicted negative and classified negative.
  Using the values of C, we calculate the measures (Precision, Recall, F1).

  - $Precision(C) := \frac{tp}{tp+fp}$
  - $Recall(C) := \frac{tp}{tp+fn}$
  - $F1(C) := \frac{2*Precision(C)*Recall(C)}{Precision(C)+Recall(C)}$

- Hamming Loss
  The Hamming Loss refers to the relevant labels that were predicted or the irrelevant labels that were not predicted.
  It is usually denoted by: $HammingLoss(x_i) = \frac{|Y_i \Delta Z_i|}{|L|}$
  Here the symmetric difference $Y_i \Delta Z_i$ denotes the misclassified labels $\lambda \in L$, where $Y_{i,\lambda} \neq Z_{i,\lambda}$ and is given by:
  $Y_i \Delta Z_i := \{\lambda | (\lambda \in Y_i \wedge \lambda \notin Z_i) \vee (\lambda \notin Y_i \wedge \lambda \in Z_i)\}$

  This denotes the label-set as result of a set-theoretic XOR-function and therefore the complement to the common intersection of Y_i and Z_i.

  It has also been shown, that the Hamming Loss can be considered as macro-averaged classification error related to the confusion matrix:
  $HammingLoss(C) = \frac{fp+fn}{tp+fp+fn+tn}$

For the evaluation of the bipartition-ordering we will use the following label-based measures:

- Ranking Loss
  For the Ranking Loss first the ErrorSet for an instance x_i is calculated. This is given by:
  $ErrorSet_i = \{(\lambda_a, \lambda_b) : r_i(\lambda_a) > r_i(\lambda_b), (\lambda_a, \lambda_b) \in Y_i \times \overline{Y_i}\}$
  There the rank-position $r_i(\lambda)$ is determined as described in Section 5.2
  The Ranking Loss is then defined by:
  $RankingLoss(x_i) = \frac{1}{|Y_i|*|\overline{Y_i}|} * |ErrorSet_i|$

- Average Precision
  For the Average Precision first the RankedAbove value for an instance x_i is calculated. This is given by:
  $RankedAbove_{i,\lambda} = \{\lambda' \in Y_i : r_i(\lambda') \leq r_i(\lambda)\}$
  The Average Precision is then defined by:
  $AveragePrecision(x_i) = \frac{1}{|Y_i|} \sum_{\lambda \in Y_i} \frac{RankedAbove_{i,\lambda}}{r_i(\lambda)}$

For the evaluation of all dataset-bipartitions we will calculate the following aggregated measures:

- Confusion Matrix
  Here we consider the Confusion Matrix C as aggregated over all cells of the instance-based matrices C_i.
  Therefore it is denoted by: $C = \sum_{i=1}^{|D|} C_i$
  For that aggregated matrix we can calculate the Micro-Average measures (Precision, Recall, F1). These are denoted by:
  $MicroAvgPrecision(C) := \frac{tp}{tp+fp}$
  $MicroAvgRecall(C) := \frac{tp}{tp+fn}$
  $MicroAvgF1(C) := \frac{2*MicroAvgPrecision(C)*MicroAvgRecall(C)}{MicroAvgPrecision(C)+MicroAvgRecall(C)}$

- Hamming Loss
  Here we calculate the (Min, Max, Avg) value over all instances in the dataset D.
  The average value is given by:
  $HammingLossAvg(D) = \frac{\sum_{x_i \in D} HammingLoss(x_i)}{|D|}$
  The (min, max) values are calculated using the instance-based matrices C_i:
  $HammingLossMin(D) = min_{x_i \in D}\{HammingLoss(C_i)\}$
  $HammingLossMax(D) = max_{x_i \in D}\{HammingLoss(C_i)\}$

For the evaluation of all bipartition-orderings in the dataset we will calculate the following aggregated measures:

- Ranking Loss
  Here we calculate the (Min, Max, Avg) value over all instances in the dataset D.
  The average value is given by:
  $RankingLossAvg(D) = \frac{\sum_{x_i \in D} RankingLoss(x_i)}{|D|}$
  The (min, max) values are calculated by:
  $RankingLossMin(D) = min_{x_i \in D}\{RankingLoss(x_i)\}$
  $RankingLossMax(D) = max_{x_i \in D}\{RankingLoss(x_i)\}$

- Average Precision
  Here we calculate the average value over all instances in the dataset D.
  The value is given by:
  $AveragePrecision(D) = \frac{\sum_{x_i \in D} AveragePrecision(x_i)}{|D|}$

7.3 Setup

For our experiments we will use the Reuters RCV1 V2 dataset, which we described before in Section 3.2. Out of the set of test-instances we will take the first 200.000 documents.

Therefore we define: $m = |D| = 200.000$

As proposed in [31] we can calculate the following statistics for our considered subset:

- $Label - Cardinality = \frac{1}{m} \sum_{i=1}^{m} |Y_i|$
  Avg Number of Labels per Test-Instance
- $Label - Density = \frac{1}{m} \sum_{i=1}^{m} \frac{|Y_i|}{q}$
  Avg Number of Labels per Test-Instance divided by q.
  Here q is the number of labels on the considered level.

The dataset (200.000 test-instances) is specified by the following numbers:

Level	# of Labels	Label-Cardinality	Label-Density	# of Instance-Assignments for Level-Labels
0	101	3.2429	0.0321	1478091
1	0	0.0	0.0	0
2	4	1.1781	0.2945	945334
3	54	1.4623	0.0271	1128784
4	42	1.1354	0.0270	509546
5	1	1.0	1.0	23211

7.3.1 Configuration of the Classifier

For the binary SVM-classifiers we will use the default parameters:

- Cost: 1.0
- Bias: 1.0
- Tolerance: 0.01
- Weight of positive class: 1.0
- Weight of negative class (w_{neg}): 1.0

In order to consider the class imbalance problem [37] [38] we will use the following parameters for the binary SVM-classifiers in Approach 3:

- Cost: 15.0
- Weight of negative class
  $pnratio := \#instances(positive-set)/\#instances(negative-set)$
  $w_{neg} := pnratio + (1 - pnratio)/2$

7.3.2 Configuration of the Hierarchy-Exploration

For the exploration we will use the default parameters:

- Dynamic Threshold: 0.5

For the exploration in Approach 3 we will use the parameter:

- Dynamic Threshold: 0.0

7.3.3 Approach 1

As explained in Section 6.1 we only have to explore binary models for one subtree, since all leaf-labels are assumed to be assigned to the root-node. This case can be treated as single multilabel classifier.

7.3.4 Approach 2,3

As explained in Section 6.2 the hierarchical (child-to-parent) relations between the labels are taken into account. Through the hierarchical decomposition we have to build one multilabel classifier for each subtree. Again each multilabel-classifier-subtree is represented using a set of binary models. Since the predicted output-label of one binary model is treated as aggregated parent of a possible subtree, it can be decided if this predicted label should be further resolved in exploration.

Approach 2.1

For the bipartition-ordering we will use the following parameters:

- Ordering of Bipartition-Positive (Z): Apriori-Probability / 2 + 0.5
- Ordering of Bipartition-Negative ($\overline{Z}$): Apriori-Probability / 2

Approach 2.2

For the bipartition-ordering we will use the following parameters:

- Ordering of Bipartition-Positive (Z): Path-Probability / 2 + 0.5
- Ordering of Bipartition-Negative ($\overline{Z}$): Apriori-Probability / 2

Approach 3

For the bipartition-ordering we will use the following parameters:

- Ordering of Bipartition-Positive (Z): Path-Probability / 2 + 0.5
- Ordering of Bipartition-Negative ($\overline{Z}$): Path-Probability / 2

8 Evaluation

In the Reuters-RCV1-V2 dataset the root-node on level 1 has no documents assigned directly as leaf-item. Therefore it is initially assigned to the Bipartition-Negative-Set with an apriori-probability of 0.0 and the set of classified relevant labels is empty. Since we have only one label on level 5, the set of classified irrelevant labels for that level is empty and the ranking-loss can not be computed. Therefore we set the value to 0.0. For the labels of level 2 we can observe in the dataset that each instance is assigned in the multi-label associations to at least one of these labels.

8.1 Approach 1

- Bipartition-Measures:

		Micro Average			Hamming-Loss		
Test-Instances	Level	Precision	Recall	F1	Min	Max	Average
200000	0	0.9529	0.5208	0.6735	0.0000	0.1635	0.0157
200000	1	1.0000	1.0000	1.0000	0.0000	0.0000	0.0000
200000	2	0.9487	0.8224	0.8810	0.0000	1.0000	0.0654
	3	0.9560	0.3230	0.4829	0.0000	0.1455	0.0178
	4	0.9643	0.4096	0.5750	0.0000	0.1395	0.0090
200000	5	0.7923	0.1227	0.2125	0.0000	1.0000	0.0092

  For the levels 3-5 we can observe low Recall- and F1-values as the number of negative examples in the binary models is high related to the number of positive examples, since each binary model in the approach contains all training-examples of the tree. This shows the class imbalance problem in the default SVM-configuration.

- Ranking-Measures:

		Ranking-Loss			Ranking-Precision
Test-Instances	Level	Min	Max	Average	Average
200000	0	0.0000	0.5850	0.0245	0.8556
200000	1	0.0000	0.0000	0.0000	NaN
200000	2	0.0000	1.0000	0.0301	0.9679
	3	0.0000	1.0000	0.0316	0.8403
	4	0.0000	1.0000	0.0193	0.8540
200000	5	0.0000	0.0000	0.0000	1.0000

For the evaluation of the levels 2-5 we can make the following observations:

- Level 2
  - Bipartition Measures
    The Micro-Average values are high, since each instance in the dataset is at least assigned to one of these labels. The average Hamming-Loss is high with respect to the other levels due to the default configuration of the SVM, resulting in high FN-values in the confusion matrix.
  - Ranking Measures
    The high FN-values are resulting in a high average Ranking-Loss.
- Level 2 vs. 3
  - Bipartition Measures
    The Micro-Average values for Recall and F1 for level 3 are low. Here a large number of labels on level 3 is not predicted. In the default SVM-configuration we have a high weight on the negative class due to the large set of negative examples.
  - Ranking Measures
    The Ranking-Loss is increasing and the AveragePrecision is decreasing, since a large set of labels is not covered due to the class imbalance.
- Level 3 vs. 4

- Bipartition Measures
  The Micro-Average values for Recall and F1 are increasing with a decreasing number of labels to be predicted. The values for Hamming-Loss-Max are low, since the set of labels for each level is large and the values for (FP, FN) in the confusion matrix are low. The average Hamming-Loss is decreasing with a decreasing number of labels to be predicted.
- Ranking Measures
  The average Ranking-Loss is decreasing and the AveragePrecision is increasing, since a lower number of labels needs to be predicted.

- Level 4 vs. 5
  - Bipartition Measures
    The Micro-Average values are decreasing and the average Hamming-Loss is increasing, since the set of labels to be predicted is smaller and the large set of negative examples in the binary models leads in the default configuration to wrong predictions due to the class imbalance problem.

8.2 Approach 2

- Bipartition-Measures:

		Micro Average			Hamming-Loss		
Test-Instances	Level	Precision	Recall	F1	Min	Max	Average
200000	0	0.4511	0.6201	0.5223	0.0000	0.1635	0.0354
200000	1	1.0000	1.0000	1.0000	0.0000	0.0000	0.0000
200000	2	0.9487	0.8224	0.8810	0.0000	1.0000	0.0654
	3	0.2293	0.4813	0.3107	0.0000	0.2545	0.0551
	4	0.7794	0.5625	0.6534	0.0000	0.1395	0.0088
200000	5	0.9462	0.1479	0.2559	0.0000	1.0000	0.0087

- Approach 2.1 Ranking-Measures:

		Ranking-Loss			Ranking-Precision
Test-Instances	Level	Min	Max	Average	Average
200000	0	0.0000	0.6275	0.0940	0.4622
200000	1	0.0000	0.0000	0.0000	NaN
200000	2	0.0000	1.0000	0.0848	0.9347
	3	0.0000	0.9815	0.1411	0.3557
	4	0.0000	1.0000	0.0412	0.6550
200000	5	0.0000	0.0000	0.0000	1.0000

- Approach 2.2 Ranking-Measures:

		Ranking-Loss			Ranking-Precision
Test-Instances	Level	Min	Max	Average	Average
200000	0	0.0000	0.6650	0.0403	0.8023
200000	1	0.0000	0.0000	0.0000	NaN
200000	2	0.0000	1.0000	0.0306	0.9675
	3	0.0000	0.9815	0.0727	0.7218
	4	0.0000	1.0000	0.0230	0.7897
200000	5	0.0000	0.0000	0.0000	1.0000

For the evaluation of the levels 2-5 we can make the following observations:

- Level 2 vs. 3
  - Bipartition Measures
    The Micro-Average values are decreasing, since the set of labels to be predicted for level 3 is large. Here a large set of labels is not explored, since the default threshold 0.5 is used. The Hamming-Loss-Max value for level 3 is low, since the number of labels to be predicted is large and the FN-value in the confusion matrix is small. The average Hamming-Loss is lower, since the number of labels to be predicted increases more than the (FP, FN)- values.
  - Ranking Measures
    The Ranking-Loss-Max value is high but lower, since a large set of labels is to be predicted and not covered. The average Ranking-Loss is higher, since the set of labels on level 3 is large. The AveragePrecision is decreasing, since a large set of labels to be predicted is not covered.
  - Ranking Measures: Approach 2.1, Approach 2.2
    The Ranking-Loss-Average is increasing and the AveragePrecision is decreasing.

- Level 3 vs. 4
  - Bipartition Measures
    The values for the Micro-Average measures are increasing, since the set of negative examples for the binary models is smaller that predict the labels of level 4. For the same reason the Hamming-Loss-Max and the average Hamming-Loss are decreasing.
  - Ranking Measures
    The average Ranking-Loss is decreasing and the AveragePrecision is increasing, since the number of labels to be predicted and the set of negative examples in the binary models is smaller. As the number of instance-assignments for the labels of level 4 is smaller compared to level 3, the set of instances in the decomposed binary models is smaller.
  - Ranking Measures: Approach 2.1, Approach 2.2
    The Ranking-Loss-Average is decreasing and the AveragePrecision is increasing.
- Level 4 vs. 5
  - Bipartition Measures
    The Micro-Average-Precision is increasing, since the explored binary models make good predictions. The Micro-Average Recall and F1 values are decreasing, since for a large set of test-instances the binary models are not explored due to the static threshold.

8.3 Approach 3

- Bipartition-Measures:

		Micro Average			Hamming-Loss		
Test-Instances	Level	Precision	Recall	F1	Min	Max	Average
200000	0	0.4558	0.7871	0.5773	0.0000	0.2115	0.0359
200000	1	1.0000	1.0000	1.0000	0.0000	0.0000	0.0000
200000	2	0.9227	0.8982	0.9103	0.0000	1.0000	0.0522
	3	0.2660	0.7143	0.3876	0.0000	0.2909	0.0582
	4	0.7444	0.7458	0.7451	0.0000	0.1395	0.0076
200000	5	0.8653	0.6487	0.7415	0.0000	1.0000	0.0046

Since all subtrees are explored with the dynamic threshold 0.0, we can observe low FN values in the Confusion Matrix.

- Approach 3 Ranking-Measures:

		Ranking-Loss			Ranking-Precision
Test-Instances	Level	Min	Max	Average	Average
200000	0	0.0000	0.7228	0.0236	0.8802
200000	1	0.0000	0.0000	0.0000	NaN
200000	2	0.0000	1.0000	0.0236	0.9751
	3	0.0000	0.9815	0.0403	0.8414
	4	0.0000	1.0000	0.0129	0.8726
200000	5	0.0000	0.0000	0.0000	1.0000

Here more labels are moved from bipartition-negative to bipartition-positive. Additionally the labels in bipartition-negative are ordered by the path-probability, leading to an overall higher average Ranking-Precision.

For the evaluation of the levels 2-5 we can make the following observations:

- Level 2 vs. 3
  - Bipartition Measures
    The Micro-Average values are decreasing, since the set of labels to be predicted for level 3 is large. The Hamming-Loss-Max value for level 3 is low, since the number of labels to be predicted is large and the FN-value in the confusion matrix is small.
  - Ranking Measures
    The average Ranking-Loss is increasing and the AveragePrecision is decreasing, since the set of labels on level 3 is large.
- Level 3 vs. 4
  - Bipartition Measures
    Comparable high number of labels to be predicted on both levels Labels to be predicted lower on level 4. The values for the Micro-Average measures are increasing, since the set of negative examples for the binary models is smaller that predict the labels of level 4. For the same reason the Hamming-Loss-Max and the average Hamming-Loss are decreasing.

  - Ranking Measures
    The average Ranking-Loss is decreasing and the AveragePrecision is increasing, since the number of labels to be predicted and the set of negative examples in the binary models is smaller. As the number of instance-assignments for the labels of level 4 is smaller compared to level 3, the set of instances in the decomposed binary models is smaller.

- Level 4 vs. 5
  - Bipartition Measures
    The Micro-Average-Precision is increasing, since the explored binary models make good predictions.

8.4 Approach 1 vs. 2

- Bipartition Measures:
  - Level 3,4,5: Here we can observe an increasing Micro Average Recall value
  - Level 3,4,5: Here we can observe an increasing Hamming-Loss Average value
  - Level 4,5: Here we can observe an increasing Micro Average F1 value
  - Level 5: Here we can observe an increasing Micro Average Precision value

  At the levels 4,5 in the hierarchical modelling the negative-set of instances is small. Therefore the weight between the positive class and negative class is relatively balanced.

- Ranking Measures:
  - Level 2 vs 3: Here the Ranking-Loss Average is increasing and the Ranking-Precision Average is decreasing
  - Level 3 vs 4: Here the Ranking-Loss Average is decreasing and the Ranking-Precision Average is increasing

8.5 Approach 2 vs. 3

- Bipartition Measures:
  - Level 0,2,3,4,5: Here we can observe an increasing Micro Average Recall and F1 values
  - Level 4,5: Here we can observe an decreasing Hamming-Loss Average value

  At the levels 4,5 in the hierarchical modelling the negative-set of instances is small. Therefore the weight between the positive class and negative class is relatively balanced.

  Since all subtrees of the hierarchical dataset are explored we can observe higher tp and fp values in the confusion-matrices for the levels 0,2,3,4.

- Ranking Measures:
  - Level 0,2,3,4,5: Here the Ranking-Loss Average is decreasing
  - Level 0,2,3,4: Here Ranking-Precision Average is increasing

  Here both parts of the bipartition are ordered by the path-probability.

9 Conclusions, Further Work

In this thesis we could show the evaluation of two different decomposition techniques for the multi- label hierarchical dataset Reuters-RCV1-V2.
Therefore we calculated bipartition-measures and ranking-measures for different levels of the hierarchy. For the hierarchical dataset-modelling we could show an improvement for the lower levels of the hierarchy, since there less instances are considered in the negative-set of the binary models. We conclude that the class imbalance there is relatively low.
Using the dynamic threshold in the exploration, a higher cost value and a lower weight of the negative class for the binary models we could show an improvement in the rankings.

Basically we set each subtree-MLC (set of binary classifiers) to the default SVM-configuration for each hierarchy-level. Looking forward to further developments, it can be considered to set different configurations for the levels / subtree-MLCs.

For the further work, user-generated content in terms of text-summaries and the reorganization of the tree-structure can be taken into account. This is partially adressed as gradual forgetting.

Acknowledgements

We want to thank Frédéric R. Bürthel (MarketDialog GmbH) for his support in Metadata Modelling and Digital Library Tools.

List of Figures

Bibliography

[1] D. Barreau and B. Nardi, "Finding and reminding: File organization from the desktop," *ACM SIGCHI Bulletin*, vol. Vol. 27 (3), July 1995.

[2] S. Dumais, E. Cutrell, J. J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins, "Stuff i've seen: a system for personal information retrieval and re-use," in *SIGIR '03 Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, vol. 26, pp. 72 – 79, ACM New York, NY, USA, 2003.

[3] V. Bush, "As we may think," *Atlantic Monthly*, vol. 176, pp. 101 – 108, 1945.

[4] D. Abrams, R. Baecker, and M. Chignell, "Information archiving with bookmarks: Personal web space construction and organization," *In ACM SIGCHI '98*, pp. 41–48, 1998.

[5] M. J. Hofmann, L. Kuchinke, C. Biemann, S. Tamm, and A. M. Jacobs, "Remembering words in context as predicted by an associative read-out model," *Frontiers in psychology*, vol. 2, 2011.

[6] J. Grainger and A. M. Jacobs, "Orthographic processing in visual word recognition: a multiple read-out model.," *Psychological review*, vol. 103, no. 3, p. 518, 1996.

[7] M. S. Seidenberg and J. L. McClelland, "A distributed, developmental model of word recognition and naming," *Psychological review*, vol. 96, p. 523, 1989.

[8] M. E. Casey and L. C. Savastinuk, "Library 2.0: Service for the next-generation library," *Library Journal*, 2006.

[9] J. M. Maness, "Library 2.0 theory: Web 2.0 and its implications for libraries," *Webology*, vol. Vol. 3 (2), June, 2006.

[10] A. Foster and N. Ford, "Serendipity and information seeking: an empirical study," *Journal of Documentation*, vol. Vol. 59 (3), pp. 321 – 340, 2003.

[11] A. Spink, M. Park, B. J. Jansen, and J. Pedersen, "Multitasking during web search sessions," *Inf. Process. Manage.*, vol. 42, pp. 264–275, Jan. 2006.

[12] S. Whittaker and C. Sidner, "Email overload: exploring personal information management of email," in *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground*, (Vancouver, British Columbia, Canada), ACM, 1996.

[13] C. Speier, J. S. Valacich, and I. Vessey, "The influence of task interruption on individual decision making: An information overload perspective," *Decision Sciences*, vol. Vol. 30, pp. 337 – 360, 1999.

[14] O. Serrat, "Showcasing knowledge," *Washington, DC: Asian Development Bank*, 2010.

[15] P. Morville and L. Rosenfeld, *Information Architecture for the world wide web: designing large-scale web sites*. O'Reilly, 2006.

[16] M. Ehrig and A. Maedche, "Ontology-focused crawling of web documents," in *SAC '03 Proceedings of the 2003 ACM symposium on Applied computing*, pp. 1174–1178, 2003.

[17] J. Huang and R. W. White, "Parallel browsing behavior on the web," in *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, HT '10, (New York, NY, USA), pp. 13–18, ACM, 2010.

[18] R. D. García, *Unterstützung des Ressourcen-Basierten Lernens in Online Communities - Automatische Erstellung von Grosstaxonomien in verschiedenen Sprachen*. PhD thesis, Fachbereich Elektrotechnik und Informationstechnik der Technischen Universität Darmstadt, 2013.

[19] D. E. Rumelhart and J. L. McClelland, "An interactive activation model of context effects in letter perception: Ii. the contextual enhancement effect and some tests and extensions of the model.," *Psychological Review*, vol. 89(1), pp. 60 – 94, 1982.

[20] M. Coltheart, K. Rastle, C. Perry, R. Langdon, and J. Ziegler, "Drc: a dual route cascaded model of visual word recognition and reading aloud.," *Psychological review*, vol. 108, no. 1, p. 204, 2001.

[21] M. Lansdale, "The psychology of personal information management," *Applied Ergonomics*, pp. 55 – 66, 1988.

[22] R. Mander, G. Salomon, and Y. Y. Wong, "A "pile" metaphor for supporting casual organization of information," in *CHI '92 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 627 – 634, ACM, 1992.

[23] I. Katakis, G. Tsoumakas, and I. P. Vlahavas, "Multilabel text classification for automated tag suggestion," in *Proceedings of the ECML/PKDD-08 Workshop on Discovery Challenge*, 2008.

[24] E. S. Xioufis, M. Spiliopoulou, G. Tsoumakas, and I. & Vlahavas, "Dealing with concept drift and class imbalance in multi-label stream classification," in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume*, vol. Vol. 2, pp. 1583 – 1588, AAAI Press, 2011.

[25] E. L. Cohen, "Online journalism as market-driven journalism," *Journal of Broadcasting & Electronic Media*, vol. Vol. 46 (4), pp. 532 – 548, 2002.

[26] A. Bruns, *Gatewatching: Collaborative online news production*, vol. Vol. 26. Peter Lang Pub Incorporated, 2005.

[27] D. D. Lewis, Y. Yang, T. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of Machine Learning Research*, vol. Vol. 5, pp. 361 – 397, 2004.

[28] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. Vol. 97 (1-2), pp. 273 – 324, December 1997.

[29] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *ECML '98 Proceedings of the 10th European Conference on Machine Learning*, pp. 137 – 142, Springer-Verlag London, UK, 1998.

[30] K. Brinker, J. Fürnkranz, and E. Hüllermeier, "A unified model for multilabel classification and ranking," in *IN PROCEEDINGS OF THE 17TH EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pp. 489–493, 2006.

[31] G. Tsoumakas, I. Katakis, and I. P. Vlahavas, "Mining multi-label data," in *In Data Mining and Knowledge Discovery Handbook*, pp. 667–685, 2010.

[32] G. Tsoumakas, I. Katakis, and I. P. Vlahavas, "Effective and efficient multilabel classification in domains with large number of labels," in *ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 2008.

[33] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear a library for large linear classification," *The Journal of Machine Learning Research*, vol. Vol. 9, pp. 1871 – 1874, 2008.

[34] E. Leopold and J. Kindermann, “Text categorization with support vector machines. how to represent texts in input space?,” *Machine Learning*, vol. Vol. 46 Issue 1-3, pp. 423 – 444, 2002.

[35] L. Cai and T. Hofmann, “Hierarchical document categorization with support vector machines,” in *CIKM ’04 Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pp. 78 – 87, ACM New York, NY, USA, 2004.

[36] M. Lan, C.-L. Tan, H.-B. Low, and S.-Y. Sung, “A comprehensive comparative study on term weighting schemes for text categorization with support vector machines,” in *WWW ’05 Special interest tracks and posters of the 14th international conference on World Wide Web*, pp. 1032 – 1033, ACM New York, NY, USA, 2005.

[37] U. Brefeld, P. Geibel, and F. Wysotzki, “Support vector machines with example dependent costs,” in *Machine Learning: ECML 2003*, pp. 23 – 34, Springer, 2003.

[38] C. Drummond and R. C. Holte, “Severe class imbalance: Why better algorithms aren’t the answer,” in *Machine Learning: ECML 2005*, pp. 539 – 546, Springer, 2005.