Diploma thesis

Tag prediction in micro-blogging systems

Tobias Schlitt

Diploma thesis
at the department for computer science
of the TU Dortmund university

Dortmund, October 2, 2010

Supervisors:

Prof. Dr. Katharina Morik
Dipl.-Inform. Christian Bockermann

We are like dwarfs on the shoulders of giants.

– Bernard of Chartres

Acknowledgment

I would like to thank everyone who has supported me while writing this diploma thesis, especially Prof. Dr. Katharina Morik and Dipl.-Inform. Christian Bockermann, my supervisors. In addition, I would like to thank my beloved parents for supporting me the whole way through my studies and my girlfriend Jenny who always stood by my side.

Contents

List of Figures

List of Tables

Notation

$A, B, C, \ldots$	– denote sets of elementary objects.
$a, b, c, \ldots$	– are used to denot elementary objects.
$\mathcal{A}, \mathcal{B}, \mathcal{C}, ...$	– denote sets of sets, so-called set families.
$\mathbb{A}, \mathbb{B}, \mathbb{C}, ...$	– denote special sets of base elemnts, which are used to create nested set families.

Chapter 1

Introduction

With the emergence of the *Web 2.0* movement, characterized by the change of mind from a content-centric to a user-centric web approach, a variety of new web application types appeared in the past years. All of them have in common that the user is no more in the role of a pure consumer of content, but actively publishes information, so-called user-generated content. Furthermore, applications tend to be implemented as social networks, where fundamental features are user interaction facilities such as community building, friendship relations, commentary functionality and private messaging.

One recent type of social network applications in the Web 2.0 are micro-blogging services. These services allow users to publish limited size text statements, so denoted as status updates or micro-blogging entries. Users tend to write such entries in varying frequencies, ranging from once a week to multiple times per hour. The content of micro-blogging entries is commonly related to daily life experiences, commentary about breaking news and recent events. Sharing of links to websites, photos and videos, but also discussions, are further applications of micro-blogging. The social network characteristic of the micro-blogging service is typically reflected in the possibility to subscribe to other users' status updates. The user is then provided with an aggregated stream of micro-blogging entries from all users being subscribed to. Micro-blogging networks are accessed typically from a standard web browser, but also through dedicated client applications, which communicate with the network through a web service interface. This way of usage is commonly exercised via mobile devices, such as cellphones or handheld computers, allowing the users to send status updates from wherever they are. By integrating photo and video upload services into such client applications, users can instantly share impressions from their daily life.

Although being a quite new type of application, micro-blogging has already reached great popularity, even among mainstream web users. The first known micro-blogging service *Twitter*[1], which is still the market leader in the area of dedicated micro-blogging services, registers constant increases in the number of micro-blogging entries published per day, as can be seen in Figure 1.1. Roughly 50 million *Tweets*, as micro-blogging entries are denoted on Twitter, have been published in January 2010, per day. However, the impressive spread of micro-

[1] `http://twitter.com` (2010-07-08)

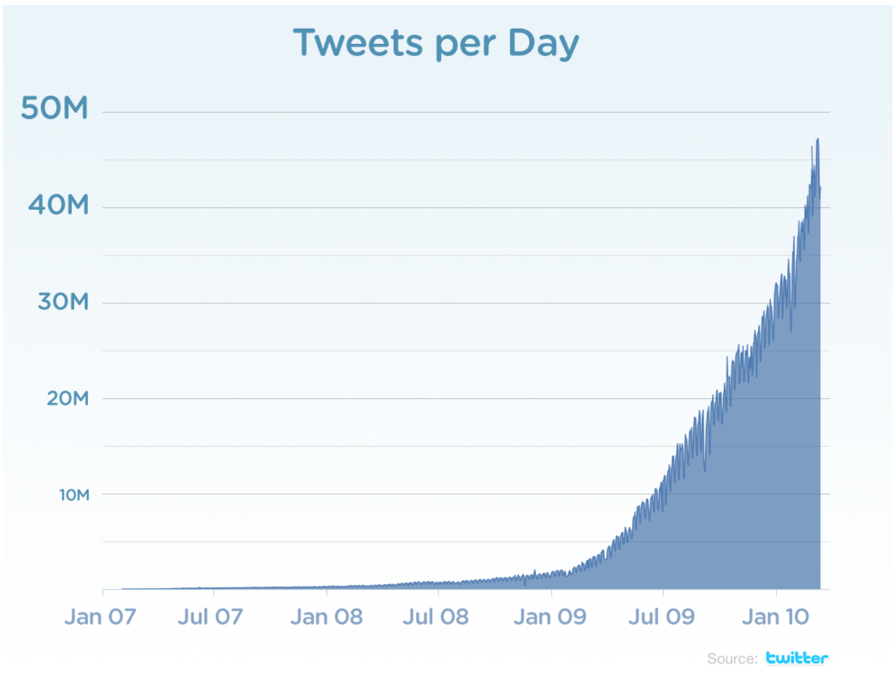


Figure 1.1.: Micro-blogging entries published on Twitter per day[a].

[a]`http://blog.twitter.com/2010/02/measuring-tweets.html` (2010-07-09)

blogging has also to be attributed to the large general-use social networks, such as Facebook[2] and the German StudiVZ[3], which adopted the concept rapidly into their feature repertoire.

1.1. Motivation

A typical feature of Web 2.0 applications is *tagging*, a way of user-based categorization for content items. In contrast to classical categories, which are well thought out to provide a degree of objective semantics, tags consist of arbitrary keywords or short description sentences provided by the users of a system. Users can typically assign an arbitrary number of tags to their own, and sometimes even to another user's, content items. Tags are typically used by the application for information retrieval and recommendation purposes. This observation also holds for micro-blogging services, where users can assign tags to their status updates.

Tagging systems normally suffer from several problems: Firstly, to categorize their content items, users are not enforced to assign tags to them. This often

[2]`http://facebook.com` (2010-07-08)
[3]`http://studivz.net` (2010-07-08)

results in a system with a large proportion of items that are not tagged at all. Secondly, users do mostly not care if they re-use tags they have already assigned to other content items. The effect is, that different keywords, e.g. synonyms and different slang words, are used to denote semantically equivalent categories. The result is a distraction of information retrieval processes. Finally, spelling mistakes lead to a similar result, where such categorized items are literally ignored by search and recommendation features.

One typical approach to mitigate these effects is the usage of a recommendation system for tags on the basis of the earlier tagging behavior of a user. Instead of just allowing the user to enter tags for a content item in form of text, the system displays to the user a selection of existing tags that might be fitting to the content. Users can then select from these tags by simply clicking on them, but also enter different ones in case the recommendation does not reflect their intention. The expected result is that users are motivated to tag their content, i.e. fewer not tagged items exist, and that the use of synonyms for semantically similar content and spelling mistakes are reduced.

1.2. Goal

The goal of this thesis is to develop a system for tag recommendation in micro-blogging systems on the basis of machine learning and data mining techniques. The desired system should learn the tagging behavior of a user in order to recommend tags for new micro-blogging entries. In order to make such a tag recommendation system available to a wide range of web users, the developed approach is meant to be independent from a specific micro-blogging service and client application. Therefore, the goal is to design the system to be feasible for implementation as a web service. This way, any micro-blogging service or client application can integrate the approach without the need of re-implementation.

The automatic categorization of new data examples on basis of a correctly categorized data set is commonly realized by the approach of *supervised learning*, also denoted as *classification*. This approach is also utilized in this thesis. The target is to design a classification approach, which is capable of learning the tag-based categorization habit of a specific micro-blogging user so that new, not tagged micro-blogging entries of this person can be classified correctly with tags this user has created before.

The first basic challenge to be tackled in this direction is the problem of *multi-label classification*. While classical classification approaches do only allow a single *label*, considered a tag in the framework of this thesis, to be predicted for a new example, tagging systems feature the assignment of an arbitrary number of tags. Research literature in the area of multi-label classification is still sparse, although this problem environment is getting more and more evident during the past years. The most common approach to tackle the multi-label problem is to transform it into an equivalent single-label task in order to use existing classifiers. This methodology is also featured in the presented thesis.

The targeted web service environment involves a second challenge to be tackled in this thesis: In classical supervised learning environments, a batch learning approach is utilized. A classifier is trained once on a basis of a well-defined data set, to be then used without further adjustment. However, the tagging behavior of users of micro-blogging systems is expected to evolve over time, caused by breaking news, seasonal events and general change of interest. Furthermore, the desired web service is not expected to be solely used by long-term users, so that a sufficiently large set of data to train the classifier may not be available from scratch.

For these reasons, an anytime algorithm is desired to solve the classification task, so as to classify new examples already in an early training stage without fully completing the training of the classifier. The developed approach should also support adjustment of the classifier to conceive drift over time, so as to react on changing tagging behavior. One approach in this direction, which is featured in research for the past years, is classification on basis of a stream data model. Instead of modeling the data, a classifier works with a single stable database, and examples are considered as a constant stream of data. The presented thesis develops such a stream based classification approach.

In summary, the goals of this thesis can be described as follows:

1. Evaluation and selection of a feasible method to tackle the problem of multi-label classification in order to perform tag-prediction in micro-blogging systems.

2. Integration of the developed classification approach into a stream based data model, in order to tackle classifier adjustment over time.

1.3. Structure

The subsequent content of this thesis is structured as follows:

Chapter 2 presents a detailed problem analysis in respect to tagging in the micro-blogging environment. Furthermore, the goal of this thesis is defined more specifically, a motivating use-case is presented and related literature is summarized.

Chapter 3 develops a vector representation for micro-blogging entries and tag sets, to apply machine learning techniques to this kind of data. Furthermore, this chapter evaluates different multi-label transformation techniques and selects a feasible one for the underlying problem. In addition to that, the selected transformation approach is enhanced by a custom technique to reduce the number of generated labels and the designed multi-label system is empirically evaluated.

Chapter 4 selects and describes the three classification methods from the standard repertoire of machine learning and develops the custom supervised learning approach on basis of kNN and a condensed model representation. Experiments in this section evaluate the feasibility of these four approaches to solve the problem of tag-prediction in micro-blogging systems, on basis of the developed multi-label transformation technique.

Chapter 5 is dedicated to embedding the two classification approaches yielding the best classification quality into a stream data model. Two basic stream classification approaches are developed and evaluated on an empirical basis, in order to analyze their feasibility to solve the desired problem.

Chapter 6 concludes this thesis with summarizing the yielded results and proposing topics for future research in the direction of tag-prediction in micro-blogging services and the machine learning approaches developed throughout this thesis.

Chapter 2

Problem Domain

With the Web 2.0[1] movement, which denotes the migration from a content-centric to the user-centric web, tagging has become a well-established way for categorizing items through user assigned keywords and short descriptions. This concept is also widely used in the recently popular area of micro-blogging, where users provide a constant stream of status updates from their everyday life.

However, the concept of tagging involves several issues, which are to be tackled in the underlying thesis, with methods of machine learning and data mining. In order to get an insight into the problem area, the following Section 2.1 gives a detailed introduction into the concept of tagging, including a formalization and a detailed analysis of the problems to be solved within this thesis. After that, Section 2.2 gives an overview of micro-blogging and compares it to general blogging. Finally, Section 2.3 presents a use case for the tag prediction system developed within this thesis and derives concrete goals for the subsequent chapters.

2.1. Tagging

Tagging is a common way to realize user-based categorization of content items in Web 2.0 applications, e.g. social networks. A tag is a keyword or very short description for an item, categorizing its content. In contrast to classical categorization, tags are free form and can be defined arbitrarily by the users of an application, instead of being professionally edited. Users can typically assign an arbitrary number of tags to their own content; in most applications also to content published by other users. Applications typically make use of tags for the purpose of information retrieval: For example content search, recommendation of similar content items and content navigation.

2.1.1. Tags vs. Categories

Although the concepts of categorization and tagging appear quite similar at a first glance, there are significant differences involved.

[1] `http://oreilly.com/web2/archive/what-is-web-20.html` (2010-05-21)

Categories

The classical way to bring structure into a set of items is to assign each of them to one or more well-defined categories. Categories themselves are commonly organized in terms of a *Taxonomy*, a hierarchical classification scheme, reflecting supertype-subtype relationships through its structure. A typical category tree is visualized in Figure 2.1. Categories get more special from top to bottom.

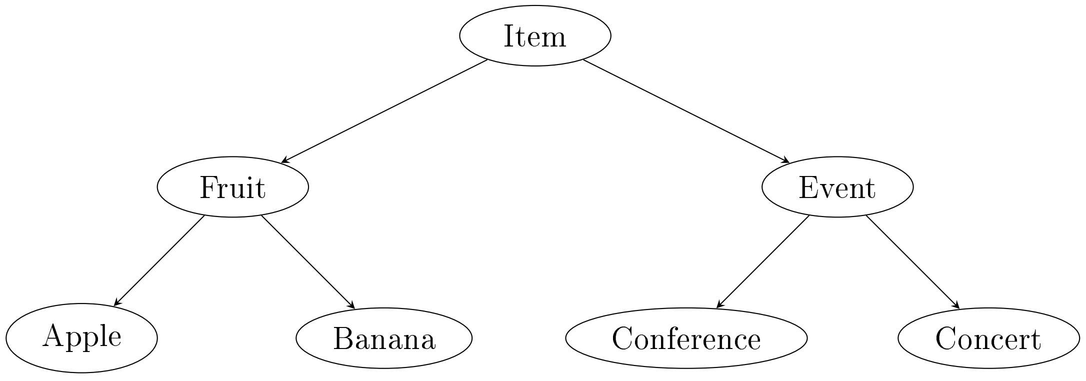


Figure 2.1.: Visualization of a category tree.

From a user's point of view, a category collects items that share one or more attributes and can therefore be considered similar. The category tree of a system is usually created by a dedicated team of editors, in order to ensure certain properties:

Children of a specific category should share a common granularity, i.e. child nodes should divide the elements of the parent category by values of the same set of attributes. It does, for example, not make sense to have the parent category *fruit* with sub-categories *apple* and *red banana from Papua New Guinea* on the same level. The latter sub-category is much more specific than *apple*.

Leaf categories with the same parent should commonly be non intersecting, i.e. a *fruit* should either be an *apple* or a *banana*. A category label should describe the shared aspects of contained items and should provide objective semantic value on its own or at least in relation to its parent category and the underlying system. For example, the category *apple* is ambiguous on its own. Considering its parent category, *fruit*, it becomes clear that *apple* does not denote the IT company in this case.

Before tagging became popular, it was already common in web applications that users categorize items they created. However, it was typically not possible for users to edit the structure of the taxonomy itself.

Tags

While tags fulfill a similar purpose and have a similar appearance than categories, they are fundamentally different: Tags are arbitrarily created and assigned by the users of an application, without influence of professional editors. There is no hierarchical structure implied and the set of tags used in an application can be highly dynamic. The terms *fruit*, *banana* and *apple* can for example be tags, but so can be *I like this* and *IPC2010SE*.

Since users are not educated to design tags with the properties that are expected to hold for categories, several differences exist between both types of item classification.

Tags vary extremely in terms of granularity and item sets defined by tags can have arbitrary intersection. Due to the missing structure, requirements in terms of intersection do not even make sense for tags at all. Additionally, the semantic value of tags varies highly, depending on the user's social background, educational level and interests.

These characteristics of tags are also reflected in the commonly used visualization form: A *tag cloud* (Figure 2.2). Tags are placed arbitrarily, introducing the missing structure. The font size of a tag reflects the number of items this tag is assigned to. The varying granularity of tags can be seen here. It can generally be noted, that tags often imply very few semantics on their own, without taking the assigning user or another special context into account. Therefore, tags can be ambiguous and misleading.

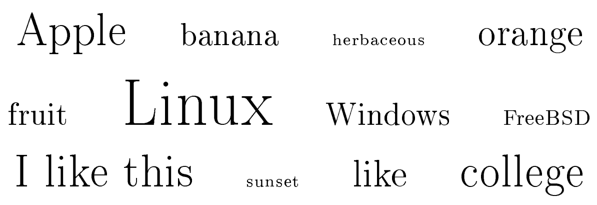


Figure 2.2.: A tag cloud. Font size indicates popularity.

Examining the example tags presented at the beginning of this section one can observe the following concrete problems:

apple	This tag is ambiguous. It is not clear if it refers to the fruit or the IT company, due to missing context, e.g. through a parent category.
IPC2010SE	Without a given context, this tag does not provide any intuitive semantic value. The background here is, that it is a common abbreviation for "International PHP Conference 2010 Spring Edition".
I like this	This tag does not provide any objective semantic value. It is highly user-specific, since it does not imply a shared attribute for its assigned items if the assigning user is unknown.

2.1.2. Formalization

As categories are commonly defined in terms of a taxonomy, tagging systems are defined as a *folksonomy*, which is a portmanteau, combining folk and taxonomy. A formal specification of a folksonomy is for example given by Jäschke et al. in [JMH$^+$07], as shown in Definition 1.

Definition 1 (Folksonomy)**:**
A Folksonomy is a tuple $\mathbb{F} = (U, T, R, Y)$ *where*

- U, T, R *denote finite sets of users, tags and resources.*
- Y *denotes the relation between users, tags and resources, i.e.* $Y \subseteq U \times T \times R$.

A tuple $(u, t, r) \in Y$ *is interpreted as the user* u *tagged resource* r *with tag* t.

Because this thesis has a user-centric view on tagging, i.e. the tag recommendation should take place on a specific user's previous tagging behavior, only the projection to $Y_u \subseteq T \times R$ of the folksonomy relation is used. That means, only the items tagged by a specific user are taken into account together with the tags assigned by this user.

2.1.3. Issues in Tag Systems

Due to their nature, tags imply several issues, which affect applications based on them. The most significant issue consists of items that are not tagged at all. Such items cannot be taken into account by the application, e.g. they do not show up in tag based searches and cannot be recommended on a basis of tags. Users can hardly be forced to assign tags anyway without being annoyed by the application. If the application does not allow tagging of items by foreign users or the addition of tags after an item has been published, such items are not useful at all to tag-based applications. Micro-blogging systems typically suffer from these problems, since they neither allow users to tag foreign users status updates, nor do they provide a possibility to add tags to an entry that has already been published.

Very few occurrences of a certain tag imply a second issue. Such tags can hardly be used for recommendation purposes and deliver very few results when being searched for. The reasons for little usage of a tag are multifarious: The user might have misspelled a tag, the tag might contain a very uncommon abbreviation or synonym, or its context might be too specific to the user and his/her environment.

A third issue is implied by the lack of objective semantics of a tag. As shown in the examples in Section 2.1.1, the tag *apple* is ambiguous and misses unique semantics. It is hardly possible to produce good precision search results for this tag, since it is not clear which meaning the user implies. Searches and recommendations based on the tag *I like this* are even worse if the user's preferences cannot be taken into account. However, the tag recommendation system developed within this thesis only tackles the first and second issue.

2.2. Blogging and Micro-Blogging

Although being popular before the term Web 2.0 was actually defined, *web-logging* (blogging) is commonly considered being a part of this movement, which leads from a content-centric to a user-centric web approach. In a *web-log* (blog), one or more authors publish small to medium sized articles around a certain topic. The articles are presented in form of a single stream, in reverse chronological order.

Micro-blogging is a very new variation of blogging, which limits the size of a blog entry to 140 characters. Users commonly use micro-blogging to provide status updates and commentary. As with many applications in the Web 2.0 area, blogs and micro-blogs typically make use of tagging for the purpose of structuring their content.

2.2.1. Blogging

A blog is a special type of article-based website, which is typically maintained by an individual or a small crew of authors. In contrast to other website types, blogs consist of a continuous stream of articles presented in reverse chronological order. There is typically very little additional structuring in a blog, i.e. no navigation. The published articles typically consist of commentary, announcements, or reports. Blogs dedicated to content types other than text, such as audio blogs (commonly known as *podcasts*) and video or image blogs, also gained popularity in the past years, but are beyond the scope of this thesis. The major aspects of a blog can be summarized as follows:

- The content is generally presented as a stream of articles in reverse chronological order.
- Articles are categorized regarding their content, typically through the method of tagging.
- Readers have the possibility to discuss the content by adding comments to an article.
- The provided content is readily encoded into an XML format to be aggregated by so-called *feed readers*, in addition to the normal HTML representation.

The software for maintaining a weblog usually provides rich editing features, similar to *CMS* (content management systems), as well as media management facilities. It is common for technically experienced users to host blog software in a dedicated web space, or else to have a weblog account at a specialized provider such Google's Blogger.com[2].

2.2.2. Micro-blogging

While weblogs are by now a well-established way for content publishing on the web, which has been adapted by companies and even news press, quite a new movement is the so-called micro-blogging. Where normal blogs may contain articles of arbitrary length, micro-blog entries are strictly limited to a certain maximum length, commonly 140 characters. Furthermore, micro-blogs are integrated as a social-network instead of being hosted independently. Although being invented as standalone social networks, established services such as Facebook[3] and the German StudiVZ[4] adopted the concept rapidly and make micro-blogging more and more popular among mainstream users.

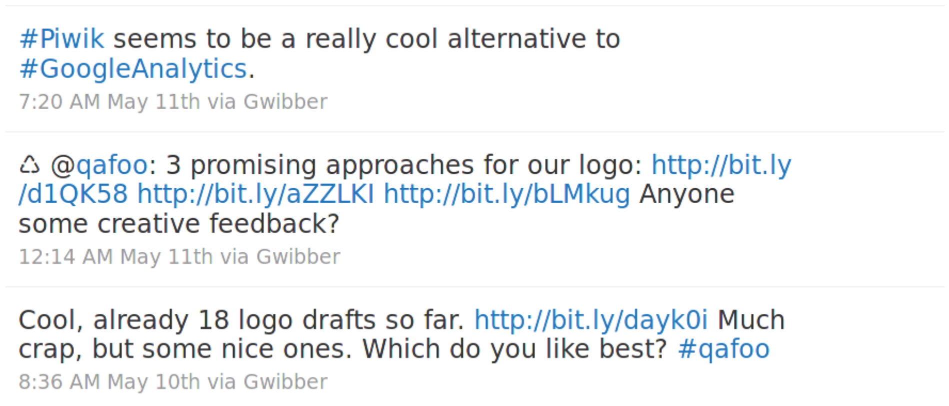
Micro-blogging satisfies the need of more and more people for being constantly online and sharing more and more information about their life. Users mainly post notes about their daily life, commentary and links to articles, images and videos. In addition, micro-blogging platforms are used for discussion. There is typically no dedicated mechanism for commenting on a post. Instead, a new post is used for that purpose, mentioning the name of the user replied to.

A micro-blog post consists only of plain text, without additional markup. Services typically annotate entries automatically with the creation date and optional geo-location information. As with blogging in general, entries are available in reverse chronological order, i.e. the newest post is presented first. A collection of entries is commonly called a *timeline* or micro-blog stream or feed. Figure 2.3 presents an extract of such a time line for a specific user on the Twitter[5] micro-blogging service. Shown are three so-called Tweets in the HTML based web interface of Twitter.

Beside the basic feature of maintaining a micro-blog, services for that purpose offer typical social network features: Users create a profile page, which may contain additional information about the user. Social relations can be reflected by *following* other users on the platform. In contrast to typical social networks, this relation does usually not require confirmation by the related user. Following other people allows a user to gain a merged timeline of their micro-blog entries.

While blog posts are often meant to provide objective value to the reader, e.g. a technical tutorial, the content of micro-blog posts is highly specific and subjective

[2]`http://blogger.com` (2010-05-21)
[3]`http://facebook.com` (2010-05-21)
[4]`http://studivz.net` (2010-05-21)
[5]`http://twitter.com` (2010-07-11)



#Piwik seems to be a really cool alternative to #GoogleAnalytics.
7:20 AM May 11th via Gwibber

♺ @qafoo: 3 promising approaches for our logo: http://bit.ly/d1QK58 http://bit.ly/aZZLKI http://bit.ly/bLMkug Anyone some creative feedback?
12:14 AM May 11th via Gwibber

Cool, already 18 logo drafts so far. http://bit.ly/dayk0i Much crap, but some nice ones. Which do you like best? #qafoo
8:36 AM May 10th via Gwibber

Figure 2.3.: Three Tweets on Twitter.

to the user. Users write on anything affecting their daily life, be it work related, their hobbies, trivial occurrences or instant thoughts and feelings.

While micro-blogging platforms provide a web interface for authoring and reading entries, users typically use special client software to perform these tasks. There is a vast variety of client software available for all operating systems. Due to the success of mobile internet connections, people use micro-blogging wherever there go, through client software on their cellphones or other mobile devices. An example of such a client application is Seesmic[6], which is available for Apple and Android mobile devices, but also as a desktop version for Linux, Windows and others, and as a web application in addition.

It can be noticed that tags in micro-blogging are even more user-centric than in other areas. People do not necessarily use tags similar to their associates. With changing interests, living conditions or breaking news, the tagging behavior of a user might change heavily.

2.2.3. Twitter

Twitter was the first micro-blogging service available and still is the market leader in the segment of dedicated micro-blogging providers. The service faced worldwide rapid growth of its user base during the past three years (see Figure 2.4) and, beside the native English speaking countries, grew especially strong in Brazil, Germany and the Netherlands.

Twitter offers only very rudimentary micro-blogging and social networking facilities. The micro-blogging entries on Twitter cannot be annotated with tags natively, nor can other users be referred. For that reason, Twitters early adopters developed

[6] `http://seesmic.com/` (2010-07-11)

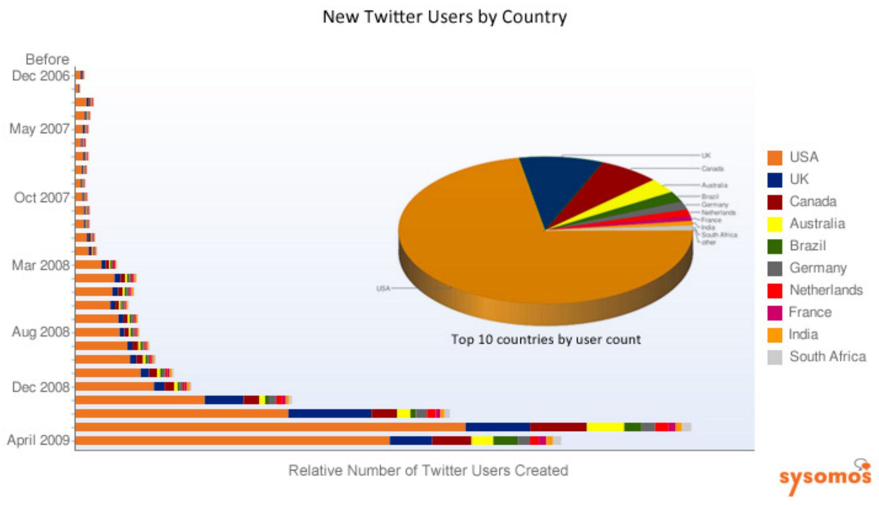


Figure 2.4.: New Twitter users by country[a].

[a]`http://www.sysomos.com/insidetwitter/` (2010-07-11)

a rudimentary markup syntax for these purposes, which encodes both information into the plain text content of an entry:

- Tags are prefixed by the `#` character. Any word of an entry can become a tag that way, if it is already part of the content or appended to it for tagging reasons.
- To refer to another user, her user name is used, prefixed by the `@` character.

Examples for both syntaxes can be seen in Figure 2.3. As can be derived from the highlighting of tags and user references in the shown example, Twitter nowadays supports theses techniques. However, there is still no dedicated data structure or editing facility for it, but Twitter relies on the user invented syntax. It is important to note, that micro-blog entries on Twitter cannot be edited. Therefore, the addition of tags to a once created entry is not possible. Additionally, only the authoring user can assign tags to his/her post, there is no possibility for foreign users to assign tags.

A data set extracted from Twitter is used for experiments throughout this thesis. The data extraction process is explained in further detail in Appendix A. Furthermore this appendix provides a basic statistical analysis of the data set.

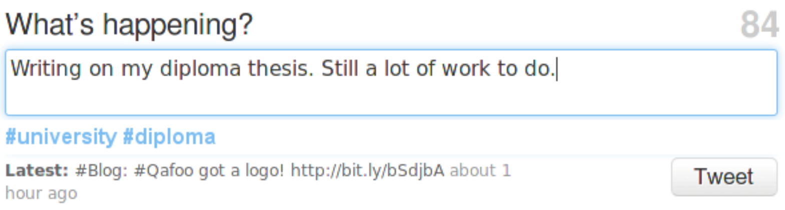
2.3. Objectives

The global goal of this thesis is to develop a tag recommendation service for micro-blogging systems. Section 2.1 and Section 2.2 already introduced the concepts of tagging and micro-blogging. In the following, the objectives of this thesis are examined in further detail.

The next Section 2.3.1 presents a use case to show how a real-life implementation of the desired tag recommendation system could work. After that, the actual goal is verbalized and formalized in Section 2.3.2. Finally, Section 2.3.3 summarizes related research.

2.3.1. Use case

A common approach for mitigating the problems involved with tagging systems (see Section 2.1.3) is to assist the user with the process of tagging. To achieve this, the *GUI* (Graphical User Interface) used for tagging is commonly enhanced by a mechanism to select recommended tags. In the web environment, such an enhancement can be achieved through a list of tags, where a tag can be added to the created content item by clicking it.


What's happening? 84

Writing on my diploma thesis. Still a lot of work to do.

#university #diploma

Latest: #Blog: #Qafoo got a logo! http://bit.ly/bSdjbA about 1 hour ago

Figure 2.5.: Mock up: Twitter GUI with tag recommendation. Below the text box for posting new entries, two recommended tags are displayed. If any of these is clicked, it would be added to the entry.

Figure 2.5 shows a mock up of the Twitter GUI for creating a new micro-blogging entries, enhanced by such a mechanism. Below the original text box, two tags are recommended for the status update just being created. In this scenario, the user could click on any of these tags, so as to apply it to the post. For the first recommendation, a word already contained in the text would become a tag. The other tag would be appended at the end of the text.

Ideally, a realization of a tag recommendation system would neither depend on a particular micro-blogging client nor on a specific micro-blogging service. Therefore, implementation as a web service would be appropriate. Client software could then integrate tag recommendation without implementing the necessary algorithms on its own. Furthermore, the recommendation mechanism itself would be available to any micro-blogging user, independently from the actual service he/she uses. Figure

2.6 visualizes the process of micro-blogging involving a tag recommendation web service.

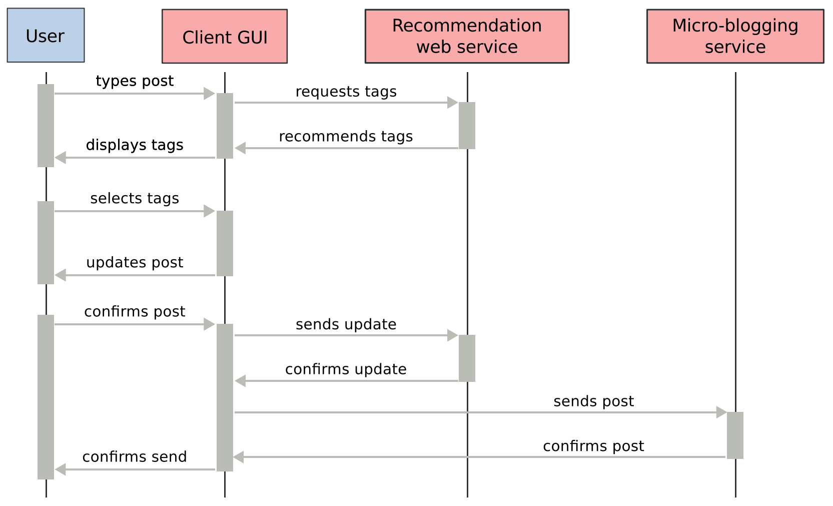


Figure 2.6.: Tag recommendation with a web service. The user just interacts with his/her favorite micro-blogging client. This software needs to integrate with the tag recommendation service in addition to the already available micro-blogging service.

The user only interacts with his/her micro-blogging client of choice. This can either be the HTML interface of a micro-blogging service, or a third party application. As the user finishes typing a new entry, the client first submits it to the tag recommendation service. The service returns a set of recommended tags for the post in question. Now the user may select an arbitrary number of these recommended tags or add further ones manually. The client software is responsible for integrating selected tags into the post. Once finished, the client submits the entry to the micro-blogging service. In addition to that, the entry must be submitted to the tag recommendation service again. This way, the service can adjust to changes in the user's tagging behavior and try to optimize recommendations for subsequent posts.

2.3.2. Goal

From the preliminary considerations of tagging problems and the use case described in Section 2.3.1, the goal of this thesis can be derived:

Develop a user-centric system for tag recommendation in micro-blogging environments, which can be used in a web service for integration with arbitrary micro-blogging clients and services.

This goal basically implies the task of classification, which can be defined as shown in Definition 3. For convenience reasons, Definition 2 first gives the definition of an example, also denoted as a data point, which is then used in subsequent definitions.

Definition 2 (Example)**:**
An example $e := (\vec{x}, l)$ is a tuple consisting of a feature vector $\vec{x} \in \mathcal{X}$, with $\mathcal{X} := \mathbb{R}^d$ being the feature space, and a label $l \in L$, where L is a finite set of labels. A set of n examples $E := \{e_1, \ldots, e_n\}$, is called an example set.

The possible values for a label l, i.e. the elements of L, are commonly denoted as *class*es. In different cases throughout this thesis, only the feature vector of an example needs to be referred to. Depending on the specific case $\vec{x}$ is used, if the the full example and its feature vector need to be distinguished, or e is meant to denote the feature vector, if its clear from the context.

Definition 3 (Classification)**:**
Given an example set $E = \{(\vec{x}_1, l_1), \ldots, (\vec{x}_n, l_n)\}$. A classification function

$$\mathrm{h} : \mathcal{X} \to L$$

is to be learned, such that $\mathrm{h}(\vec{x}_i) = l_i$, for all $(\vec{x}_i, l_i) \in E$.

A classifier h is meant to assign the correct label to a new, unseen example, for which the true label is unknown. The procedure of training a classification function is also commonly referred to as supervised learning. This basic task is refined further in the following, based on the requirements that apply to the tag recommendation system to be developed throughout this thesis.

Multi-label classification

The targeted system must be able to predict tags for a new micro-blogging entry by a specific user. As defined by the classification task, this prediction is to be learned on the basis of existing examples, i.e. the micro-blogging entries written in the past.

The standard classification task only defines a single label per example, where the concept of tagging allows an example to have an arbitrary number of labels (i.e.

tags) assigned. Such an environment is commonly defined in terms of a multi-label classification task, for example in [TK07]. In the scope of this thesis, multi-label classification is mainly defined by varying the specification of an example to a *multi-label example*, as specified in Definition 4.

Definition 4 (Multi-label example)**:**
A multi-label example $e = (\vec{x}, M)$ *is a tuple consisting of a feature vector* $\vec{x} \in \mathcal{X}$ *and a set of true labels* $M \subseteq L$*, where* L *is a finite set of labels.*

Again, the components of a multi-label example can be referred dedicatedly by the same notation as described for single-label examples (Definition 2). The multi-label classification task can then be defined analogous to the standard classification task. The formal specification is given in Definition 5.

Definition 5 (Multi-label classification)**:**
Given a set of multi-label examples $E = \{(\vec{x}_1, M_1), \dots, (\vec{x}_n, M_n)\}$*. A classification function*

$$\mathrm{h} : \mathcal{X} \rightarrow \mathcal{P}(L)$$

is to be learned, which maps the feature vector of each example to its true set of labels, i.e. $\mathrm{h}(\vec{x}_i) = M_i$ *for all* $(\vec{x}_i, M_i) \in E$*.*

While the task of multi-label classification has become more and more common in the past years, mainly due to tagging systems, literature in this area is sparse. Section 3.2 selects a technique for tackling the multi-label classification problem through a transformation into a semantically equivalent single-label classification task.

Anytime

Micro-blogging entries are produced as a continuous stream of examples. In order to be able to predict tags only the micro-blogging entries previously written by a specific user are available as training data. If a user only started with micro-blogging only recently, this training data is expected to be not sufficient to train a high-quality classifier. Furthermore, it is expected that tagging behavior of a user changes over time, which commonly denoted as a *concept drift* (see e.g. [WK96]). Both circumstances require to update the tag prediction classifier over time.

The desired behavior can be described in terms of an *anytime* algorithm, which is capable of performing a prediction, although its model is not yet finished. It is possible to achieve such a behavior by storing all micro-blogging entries of a specific user in a data base on the web service and to re-train the classification model in *batch learning* manner on the full collected training data. However, this would consume quite some computation time on every update and would not honor the expected change in tagging behavior.

It is therefore desired to use a *stream* data model for recurrent updates of the classification model. Stream models do not interpret data as a single large batch, but as a constantly updating stream of examples. Stream models are a recent topic of research and are generally used in cases where the size and velocity of data exceeds storage and computation facilities. It has to be noted, that neither of these conditions is fulfilled in the environment of this thesis: Although users might produce several micro-blogging entries per day, this update frequency is still low compared to typical stream problem environments as, e.g. tackled in [str02] and [BJC$^+$04].

A goal of this thesis is therefore the evaluation, if stream based classification is a possible way to perform tag prediction in micro-blogging services. The major objectives here are to provide a classification model on basis of very few data, so that tag prediction can take place even with few user data available and to update the classification model over time, as more data is produced. Furthermore, it should be attempted to check for changes in the tagging behavior of a user, in order to adjust the classifier correspondingly.

In order to consider consider examples in form of a stream, Definition 6 formally specifies the data stream model utilized in this thesis.

Definition 6 (Data stream):
A data stream is a finite or infinite set of examples $S = \{e_1, \ldots, e_n, \ldots\}$, which are sorted in an ascending, typically temporal, order $\prec_S$, such that

$$\forall e_i, e_j \in S : i < j \Rightarrow e_i \prec_S e_j$$

holds.

Modelling of the typical constraints of data streams, e.g. that an algorithm must not access data randomly but only has access to a certain portion of data at a given point in time, takes place in Chapter 5, where the evaluation of stream classification is tackled.

2.3.3. Related work

There has been quite some research in the field of tag recommendation systems in the past years. This section attempts to give an overview on some popular approaches, but is not intended to be exhaustive. It can generally be noted that research so far has concentrated on the recommendation of tags through the social aspect of tagging systems, i.e. tags for an information item are predicted through tags that other users had already assigned to this item. This approach, however, is generally not feasible for the environment of micro-blogging, since a newly created entry cannot have been tagged already by other users.

The ECML PKDD 2008 discovery challenge featured a tag recommendation task[7]. The goal was to develop a system for tag prediction for the BibSonomy data set. Three submissions were received, but none of them achieved an F1 score better than 0.2. Most of the system relied on completely heuristic approaches while only the paper by Katakis et al. involved a formal learning approach [KTV08]. They used the Binary Relevance classifier as the basis, which internally uses a Bayes learner. ECML PKDD 2009 featured two similar challenges without significant improvements regarding the F1 score results of the proposed approaches.

In [SOHB07] Sood et al. introduce a case-based reasoning approach for tag recommendation in blog systems. They basically index all contents of a blog using a search engine like Lucene[8]. For a newly posted blog entry, they retrieve related entries from the search engine and predict the most common tags among them. While this approach is described to be fast, it suffers from two problems regarding the objectives of this thesis: Firstly, the search index does not automatically adjust to changes in tagging behavior. Secondly, a search engine is expected to yield overfitted results on short text as they occur in the environment of micro-blogging.

Heymann et al. concentrate on social tag prediction in [HRGM08] and attempt to asses rules on when a tag is predictable or not. However, since they concentrate on predicting tags on basis of the full community database, their prediction approach is not feasible for the underlying case, because for a new micro-blogging entry there are no tags by other users available. Similarly, this applies to the work of Jäschke et al. [JMH$^+$07] and related work, since they base tag prediction on the full folksonomy relation.

[7] `http://www.kde.cs.uni-kassel.de/ws/rsdc08/` (2010-05-24)
[8] `http://lucene.apache.org` (2010-07-11)

Chapter 3

Data Preparation

As has already been noted in Section 2.3.2, the problem of tag prediction in micro-blogging is a multi-label classification task: An arbitrary number of labels, i.e. tags, can be assigned to each example. The target of this thesis is a system which can recommend a set of tags for a new, unlabeled micro-blogging entry, based on a users previous tagging behavior.

Although the multi-label environment becomes more and more popular, research literature in this area is still sparse. The most common approach is to transform the multi-label problem into a single-label variant. Such a transformation typically involves at least one of two steps: Firstly, the multi-label data set has to be prepared, e.g. by decomposing examples with multiple labels. Secondly, one or more single-label classifiers are adjusted or combined, e.g. by learning multiple binary classifiers in a one-against-all manner.

In this chapter, several such transformation approaches are described and evaluated for their feasibility, to be used in the environment of tag prediction in micro-blogging systems.

The next Section 3.1 derives a representation for micro-blogging entries and tags, which is feasible for the application of machine learning and data mining techniques. The term vector model is selected here, while vector weights are defined by the binary occurrence measure.

After that, Section 3.2 discusses different transformation methods for the multi-label problem implied by tagging. The power set transformation method is selected as the most feasible approach and experiment results are presented in order to show its feasibility. However, this technique still does not yield outstanding classification quality using several different classification methods (kNN, Naive Bayes and SVM).

Therefore, a technique to reduce the number of labels generated by the power set method is developed in Section 3.3.3. It is predicted, that this step can raise the classification quality in general. A detailed evaluation of the developed approach, together with details on the utilized classification methods, is presented in the next Chapter 4. In addition, a custom classification method is developed and evaluated there.

3.1. Data Representation

To apply methods of machine learning and data mining to micro-blogging entries, these have to be brought into a feasible representation: A vector representation is desired. The discipline of *text mining* deals with the problem of representing textual information in an appropriate way. Beside the representation for blog entries, a representation for tags assigned to a blog entry is required too.

3.1.1. Vector Generation

The basic task to be solved is to generate vector representations from a set of text documents. Such a set of documents is commonly referred to as the *document corpus* or *text corpus.*

Definition 7 (Document corpus)**:**
A document corpus D is a set of text documents $D := \{d_0, \dots, d_i\}$. Documents are reprsented in terms of the bag of words model as a set of terms, i.e. $d := \{t_0, \dots, t_n\}$.

In the underlying case, the document corpus consists of the micro-blogging entries by a specific user. Note that punctuation is omitted in the bag of words model, which is e.g. mentioned in [Joa97a]. On this basis, two representations for text content are discussed in following: The *vector space model* and the encoding through *n-gram vectors.* After that, a feasible representation is chosen, which is used throughout this thesis (Section 3.1.1).

Vector Space Model

The vector space model was developed by Salton et al. in [SWY75]. It defines the encoding of a text document based on its contained terms, i.e. words. In terms of machine learning, each word becomes a feature attribute. In order to generate a term vector, each document is interpreted as a bag of words, ignoring the semantic structure and punctuation of the document.

Definition 8 (Vector space model)**:**
Given a text corpus D with documents $d \in D$ represented as bags of words. The set of all terms $T = \bigcup_{d \in D} d$ is called the vocabulary of the corpus. By bringing an arbitrary order to the vocabulary, it is used to define an Euclidean feature space $\mathcal{X} = \mathbb{R}^{|T|}$, where each dimension corresponds to a term $t \in T$. A document d_j is encoded as a vector in this space as

$$\vec{d_j} := (t_{0,j}, \dots, t_{k,j})$$

where $t_{l,j}$ is non-zero, if $t_l \in d_j$.

Different approaches exist to determine which terms are taken into the vocabulary T and how the vector values are calculated. The most common ones are to selecting all words from all documents or to choose specially significant index terms. In order to determine the term vector weights, one of the following three measures is commonly chosen (e.g. [HMS02]):

Binary occurrence is the simplest weight, where a dimension is set to 1 if the corresponding term occurs in the document, 0 is set otherwise

Term frequency weights each dimension with the number of occurrences of the corresponding term in the document

TF/IDF represents a global significance measure, presented in Definition 9.

Definition 9 (TF/IDF measure)**:**
For a term t_l in a document $d_j \in D$, the TF/IDF measure is defined as

$$t_{l,j} := tf_{l,j} \cdot log(\frac{n}{df_l})$$

where $tf_{l,j}$ denotes the number of occurrences of term t_l in document d_j (term frequency) and df_l is the number of documents in which t_l occurs (document frequency).

The mathematical term $log(\frac{n}{df_l})$ is also known as the *inverse document frequency*. The TF/IDF measure is typically chosen in information retrieval scenarios.

n-Gram Vectors

A second solution for encoding plain text documents into a vector representation is the use of n-grams. This approach is typically utilized in natural speech processing, e.g. [MM04]. n-grams can be created from sequences of arbitrary items, e.g. micro-blogging entries as sequences of characters. A n-gram is a sub-sequence of length n from such a sequence. For the creation of a vector representation, each n-gram from a document corpus is taken as a dimension of the vector space (formalized in Definition 10).

Definition 10 (n-gram vector encoding)**:**
Let D be a text corpus. Each document $d = x_0 \ldots x_k$ consists of a sequence of characters x_i from a finite alphabet K. An n-gram is defined as a sub-sequence of consecutive characters $g_k^n := x_k x_{k+1} \ldots x_{k+n}$, where n is the cardinality of the n-gram and k denotes its starting position.

To encode a document corpus through n-grams, each document $d \in D$ is represented as its set of n-grams $\mathrm{g}(d) := \{g_0^n, g_1^n, \ldots\}$. In order to create a vector *space, the set of all possible n-grams* $G := \bigcup_{d \in D} \mathrm{g}(d)$ *is brought into an arbitrary order. Each n-gram* $g \in G$ *refers to a dimension in* $\mathcal{X} = \mathbb{R}^{|G|}$ *and the vector representation of a document* $d_j \in D$ *is defined as*

$$\vec{d_j} := (g_{0,j}, \ldots, g_{k,j})$$

where $g_{i,j}$ *becomes non-zero, if* $g_i \in \mathrm{g}(d_j)$.

The n-gram encoding can be interpreted as sliding a window of length n over a sequence of items. Each step of the window creates an n-gram.

Representation Selection

The vector space model has been chosen to encode micro-blogging entries and their sets of tags into vector representations. Since tags are words or abbreviations, which occur in the plain text content of a micro-blogging entry, any term has a high potential for being chosen as a tag. Choosing the n-gram representation would eliminate this potential correlation. Furthermore, the extraction of tags is a simple task during word vector creation.

The set of tags assigned to an example, M, is also represented as a term vector on the tags occurring in the document corpus. A micro-blogging entry is therefore represented as a tuple $e := (\vec{x}, \vec{m})$, in the notion of a multi-label example. $\vec{x}$ is the term vector of the full micro-blogging entry and $\vec{m}$ is the term vector of the tags contained in the entry. In following, the bag of words representation x, respectively m, and the corresponding vector representation $\vec{x}$ and $\vec{m}$ are used interchangeably.

To determine the vector weights, the binary occurrence measure has been selected. The TF/IDF measure is not suitable in the underlying case since the developed tag prediction technique should be based on a data stream model (see Chapter 5). TF/IDF cannot be calculated reliably in this scenario since it involves the global occurrence count of a term. Furthermore, the term frequency measure also appears to be unfeasible for the underlying case: It seems unlikely that a term occurs twice or more times within a micro-blogging entry with its limited length of maximum 140 characters. Stop words might be an exception here, but it is generally not intended to weight these higher due to their more frequent occurrence.

3.1.2. Pre-Processing

In order to create term vectors, micro-blogging entries are tokenized. The technique of tokenizing originates from the discipline of compiler construction, for which extensive information can be found in [ASU86]. The developed system uses a simple tokenizing step, splitting the document text at any sequence of non-alphanumeric characters, except for the `#` character, which denotes a tag. Since any term in a micro-blogging entry can become a tag, tags are not only added to the corresponding vector $\vec{m}$, but also to the feature vector $\vec{x}$.

One crucial issue in text mining is the large number of dimensions in feature implicit in the processing of text and the resulting problems, commonly known as the *curse of high dimensionality.* Experiments on example data extracted from the Twitter web service have shown that more than 8000 different terms can be contained in the vocabulary of a user. While this number is already low compared to other text mining scenarios (e.g. [BEX02]), the number of dimensions can be significantly reduced by different pre-processing steps. The following sections give an overview of the pre-processing steps chosen for this thesis. The selected pre-processing steps are commonly used in the area of text mining, e.g. in [TBHG00], [YP97] and [DC00].

Basic statistical analysis of the term-vectors generated from micro-blogging entries can be found in Appendix A.

Lowercase Conversion

In western languages, the same word can start with a capital letter or not, depending on its position in a sentence. In order to avoid separation of semantically equivalent words, all terms are converted to lowercase as a pre-processing step. Experiments have shown that the number of unique words can be reduced by up to 40% this way.

Stemming

Stemming denotes the process of reducing an inflected word to its stem, the root form. The porter stemming algorithm [Por80] is one of the most commonly used stemming algorithms for English words. It is therefore used for pre-processing tokens generated from micro-blog entries. A further reduction by up to 10% has been achieved using this technique.

3.2. Multi Label Transformation

A fundamental problem involved in the prediction of tags is the multi-label nature of the task, as has been already been noted in Section 2.3.2. Tagging systems allow the user to assign an arbitrary number of labels to each item. Experiments have shown that the use of more than one tag per micro-blogging entry is not rare: On average, $18.92\% \pm 14.57\%$ of tagged micro-blogging entries have multiple tags assigned (details in Appendix A). Therefore, the desired recommendation system needs to be capable of learning such multi-label associations and to recommend sets of multiple labels for new examples.

While a vast variety of classification algorithms exist for the single-label / multi-class environment, literature on multi-label classification is sparse. Tsoumakas and Katakis summarize common approaches to the multi-label problem in [TK07]. They basically conclude that the most common approach is a transformation of the problem into a single-label classification task. Even most approaches for adjusting a single-label classification algorithm to the multi-label environment inherently perform such a transformation.

3.2.1. Transformation Methods

In the scope of this thesis, different multi-label to single-label problem transformation methods have been evaluated in respect to their feasibility for the underlying problem. A summary of this evaluation is presented in the following. For illustration purposes, the sample multi-label data set visualized in Table 3.1 is used throughout the evaluation. The data set consists of four examples, which have up to three labels assigned. For simplicity, the feature vectors of the examples have been left out.

ID	#php	#database	#university
1	X	X	
2	X		X
3	X		
4	X	X	X

Table 3.1.: Example multi-label data set.

Two requirements are leading for the selection of a feasible technique: Firstly, the transformation should yield a classification model which is feasible to predict a set of tags for micro-blogging entries at all. Secondly, since the desired classification approach is expected to be used in a multi-user environment and classification must not take long, the multi-label transformation should not yield extensive overhead in computation time.

Information Loss Methods

Mhe simplest approach for transformation of a multi-label dataset into a single-label one is the removal of multi-label information. Two variants of this approach are possible:

- Removal of all examples which have more than one label assigned.

- For each multi-label example, discard all labels except for one.

These techniques might be feasible in cases where multi-label examples are outliers and most data in the analyzed data set is only of single-label nature. However, this situation is not given in the underlying case.

Power Set Method

The *power set method* attempts to convert the each label set into a new distinct label, which means a transformation from a multi-label into a single-label data set. This transformation can be formalized as a classification function

$$\begin{aligned} \text{h'}: &\quad \mathcal{X} \to L' \\ \text{h'}(\vec{x}) := &\quad \textstyle\bigwedge_{l \in \text{h}(\vec{x})} l \end{aligned}$$

For example, the label set $\{a, b\}$ is replaced by the new label $a \wedge b$. The transformed label set is therefore defined as in terms of the new label set

$$L' := \{\bigwedge_{l \in M} l | M \in \mathcal{P}(L)\}$$

The transformed example data set is shown in Table 3.2.

Note that the definition of the power set method given in this thesis varies slightly from the one presented in [TK07], for notation reasons: Tsoumakas et al. define the classification function as

$$\text{h''}: \quad \mathcal{X} \to \mathcal{P}(L)$$

which conflicts with the general definition of multi-label classification given in Definition 5 of this thesis.

ID	#php ∧ #database	#php ∧ #university	#php	#php ∧ #database ∧ #university
1	X			
2		X		
3			X	
4				X

Table 3.2.: Multi-label data set transformed using the power set method.

Boutell et al. use the power set transformation method in [BLSB04] for multi-label classification of scene data. In [DTMV05], Diplaris et al. seized the approach while evaluating different classification algorithms for motive-based protein classification. McCallum used a similar approach in terms of a mixture model to represent multiple classes assigned to an example in [McC99].

A fundamental advantage of the power set method, compared to other approaches described in this section, is the fact that only a single classifier needs to be trained. Choosing a well performing classifier, this approach could satisfy the requirement of only moderate overhead in computation time. The disadvantage here is, that the selected classification algorithm needs to cope with a large number of classes. Experiments have shown that the number of unique tags per user highly varies (average 190.6 $\pm$ 190.1), but numbers of 300 and up are not uncommon. It is assumed, that the full power set in such scenarios would be unmanageable. A statistical analysis of tagging behavior in micro-blogging systems is presented in Appendix A.

One-Against-All Method

Most commonly used, according to [TK07], is the method of one-against-all classification. Instead of transforming the data set directly, multiple classifiers are learned: A binary classifier of the form

$$\begin{aligned} \mathrm{h}_l : &\quad \mathcal{X} \rightarrow \{-1, 1\} \\ \mathrm{h}_l(\vec{x}) := &\quad \begin{cases} 1, & \text{if } l \in \mathrm{h}(\vec{x}) \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

for each possible label. Each such classifier is then trained with the examples that have the corresponding label assigned as positive training examples and all other examples as negative examples. These classifiers are then combined in a classifier of the form

$$\begin{aligned} \mathrm{h'} : &\quad \mathcal{X} \rightarrow L \\ \mathrm{h'}(\vec{x}) := &\quad \textstyle\bigcup_{l \in L} \{l\} : \mathrm{h}_l(\vec{x}) = 1 \end{aligned}$$

The one-against-all approach is used, for example, by Goncalves et al. to classify Portuguese juridical documents by concept in [GQ03]. Each document can belong to multiple concepts there. Lauser and Hotho utilize the method for subject indexing, i.e. describing by the subjects they cover, of documents maintained by the *Food and Agriculture Organization* (FAO) of the *United Nations* (UN) [LH03]. In [LO03], Li et al. explore the classification of music by emotions, where every song might reflect multiple different moods. All three papers use SVM for the binary classifiers (see Section 4.1.3 for details on SVM).

An approach quite similar to the one-against-all method is used in ML-kNN [ZZ07] by Zhang et al. They examine the probability for each individual label for a new example, based on its k nearest neighbors. Godbole and Sarawagi present an enhancement to the usage of one-against-all with SVM [GS04]: To take into account potential inter-dependencies between labels, they first train the typical binary classifiers and then append their prediction result to the training vectors. On that basis, new classifiers are trained. This technique is commonly known as stacking.

While the one-against-all method subjectively promises more accurate results than e.g. the power set method, the approach appears to be quite time consuming. Experiments identified 190 unique labels on average, resulting in 190 SVM classifiers to be trained. Since SVM training works in (n^3), this combination appears to be unfeasible. Furthermore, requesting results from 190 binary classifiers of any kind can hardly be considered feasible in terms of classification time.

Example Decomposing Method

Tsoumakas and Katakis also mention a not yet explored way of transformation: Decompose the original example set into a new one, where each example occurs as many times as it originally had labels assigned, each time with only a single label. Table 3.3 visualizes the resulting example set. On this basis, they want to learn a single coverage-based classifier, which outputs a probability distribution over all possible labels for each new example.

The fundamental problem with this approach is the selection of a feasible label-set on basis of its prediction result. A static threshold, e.g. a probability larger than 0.5, could be a potential solution. However, this value might differ for each user and a user-based setting is not feasible for the approach desired in this thesis.

ID	label
1	#php
1	#database
2	#php
2	#university
3	#php
	...

Table 3.3.: Example decomposition method.

ID	label	class
1	#php	1
1	#database	1
1	#university	-1
2	#php	1
2	#database	-1
2	#university	-1
	...	

Table 3.4.: Alternative example decomposing method.

Schapire and Singer propose a boosting based text categorization approach in [SS00], which utilizes a similar decomposition technique: They generate $|L|$ new examples for each original example, one in combination with every possible label, which is added to the decomposed examples as a normal feature. They then add a new label attribute, which becomes 1, if the corresponding original label was originally assigned and -1 otherwise. A resulting example set is visualizes in Table 3.4.

While this approach also decomposes the original example set, Shapire and Singer do not predict the original labels directly, but perform classification on an example label combination. They apply Adaboost to weak classifiers of the form

$$\mathrm{h} : \mathcal{X} \times L \rightarrow \mathbb{R}$$

to the transformed example set. Two different approaches for the actual classification are described in [SS00]: In the first variant, *Adaboost.MH*, if a weak classifier returns a positive value for a new example x and a label in question l, this label is taken into the predicted label set. In the second variant, *Adaboost.MR*, the output of the weak classifiers is considered a ranking. In this case, the actual labels for prediction need to be select from the ranking.

While the Adaboost.MH approach ships around the label selection problem, both approaches still require as many classifications as there exist unique labels to be performed, in order to perform multi-label classification.

3.2.2. Evaluation

None of the evaluated transformation methods convinced immediately as the ideal approach to tackle the multi-label problem. Methods which simply discard multi-label information are not considered feasible due to the degree of multi-label data in a tagging system. The power set method (Section 3.2.1) occurs to be more feasible than others, since only a single classifier is to be trained, which keeps the overhead in computation time moderate. However, the potentially large number of generated labels could make this approach unfeasible. Classification on basis of the one-against-all transformation is generally expected to produce better classification results, but the pure number of classifiers to be trained produces a large number of computational overhead. The example decomposition method depends on a coverage based classifier, which inheres similar problems as a ranking algorithm: Based on the generated distribution, the actually recommended label set must be selected using a heuristic. Therefore, this approach is also not considered further.

Based on these considerations, the power set method was chosen as the transformation technique for further evaluation. In addition to that, some experiments on the basis of the one-against-all method have been performed, with one expected result: The training and classification times were unacceptably slow. Therefore, these experiments have been abolished again and the focus was fully put on the power set method.

As classifiers, kNN, Naive Bayes and SVM have been chosen. Details on these and on experiment setups, as well as further evaluation of classification performance can be found in Section 4.1. Information on the data set used for experiments can be found in Appendix A.

An initial experiment revealed that the number of labels generated by the power set method is indeed large: On average, 219.2 ± 216.991 labels where generated. As the standard deviation indicates, the number of labels varies widely, depending on the tagging behavior of the user. Two tested users tagged actively, which resulted in the expected large number of up to 537 labels. Another two users did not participate much in tagging: These data sets only generated 65 power set labels. A single user tagged very little, which results in only 4 new labels.

In general, all classifiers produced a poor accuracy. Only for one user sample, an accuracy above 0.5 was reached by the Bayes classifier. As expected, the number of labels produced by the power set transformation seems to influence the classification quality. Both data sets with a large number of labels yield accuracies of maximum 0.25 and lower, while the Naive Bayes classifier produces an accuracy of more than 0.5 on a data set with few labels. This classifier yields the overall best results. While the number of labels appears not to be the only factor with influence on the classification performance, it still seems to be vital force here.

The common impression that more examples generally yield better classification results is not confirmed by experiments here. Even though the example set *User 1* provides midfield numbers of labeled examples and generated labels, classification accuracy is not adequate for kNN and SVM. Figure 3.1 visualizes the classification quality.

The absolute number of power set labels per user are given as pale blue bars in the background (related to the right value axis). The absolute number of labeled examples per example set is presented in the green background bars. To detect classification quality, the accuracy measure (see Section 4.3.1) has been used, protracted on the left value axis. The visualized values are also shown in Table 3.5. Here, the row *# labels* denotes the number power set labels and *# examples* presents the number of labeles examples. The remaining three rows present the classification accuracy of the selected classifiers.

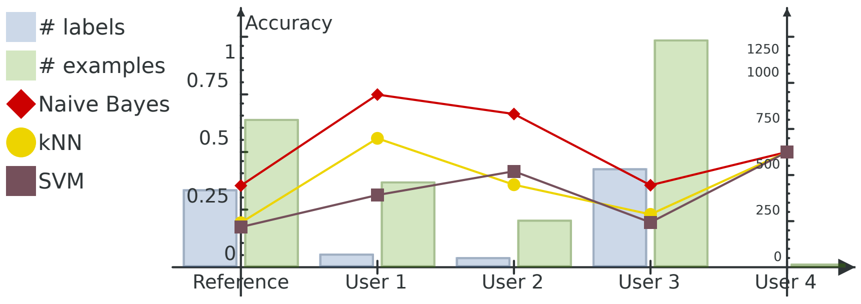


Figure 3.1.: Classification on basis of the power set transformation.

Furthermore, it can be noted, that each classifier reacts differently on the varying number of labels and labeled examples. While the Naive Bayes classifier produced overall the best results, all classifiers vary heavily in their quality.

	Reference	User 1	User 2	User 3	User 4	Avg.
# labels	423	74	55	537	4	218.60 ± 217.66
# examples	805	466	258	1236	20	557.00 ± 426.29
Naive Bayes	0.35	0.75	0.66	0.36	0.50	0.52 ± 0.16
kNN	0.19	0.56	0.36	0.23	0.50	0.37 ± 0.14
SVM	0.17	0.31	0.42	0.19	0.50	0.32 ± 0.13

Table 3.5.: Classification accuracy on power set labels.

3.3. Label Reduction with FTC

In the previous Section 3.2, different transformation approaches for tackling the multi-label-problem have been discussed and evaluated. This evaluation resulted in the power set method (Section 3.2.1) being chosen as the most promising approach. However, classification performance in combination with this approach is still poor. It has been suspected, that this is a result of the large number of labels generated by the power set transformation. In order to reduce this number, this section discusses an approach for label-reduction on basis of *frequent term-based clustering*.

The developed approach will replace miltiple distinct label set in the multi-label example set with a single label set. For example, the approach could replace the label sets $\{a, b\}$, $\{a, b, c\}$ and $\{a, b, d\}$ by $\{a, b\}$. While this results in some information loss, it still preserves the multi-label nature and reduces the number of power set labels.

The label-reduction approach developed relies on *Frequent term-based text clustering* (FTC) by Beil et al. This is a clustering technique specialized on text documents and based on the concept of frequent item set mining. The algorithm, proposed in [BEX02], yields a set of clusters and per cluster so called *cluster description*. This cluster description consists of a frequent term set, which is common to the documents in the cluster.

In order to perform label-reduction, the FTC algorithm is applied to the label sets of a multi-label example set. After that, the label sets in cluster are replaced by the clusters description. It is expected that applying the power set transformation hereafter will produce significantly less labels. Furthermore, it is assumed that classification methods will cope better with the resulting example set.

3.3.1. Frequent Item Set Mining

The finding of frequent item sets in *transactional databases*, so called *frequent item set mining* (FIMI), is a pattern recognition task, which often occurs in the area of data mining. The goal is to find sets of items, which frequently occur together. A typical scenario for FIMI is the detection of interesting associations in the market basket analysis. Using the technique, one can detect which articles were commonly bought in combination.

The basis for frequent item set mining is a finite set of items $\mathbb{I} := \{i_0, \ldots, i_n\}$. This basic set facilitates the creation of $2^{|\mathbb{I}|}$ distinct *item sets*.

A transactional data base stores the relation

$$D : T \to \mathcal{P}(\mathbb{I})$$

which assigns an item set $I \subseteq \mathbb{I}$ to each transaction ID $t \in T$. A tuple (t, I) is called a transaction. An example for a transactional database with 4 transactions is shown in Table 3.6. The leftmost column determines the transaction ID, i.e. $T = \{1, \ldots, 4\}$, while the columns a to e refer to the items, i.e. $\mathbb{I} = \{a, \ldots, e\}$. The value 1 in any of the data cells indicates that the specific item occurs in the item set of the specific transaction.

T	a	b	c	d	e
1	1	0	1	0	0
2	0	1	1	1	0
3	1	0	1	1	1
4	1	1	1	0	0

Table 3.6.: Example of a transactional database.

In the discipline of FIMI, the goal is to detect a set of item sets $\mathcal{I}(D, f) := \{I_1, \ldots, I_m\}$ which occur more often in D than a given minimum frequency f, i.e.

$$\forall I \in \mathcal{I}(D, f) : I \subseteq \mathbb{I} \wedge |\{d \in D | I \subseteq d\}| > f$$

In case D and f are clear from the context or negligible, the set of frequent item sets is simply denoted as $\mathcal{I}$. An item set I with $|I| = k$ is denoted as a k-item set. It is common to refer to all k-item sets $\mathcal{I}^k := \{I | I \in \mathcal{I} \wedge |I| = k$.

Given a transaction $(t_x, I_x) \in D$. If for an item set $I_y \subseteq I_x$ holds, purportedly I_y is contained in the transaction or I_y *covers the transaction.* Sequentially, all transactions covered by a specific item set are expressed by the cover() function [BEX02].

Definition 11 (Item set cover)**:**
The cover of an item set I on a transactional database D is defined as

$$\text{cover}(I, D) := \{(t_x, I_x) | I \subseteq I_x\}$$

The relative frequency of an item set I in a transactional database D is referred to as the *support* of the item set.

Definition 12 (Item set support)**:**
The support of an item set I over a transactional database D is defined as its relative frequency.

$$\text{support}(I, D) := \frac{|\text{cover}(I, D)|}{|D|}$$

The *support* is a rudimentary measure for the interestingness of an item set. It reflects the empirial probability for a transaction to contain a specific item set. It is common use in the field of frequent item set mining to define a minimum support threshold $sup_{min} \in [0, 1]$ instead of the minimum absolute frequency. If $\text{support}(I) > sup_{min}$ holds, I is said to be *frequent*.

The most well-known algorithms for mining of frequent item sets are the Apriori algorithm [AS94] and FTGrowth [HPYM04]. Apriori is based on a candidate generation approach, starting from 1 item sets, and utilizes the fundamental monotonicity property of frequent item sets: If an item set is frequent, all its subsets are frequent too. The other way around, if an item set is not frequent, none of its supersets can be frequent. This property creates a lattice on the search space of item sets, which is visualized in Figure 3.2.

Although Apriori and FPGrowth both have a worst case runtime of $O(2^{|D|})$, empirical analysis has shown, that FPGrowth is about one magnitude faster than Apriori [HPYM04]. This is mainly the result of saving database scans by dropping candidate generation: The algorithm uses one database scan to determine all frequent 1-item sets and a second run to create the so-called *FPTree* data structure, compressed representation of the database, which allows efficient generation of frequent item sets in a divide and conquer manner, again utilizing the frequency monotonicity.

These fundamentals on the topic of frequent item set mining are just the prerequisites for the technique of frequent term-based clustering, which is presented in the next Section 3.3.2. A good resource for extensive information on the frequent item set topic are the FIMI workshops: [GZ03], [JGZ04].

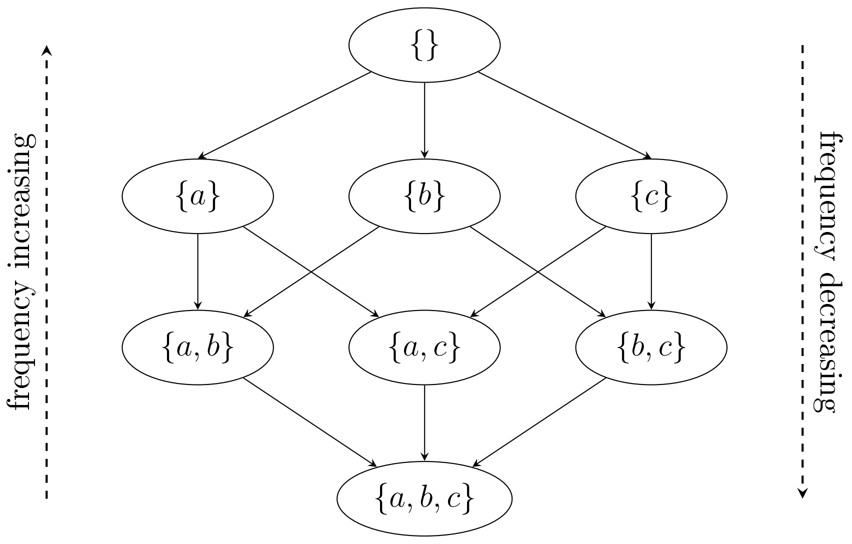


Figure 3.2.: The lattice of frequent item sets.

3.3.2. Clustering with Frequent Term Sets

Frequent term-based clustering is a clustering approach, which works on frequent item sets over a document corpus. The terms of this corpus correspond to the items and the transactional database is built from binary occurrence term vectors of the documents in the corpus (see Section 3.1).

A clustering algorithm attempts to partition a set of examples, such that examples in the same group are similar, or close to each other, in terms of a distance measure. A so-called *cluster analysis* is typically applied, when there is not a priori label assignment given for an example set, in order to detect groups of items which belong together. Where the task of training a classification model is referred to as supervised learning, clustering denotes the counterpart, *unsupervised learning.* In the scope of this thesis, a clustering is defined as:

Definition 13 (Clustering)**:**
A clustering $\mathcal{C} \subseteq \mathcal{P}(E)$ for an example set $E = \{e_1, \ldots, e_n\}$ is a set

$$\mathcal{C} := \{C_1, \ldots, C_m : C_i \subseteq E\}$$

for which $\bigcup_{C \in \mathcal{C}} C = E$ holds.

In the area of text mining, documents are typically clustered in order to group them by topic area or other criteria, see for example [HMS02], [LA99] and [ST00]. With the technique of frequent term-based text clustering, Beil et al. tackle two fundamental problems involved with clustering of text documents: Firstly, term vectors typically inhere in very high dimensionality, but are sparsely populated. This leads to long running clustering processes and possibly to poor clustering quality. Secondly, while a clustering groups the documents, there is typically no meaningful description for a cluster, e.g. a set of keywords.

The FTC technique is motivated by these problems: It avoids working directly in the highly dimensional space of term vectors by clustering on frequent term sets, and yields a description for each cluster in the form of a frequent term set.

The basis of the FTC technique is a frequent item set mining on text documents, i.e. the terms of a document corpus are the items. Through the cover function (Definition 11) each frequent term set is associated with the set of documents supporting it. This yields the definition of a frequent term-based clustering description.

Definition 14 (Frequent term based clustering description)**:**
Let D be a document corpus (Definition 7) and $\mathcal{F} = \{F_1, \ldots, F_n\}$ the set of frequent term sets, i.e. frequent item sets, over a binary occurrence term vector encoding of D. $\mathcal{CD} \subseteq \mathcal{F}$ is a frequent term-based clustering description over D, if

$$\bigcup\nolimits_{F_i \in \mathcal{CD}} \mathrm{cover}(F_i, D) = D$$

holds.

Note, that other common definitions of a clustering exclude overlapping clusters. This is not the case for Definition 13. Inferring a frequent term-based clustering directly from its description has a high probability of containing such clusters. However, the FTC algorithm itself returns a non-overlapping clustering.

Since overlapping clusters are not desirable, Beil et al. define two measures for the degree of cluster overlap: *Standard overlap* and *entropy overlap*. These measures are used in the FTC algorithm to select cluster candidates with low overlap potential. Let

$$\mathrm{f}(\mathcal{R}, D_j) := |\{F_i \in \mathcal{R} | F_i \subseteq D_j\}|$$

denote the number of term sets $F_i \in \mathcal{R}$, in some set $\mathcal{R} \subseteq \mathcal{F}$, which are supported by a document $D_j \in D$, i.e. $D_j \in cover(F_i, D)$.

Definition 15 (Standard overlap)**:**
The standard overlap (SO) measure of a cluster candidate C_i in respect to $\mathcal{R} \subseteq \mathcal{F}$ is defined as:

$$\mathrm{SO}(C_i, \mathcal{R}) := \frac{\sum_{D_j \in C_i} (\mathrm{f}(\mathcal{R}, D_j) - 1)}{|C_i|}$$

The standard overlap calculates the average number of other term sets supported by the documents in a cluster candidate, i.e. the potential of a cluster candidate to overlap with other clusters. For an optimal cluster description, the standard overlap of each cluster would be zero.

On basis of such an overlap measure, the FTC algorithm is in Algorithm 1. FTC follows a greedy approach, selecting subsequent clusters on basis of the smallest overlap with remaining cluster candidates. This is motivated by the large search space of potential clusterings, which is $O(2^{|\mathcal{F}|})$ in size. Once selected as a cluster, all documents covered by the term set are removed from the database and therefore from the cover of all remaining term sets. The overlap() function here is a place holder for either the standard overlap or the entropy overlap measure. The FrequentSets() function can be replaced by an arbitrary algorithm for frequent item set mining, e.g. FPGrowth.

Algorithm 1 Flat frequent term-based clustering

```
function FTC(database $D$, float $sup_{min}$)
    $selected := \emptyset$
    $n := |D|$
    $remaining :=$ FrequentSets($D, sup_{min}$)
    while $|\text{cover}(selected)| \neq n$ do
        $bestCand := \text{argmin}_{F_i \in remaining}\, \text{overlap}(F_i, remaining)$
        $selected := selected \cup \{bestCand\}$
        $remaining := remaining \setminus \{bestCand\}$
        $D := D \setminus \text{cover}(bestCand)$
        for $F_i \in remaining$ do
            $\text{cover}(F_i) := \text{cover}(F_i) \setminus \text{cov}(bestCand)$
        end for
    end while
    return $(selected, \{\text{cover}(F_i) | F_i \in selected\})$
end function
```

Note, that the FTC algorithms returns a set of cluster descriptions in addition to the clustering. The returned clustering is non-overlapping, since already covered documents are explicitly removed from the database and from the cover of all remaining term sets.

While the standard overlap measure is easy to calculate, it suffers from a fundamental drawback ([BEX02]): Due to the monotonicity property (Section 3.3.1) of frequent item sets, each document in the cover of a k term set supports at least all of its subsets. An algorithm based on the standard overlap measure is therefore expected to favor small item sets. To overcome this, Beil et al. also define a second overlap measure, the entropy overlap.

Definition 16 (Entropy overlap):
Let $C_i \in \mathcal{R}$ be a cluster candidate, $\mathcal{R} \subseteq \mathcal{F}$. The entropy overlap (EO) measure is defined as:

$$\mathrm{EO}(C_i, \mathcal{R}) := \sum_{D_j \in C_i} -\frac{1}{\mathrm{f}(\mathcal{R}, D_j)} \cdot \ln \frac{1}{\mathrm{f}(\mathcal{R}, D_j)}$$

The entropy overlap of a cluster becomes zero if the contained documents do not support any other remaining term set and it increases monotonically with the number of term sets supported by the documents in a cluster candidate. Beil et al. show empirically that the entropy overlap generally yields cluster descriptions with more and larger term sets than the standard overlap does. However, a larger number of term sets is not desired in the underlying case. Spot check experiments with the entropy overlap confirmed the assumption that the yielded label-reduction results in worse classification quality, compared to the standard overlap.

[BEX02] also describes a hierarchical flavor of FTC, which produces nested clusterings. The *Hierarchical FTC* (HFTC) discovers a tree structured clustering with the empty term set as its root (this covers the full database). The hierarchy is then created by applying the flat FTC recursively to the documents covered by a tree node. In addition, on every level k of the tree, only the k-term sets are taken into account. Furthermore, only such term sets need to be exploited, which are super sets in the tree node. This way, HFTC explores the frequent term set search space by seizing the monotonicity property.

The HFTC algorithm is not further concerned here, since it would require the selection of feasible nodes from the tree for further proceeding. The label reduction technique described in the following Section 3.3.3 uses only the flat FTC algorithm together with the standard overlap measure. For the same reason, the consideration of a similar approach by by Kaspari and Wurst ([KW07]) is omitted.

3.3.3. Label Reduction Technique

Section 3.2.2 showed that the power set transformation method is feasible for creating a single-label classification task from the problem of tag prediction in micro-blogging systems. However the performance of the classifiers evaluated on basis of this transformation yielded only moderate performance. It is suspected, that this is a result of the high number of labels generated by the power set transformation.

In order to reduce the number of labels, a custom label reduction approach is developed in this section. This approach is based on frequent term-based clustering of the tag vectors of micro-blogging entries. The tag vectors are clustered using the FTC algorithm. As described in Section 3.3.2, the algorithm does not only return a clustering of the given vectors, but also a description for each cluster, consisting of a frequent term set. This frequent term set is used to replace all tag sets contained in the corresponding cluster.

In this section, only the label reduction effect of this technique is shown by experiments. An empirical evaluation of the effects to classification quality can be found in Section 4.3.3.

The task of the label reduction algorithm is to lower the number of labels generated by the power set transformation approach (see Section 3.2.1).

Formally, this transformation method is based on a classifier

$$\begin{aligned} \text{h'}: &\quad X \to L' \\ \text{h'}(x) := &\quad \textstyle\bigwedge_{l \in h(x)} l \end{aligned}$$

In practice, this means a transformation of the multi-label example set, where a set of labels $M = \{l_1, \ldots, l_k\}$ is replaced by a single new label $M' = \bigwedge_{l_i \in M} l_i$. A transformed example set was presented in Table 3.2.

In order to reduce the number of transformed labels, the number of unique label sets must therefore be reduced. The idea behind the label reduction algorithm is to replace a set of, maybe slightly different, label sets with a frequent term set they have in common. An important requirement here is, that all tag sets must be covered by the label reduction mechanism. In order to assure this, all item sets, instead of just the frequent ones, are mined from the tag sets. Experiments presented in Section 3.3.4 have shown, that the set of all item sets over tag sets does not grow too large to be efficiently handled.

In order to identify tag sets, which are reasonably replaced by a common label set, and to gain the corresponding new label, the FTC algorithm is used. The algorithm clusters the given label sets based on frequent term sets mined from them. A cluster contains only tag sets, which have a specific item set in common. Furthermore, the algorithm delivers the common term set of a cluster as its description. This descriptive frequent term set is chosen to replace the label sets contained in the cluster.

The algorithm developed for label-reduction is presented in Algorithm 2. It receives the transactional database created from the label sets of a the multi-label example set. As the transaction ID, the example ID is used. Furthermore, it receives the example set itself, in order to replace the label sets. The transformed example set is returned.

On the label database, a FTC clustering is generated, using a support threshold which ensures that all item sets are mined instead of just the frequent ones. This has been chosen in order to guarantee that all original label sets are covered in the end. After that, the algorithm iterates over all frequent term clusters and all example IDs contained in them. It retrieves the corresponding multi-label example and adds it to the transformed example set, with the cluster description as its new label set.

Algorithm 2 Label reduction algorithm.

```
function FTCLabelReduction(label database $lD$, example set $E$)
    $E' := \emptyset$
    $(clusterDescs, clusters) := FTC(lD, \frac{1}{|E|})$
    for $c \in clusters$ do
        $M' := clusterDescs(c)$
        for $id \in c$ do
            $(x_i, M_i) := E(id)$
            $E' := E' \cup \{(x_i, M')\}$
        end for
    end for
    return $E'$
end function
```

It is expected, that the example set returned by the algorithm contains far less distinct multi-label sets, compared to the original one. If this is the case, the power set transformation method will also yield less distinct labels and it is expected that classification on this basis yields better results.

3.3.4. Evaluation

In following, the developed label-reduction technique (Algorithm 2) is evaluated on basis of two leading questions:

1. Does the algorithm yield the expected label-reduction effect?

2. Does the FTC algorithm produce sensible clusterings?

Label Reduction Effect

It is expected that the number of labels can be reduced significantly using the developed label reduction algorithm. Experiments confirmed this expectation: While on average a reduction rate of just 0.264 ± 0.199 could be achieved, an analysis of individual user data sets showed that, for data sets with a large amount of tags, the reduction rate is significantly better. Figure 3.3 visualizes the absolute reduction per user and reveals that a reduction by up to 0.57 is possible for users with many tags. In absolute numbers, this is in the best case a reduction to 182 power set labels, while it was 423 before.

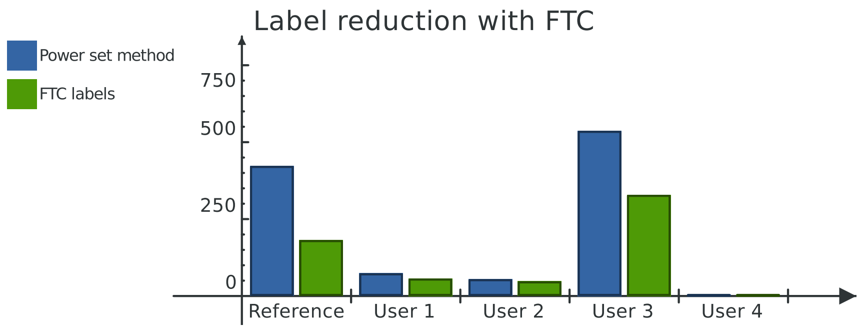


Figure 3.3.: Reduction of labels effect

Cluster Quality

To measure the quality of the FTC clustering, two criteria have been defined: Firstly, the intra and inter cluster distances has been computed, which is a common technique to judge the quality of a clustering. Secondly, clustering generated by the FTC technique haven been compared to clusterings generated my kMedoids, a clustering technique with well-known quality.

The general goal in clustering is to have compact clusters, i.e. the average intra cluster distance is low, while the distance between the clusters, i.e. average inter cluster distance, is higher by at least one magnitude. The intra and inter cluster distance measures, sometimes also computed as similarity measures, are commonly used to evaluate clusterings, e.g. [Azu02] and [KW98].

Different definitions of the two measures exist. The definitions used in this thesis are given in Definition 17 and Definition 19. For the intra cluster distance, the average distance between all contained examples is used. The inter cluster distance is calculated as the average distance between all examples from each cluster.

Definition 17 (Intra cluster distance)**:**
Given a cluster $C_i = \{e_1, \ldots, e_n\}$, the intra cluster distance is defined as

$$D_{intra}(C_i) := \frac{1}{|C_x|^2} \sum_{e_x \in C_i} \sum_{e_y \in C_i} \mathrm{d}(e_x, e_y)$$

$\mathrm{d}(e_x, e_y)$ is an arbitrary distance measure on the feature space $\mathcal{X}$. In case of this thesis, the *Euclidean distance* is used as the general distance measured, as specified in Definition 18.

Definition 18 (Euclidean distance)**:**
The Euclidean distance of two vectors $\vec{x} = (x_1, \ldots, x_d), \vec{y} = (y_1, \ldots, y_d)$ is defined

as

$$d(\vec{x}, \vec{y}) := \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

The inter cluster distance is defined in correspondence to the intra cluster distance:

Definition 19 (Inter cluster distance)**:**
Given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ and let $\mathcal{C}_{\setminus i} = \mathcal{C} \setminus C_i$ be the set of cluster without a specific cluster C_i. The inter cluster distance for a cluster C_i is defined as

$$D_{inter}(C_i) := \frac{1}{|C_i| \cdot \sum_{C_j \in \mathcal{C}_{\setminus i}} |C_j|} \sum_{e_x \in C_i} \sum_{e_y \in \bigcup \mathcal{C}_{\setminus i}} \mathrm{d}(e_x, e_y)$$

In order to measure the quality of a complete clustering, both defined measures are averaged over all contained clusters, while average values have not been weighted in any way.

Definition 20 (Average x cluster distance)**:**
Given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$

$$D_x(C) := \frac{\sum_{C_i \in C} D_x(C_i)}{|C|}$$

denotes the average cluster distance, where x is either intra or inter.

As an additional quality assessment, the given data sets have been clustered with a second clustering algorithm with well-known clustering quality: The kMedoids algorithm. In order to assess the clustering quality of FTC, the clusterings have been compared using the *Rand index* ([Ran71]). This measure assesses the similarity of two clusterings, as specified in Definition 21.

Definition 21 (Rand index)**:**
Given an example set $E = \{e_1, \ldots, e_n\}$ and two clusterings on E: $\mathcal{C}_1 = \{C_{1,1}, \ldots, C_{1,k}\}$ and $\mathcal{C}_2 = \{C_{2,1}, \ldots, C_{2,l}\}$. Let $\mathcal{C}_i(e_j)$ denote the cluster from $\mathcal{C}_i$ in which e_j resides.

The Rand index of $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as

$$\mathrm{rand}(\mathcal{C}_1, \mathcal{C}_2) := \frac{\sum_{i=1}^{n} \sum_{j=1}^{i} \gamma(e_i, e_j)}{\binom{n}{2}}$$

where

$$\gamma(e_i, e_j) := \begin{cases} 1, & \textit{if } \mathcal{C}_1(e_i) = \mathcal{C}_1(e_j) \Leftrightarrow \mathcal{C}_2(e_i) = \mathcal{C}_2(e_j) \\ 0, & \textit{otherwise} \end{cases}$$

In other words: The Rand index scores 1 for each two examples, if they reside in the same clusters in both clusterings or if they reside in different clusters in both clusterings. The Rand index is therefore even capable of comparing clusterings with a different number of clusters. The Rand index returns a value $\mathrm{rand}(\mathcal{C}_1, \mathcal{C}_2) \in [0, 1]$, while a value near 1 denotes a high similarity between the clusterings and lower values indicate more difference.

Two different experiments have been performed in order to asses the clustering quality of FTC in the underlying case: Firstly, FTC has been applied to the tag sets, as it is done in the label reduction technique. The resulting clustering is compared to a kMedoids clustering with k set to the number of clusters yielded by FTC.

The second experiment attempts to asses the quality of the label reduction process. As the basis, the label reducted example set is used, not only the clustered tag vectors. The label assignement is here interpreted as a clustering. This clustering is compared to a kMediods on the full example set.

It is expected, that FTC on the tag vectors returns a high-quality clustering, since it is specialized on document clustering. For the second experiment, results are not expected assess moderate to bad clustering quality, because the full example set contains much more information, which is not available to FTC on the tag vectors, but to the kMedoids algorithm.

The experiment results, presented in Table 3.7, underline these expectations. The table presents the values of inter and intra cluster distance, as well as the Rand index in comparison to a kMedoids clustering, for both described cases: The upper half of Table 3.7 shows the values for FTC and kMedoids on the tag vectors. The bottom half presents the intra and inter cluster distance for the example set after label-reduction has been applied. Rand index compares the clustering induced by reduced labels on the example set with the kMedoids clustering on it.

The frequent term-based clustering on the tag vectors turned out to be, as expected, of good quality. The average intra cluster distance over all data sets is 0.08 ± 0.08 while the inter cluster distance is, with 1.57 ± 0.12, more than a magnitude higher. The Rand index based comparison with kMedoids on the tag vectors also indicates a good clustering quality for FTC: The average Rand index between both clusterings is 0.91 ± 0.06, which is pretty high, indicating that both clusterings are almost equal.

The difference between average intra and inter cluster distance of the mapped FTC clustering is, as expected, lower. The inter cluster distance is not even half a magnitude larger than the intra cluster distance. Interestingly, the Rand index between this mapped FTC and kMedoids without mapping is higher than expected.

		Reference	User 1	User 2	User 3	User 4	Avg.
FTC (tags)	Intra	0.21	0.03	0.04	0.11	0	0.08 ± 0.08
	Inter	1.77	1.61	1.48	1.56	1.41	1.57 ± 0.12
	Rand	0.89	0.93	0.92	0.81	0.99	0.91 ± 0.06
FTC (mapped)	Intra	1.40	1.24	1.27	1.17	1.63	1.34 ± 0.16
	Inter	5.08	4.27	3.94	4.67	4.66	4.52 ± 0.39
	Rand	0.84	0.75	0.71	0.88	0.69	0.77 ± 0.07

Table 3.7.: FTC clustering evaluation.

With 0.77 ± 0.07 on average, the Rand index can still be considered quite high. While the clusterings are not almost equal, they still seem to be pretty similar.

Chapter 4

Classification

The task of tag prediction is a multi-label classification problem. In the concrete case of this thesis, this means: A micro-blogging entry can have an arbitrary number of tags assigned. Therefore, the targeted tag prediction technique should be capable of classifying a new micro-blogging entry with a set of multiple labels, i.e. tags.

In Section 3.2.1, different methods for transforming the multi-label problem into a single-label variant were presented and evaluated. The goal of these transformation methods is to tackle the multi-label problem with standard, single-label classification techniques instead of developing new approaches dedicated to the multi-label scenario.

As the result of this evaluation, the so-called power set method has been selected for being used within this thesis. The power set transformation method creates a new label for every distinct label set of the multi-label data set. This has been formalized by a classifier of the form

$$\begin{aligned} \text{h'}: &\quad \mathcal{X} \rightarrow L' \\ \text{h'}(x) := &\quad \textstyle\bigwedge_{l \in h(x)} l \end{aligned}$$

Experiments presented in Section 3.2.2 proved the assumption that the power set transformation yields a large number of labels. In order to diminish the effect of this large number of labels to classification quality, Section 3.3.3 presented a label reduction technique, which has been developed in scope of this thesis. The technique applies frequent term-based clustering to the label sets of the original data set and replaces all label sets in a cluster with a new one. It has been shown in Section 3.3.4, that this approach is capable of significantly reducing the number of labels generated by the power set transformation method.

The subsequent sections in this chapter are dedicated to evaluating four different classifiers on this basis: Three standard classifiers are selected for the evaluation in Section 4.1 – kNN, Naive Bayes and SVM. In addition to these, a custom classification technique is developed in Section 4.2. This approach works similarly to kNN, but generates a condensed representation of the training set. All four classifiers are evaluated in terms of their classification quality in Section 4.3.

4.1. Standard Classifiers

Three classifiers from the standard repertoire of machine learning have been selected to evaluate the multi-label transformation methods described in Chapter 3. These are:

1. The *k-nearest-neighbor* (kNN) algorithm was selected due to its simplicity, low training and moderate classification time. Furthermore, [Yan99] showed a general suitability of kNN for text classification purposes.

2. The *support-vector machine* (SVM) approach was chosen, because it is still an active topic of research (e.g. [JFY09], [TCCL07] and [MM05]) and well-known for its good generalization capabilities.

3. Finally, Naive Bayes is considered, since it is most commonly used in industry applications[1],[2], due to its high classification quality with low computational effort. Its most familiar application is detection of unsolicited email, which is also a text classification task.

In following, these three algorithms are explained in further detail.

4.1.1. kNN

The kNN algorithm is a so-called *lazy learner*, which performs all necessary computation at classification time. It simply stores all training examples in its model. For classification of a new example $e = (x, l)$, the algorithm searches for the k nearest neighbor examples in the training set E. The most common label among the neighbors is then predicted for the new example.

Formally, the classification function is defined as follows in this thesis

$$\mathrm{h}(\vec{x}) := \operatorname*{argmax}_{l \in L} |\{(\vec{x}_i, l) \in \mathrm{N_k}(\vec{x}, E)\}|$$

$\mathrm{N_k}(\vec{x}, E)$ denotes the k-neighborhood of $\vec{x}$ in the training set E, i.e. the set of k nearest training examples to $\vec{x}$, which is formalized in Definition 22, as used in this thesis.

Definition 22 (k-neighborhood):
Let $X := \{\vec{x}_1, \dots, \vec{x}_n\}$ be a set of vectors $\vec{x}_i \in \mathcal{X}$ and let $\vec{x} \in \mathcal{X}$ be a vector in the same space. The k-neighborhood of $\vec{x}$ in E is defined as

$$\mathrm{N_k}(\vec{x}, E) \subseteq E$$

subject to

[1]`http://getpopfile.org/` (2010-06-22)
[2]`http://www.sux0r.org/` (2010-06-22)

1. $|\mathrm{N_k}(\vec{x}, E)| = k$

2. $\forall \vec{u} \in \mathrm{N_k}(\vec{x}, E), \vec{v} \in E \setminus \mathrm{N_k}(\vec{x}, E) : \mathrm{d}(\vec{x}, \vec{u}) \leq \mathrm{d}(\vec{x}, \vec{v})$

where $\mathrm{d}(\vec{x}, \vec{y})$ *is a distance measure on* $\mathcal{X}$*.*

As the distance measure, the Euclidean distance is used in this thesis, which has already been specified in Definition 18 as follows:

$$d(\vec{x}, \vec{y}) := \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

A common variation of the kNN approach weights the nearest neighbors and casts a weighted vote among them, in order to classify the new example. The classification function can then be defined as follows:

$$\mathrm{h_w}(\vec{x}) := \operatorname*{argmax}_{l \in L} \sum_{(\vec{x}_i, l) \in \mathrm{N_k}(\vec{x}, E)} \mathrm{w}(\vec{x}, \vec{x_i})$$

where w is an arbitrary weighting function, typically based on the distance between the given vectors, i.e. more distant examples are weighted lower. Choosing a weight function, which always returns a constant value, e.g. 1, the weighted classification function is equivalent to the unweighted case.

A naive implementation of the kNN algorithm consumes $O(|E|)$ memory for storing all training examples. Furthermore it has a worst case classification runtime of $O(|E|)$, for comparing all training examples with the example to classify. Using an optimized data structure such as kd-tree [FBF77], the search of nearest neighbors can be reduced to an average $O(\log |E|)$ runtime with the drawback of additional computation time during the training phase, in order to prepare the data structure.

4.1.2. Naive Bayes

The Naive Bayes classifier produces a probabilistic classification model, that relies on a strong independence assumption regarding the features of an example. Despite this highly over-simplified assumption, the approach has practically proven its high classification quality, e.g. shown in [Ris] and [ELM03].

A Naive Bayes classifier relies on a conditional probability model. Its classification function can be described as

$$h(\vec{x}) := \operatorname*{argmax}_{l} \mathrm{P}(l | x_1, \ldots, x_d) \tag{4.1}$$

So the Naive Bayes classifier assigns the label with the highest probability, give the observed feature values.

The basis of this classification method is the probability

$$\mathrm{P}(L|X_1, \dots, X_d) \tag{4.2}$$

where $X_1, \dots, X_d$ are stochastic variables for the features and L is a stochastic variable for the label. The given probability is not easy to calculate itself, it is therefore transformed as follows:

$$\mathrm{P}(L|X_1, \dots, X_d) = \frac{\mathrm{P}(L)\,\mathrm{P}(X_1, \dots, X_d|L)}{\mathrm{P}(X_1, \dots, X_d)}$$

Further transformation using the rules for conditional probabilities yields

$$\frac{\mathrm{P}(L)\,\mathrm{P}(X_1|L)\,\mathrm{P}(X_2|L, X_1)\dots\mathrm{P}(X_n|L, X_1, \dots, X_{d-1})}{\mathrm{P}(X_1, \dots, X_d)}$$

The speciality of Naive Bayes classifiers is now the naive assumption, that every feature is conditionally independent of the others, given the label. Under this assumption, the term can be rewritten as

$$\frac{P(L)P(X_1|L)\dots P(X_d|L)}{P(X_1, \dots, X_d)} = \frac{P(L)\prod_{i=1}^{d} P(X_i|L)}{P(X_1, \dots, X_d)}$$

which results in a reformulation of the initial probability model from Equation 4.2:

$$P(L|X_1, \dots, X_n) = \frac{P(L)\prod_{i=1}^{d} P(X_i|L)}{P(X_1, \dots, X_d)} \tag{4.3}$$

In order to classify a new example, a Naive Bayes classifier calculates the conditional probability for every label, given the observed example features. It then assigns the label with the highest probability. The initial classification function (Equation 4.1) is the reformulated as

$$\mathrm{h}(\vec{x}) := \underset{l}{\mathrm{argmax}} \frac{P(L=l)\prod_{i=1}^{d} P(X_i = x_i|L=l)}{P(x_1, \dots, x_d)}$$

Storage and computation time consumption of a Naive Bayes model during classification are independent of the number of training examples: Examining the prediction probability term in Equation 4.3 yields, that the denominator is constant for a given example. The calculation of this part can therefore be safely left out in practice, with the result that no storage is consumed for a priori probabilities of feature values. The other probabilities can be estimated through relative occurrence frequencies in the training set.

It can therefore be deduced, that the storage consumption of a Naive Bayes model only depends on the number of features, their possible values, and the number of classes. Training of a Naive Bayes classifier works in linear time, based on the number of examples and features, while classification consumes linear time in the number of features and classes.

4.1.3. SVM

Support vector machines descend from the area of large margin classifiers, i.e. they determine a set of hyperplanes, which best divide data points belonging to different classes. While this works by default only with linearly separable data sets, SVMs can even be used with non-linearly separable data by using the so-called *kernel trick*. Figure 4.1 visualizes the idea of SVM with linearly separable data and two labels.

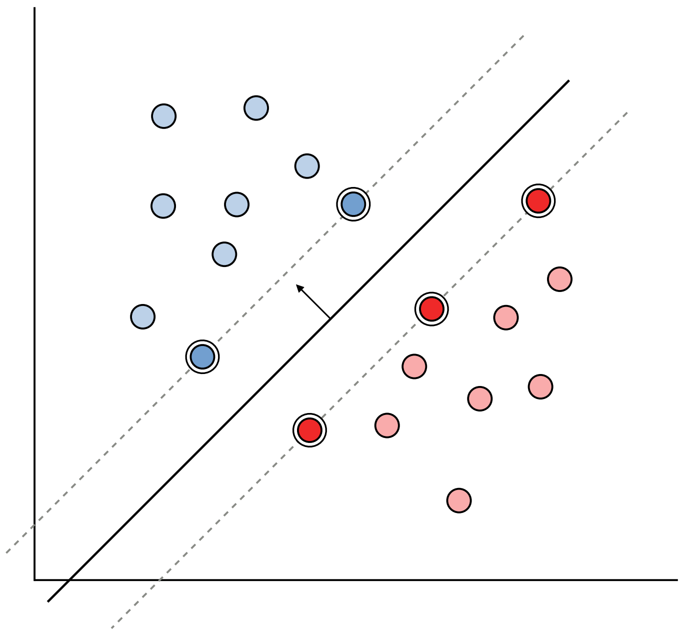


Figure 4.1.: Hyperplane in $\mathbb{R}^2$ with linearly separable data. The maximum margin hyperplane is shown, including margin (grey) and support vectors (strong color).

In the following, the SVM technique is described for the case of two labels and linearly separable data. The adjustment for data which is not linearly separable follows after that. In case there are more than two classes, one typically chooses a one-against-all approach or the multi-class SVM [FH02]. Without loss of generality, possible label values are in the following assumed to be $\{-1, 1\}$. In practice, label values can either be mapped to these or can be scaled. The presented description is inspired by the SVM definitions given by Mierswa in [Mie06] and Hastie et al. in [HTF01].

Linearly Separable Data

The goal of SVM is to find a hyperplane that separates a given set of data points in $\mathcal{X}$ by their class. This desired hyperplane can be defined as

$$H := \{\vec{x} | \langle \vec{w}, \vec{x} \rangle + b = 0\} \tag{4.4}$$

where $\vec{w}$ is the normal vector of the hyperplane and $\frac{|b|}{\|\vec{w}\|}$ determines its perpendicular distance to the origin. $\|\vec{w}\|$ denotes the Euclidean length of the vector $\vec{w}$. Once the parameters $\vec{w}$ and b have been determined, classification of a new example can be performed through

$$\mathrm{h}_{\vec{w},b}(\vec{x}) := \mathrm{sgn}(\langle \vec{w}, \vec{x} \rangle + b) \tag{4.5}$$

The classification function of SVM returns either -1 or 1. Assuming that a separating hyperplane is found, the following constraint must hold for all n examples in the training set E:

$$\forall (\vec{x}_i, l_i) \in E : l_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 0 \tag{4.6}$$

Since there exist an infinite number of hyperplanes that separate the given data, the target is an optimally separating hyperplane. Optimality in this case is defined as the hyperplane with maximum margin to the data points of both classes, formalizing the concept of *structural risk minimization* (SRM) [Mie06]. In fact, not all examples, but only the closest ones to the hyperplane must be considered in order to achieve this goal.

By normalizing b and $i\vec{w}$ the way that $|\langle \vec{w}, \vec{x}_i \rangle + b| = 1$ holds, Equation 4.6 can be transformed to

$$\forall (\vec{x}_i, l_i) \in E : l_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1$$

so that the margin of the desired hyperplane is given by $\frac{1}{\|\vec{w}\|}$. The goal is to maximize this margin, in order to find the optimal hyperplane. To ease later equations, this problem is commonly rewritten to the minimization of $\frac{1}{2}\|\vec{w}\|$, yielding the optimization problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\vec{w}\|^2 && (4.7)\\ &\text{s.t.} && \forall (\vec{x}_i, l_i) \in E : l_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 && (4.8) \end{aligned}$$

For easier handling of the inequality constraint, this problem is brought into Lagrangian form, yielding a Lagrangian multiplier α_i for every training example:

$$\mathrm{L_P}(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i l_i(\langle \vec{w}, \vec{x}_i \rangle + b) \tag{4.9}$$

In order to find the minimum, the derivatives in direction of w and b need to become zero, i.e.

$$\vec{w} = \sum_{i=1}^{n} \alpha_i l_i \vec{x}_i \tag{4.10}$$

$$0 = \sum_{i=1}^{n} \alpha_i l_i \tag{4.11}$$

Transforming the minimization problem to a maximization (Wolfe dual) version is achieved by substituting Equation 4.10 and Equation 4.11 into the Lagrangian (Equation 4.9):

$$\mathrm{L_D}(\vec{w}, b, \vec{\alpha}) := \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} l_i l_j \alpha_i \alpha_j \langle \vec{x}_i, \vec{x}_j \rangle$$

This finally yields the dual optimization problem to be solved in order to find the parameters for an optimally separating hyperplane:

$$\text{maximize} \quad \textstyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} l_i l_j \alpha_i \alpha_j \langle \vec{x}_i, \vec{x}_j \rangle \tag{4.12}$$

$$\text{s.t.} \quad \alpha_i \geq 0, \forall i = 1, \ldots, n \tag{4.13}$$

$$\text{and} \quad \textstyle\sum_{i=1}^{n} \alpha_i l_i = 0 \tag{4.14}$$

The optimal normal vector $\vec{w}^*$ can now be calculated by substituting the optimal vector $\vec{\alpha}^*$ into Equation 4.10, while the optimal offset b^* can be received using the initial constraint from Equation 4.8. $\vec{w}^*$ is actually a linear combination of example vectors $\vec{x}_i \in E$ for which α_i is non-zero. These examples, called the *support vectors*, are nearest to the hyperplane and therefore define the margin. The support vectors are highlighted through strong color and additional circles in Figure 4.1.

On this basis, an optimal separating hyperplane for linearly separable data can be found.

Non-Linearly Separable Data

In a scenario where data is not linearly separable, the described optimization does not yield a result, since Equation 4.8 cannot be fulfilled for all examples $(\vec{x}_i, l_i) \in E$. A simple solution to this issue is relaxing the constraint using slack variables $\xi_i, i = 1, \ldots, n$, as well as a corresponding correction term. This yields to an adjusted initial optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\|\vec{w}\|^2 + C \textstyle\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & \forall i : l_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 - \xi_i \end{aligned}$$

Solving this problem already helps with classifying data sets which are not fully linearly separable, for example due to outliers or noise. However, many cases exist, where a data set is not linearly separable at all, so this adjustment fails. The introduction of a kernel function allows the SVM to cope with such non-linearly separable data sets as well.

Examining Equation 4.12, it can be noted that the training feature vectors only occur within the dot product $\langle \vec{x}_i, \vec{x}_j \rangle$, which can be interpreted as the similarity of two data points in $\mathcal{X}$. Mapping data points into a different features space $\mathcal{K}$ before the dot product is calculated, makes the optimization problem depend on the dot product in this target space. Such an effect is achieved by a kernel function.

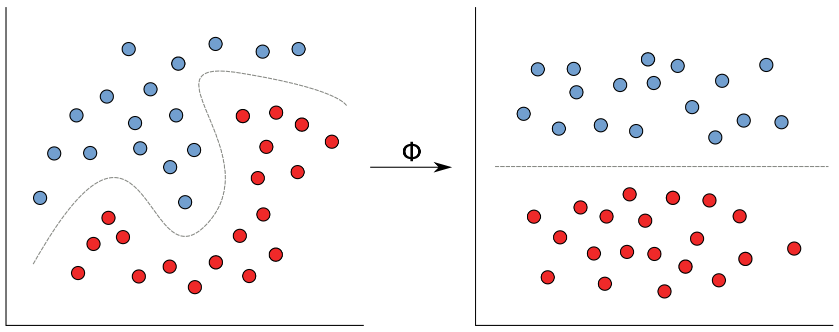


Figure 4.2.: Transformation of data which is not linearly separable to a different feature space where it becomes linearly separable.

Definition 23 (Kernel function)**:**
Given vectors $\vec{x}_1, \vec{x}_2 \in \mathcal{X}$ and a mapping $\Phi : \mathcal{X} \to \mathcal{K}$ into another space $\mathcal{K}$ with a dot product defined. A function $\mathrm{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for which

$$\mathrm{k}(\vec{x}_1, \vec{x}_2) = \langle \Phi(\vec{x}_1), \Phi(\vec{x}_2) \rangle$$

holds is called a kernel function or simply kernel.

Figure 4.2 visualizes the idea behind transformation into a different feature space so as to making data linearly separable. The fundamental advantage of such transformations is, that for some mappings Φ a kernel function k exists, which can be calculated without performing the actual feature space transformation. This can reduce computation time significantly and allows the usage of infinite dimensional spaces. A prominent example for kernel functions are *radial basis function* (RBF) kernels:

$$\mathrm{k}_{\mathrm{RBF}}(\vec{x}_1, \vec{x}_2) := \exp(-\frac{\|\vec{x}_1 - \vec{x}_2\|^2}{2\sigma^2}), \sigma > 0$$

Replacing the dot product in the linear optimization problem with a kernel function yields the general optimization problem

$$\begin{array}{rc} \text{maximize} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} l_i l_j \alpha_i \alpha_j \,\mathrm{k}(\vec{x}_i, \vec{x}_j) \\ \text{s.t.} & 0 \leq \alpha_i \leq C, \forall i = 1, \ldots, n \\ \text{and} & \sum_{i=1}^{n} \alpha_i l_i = 0 \end{array}$$

Solving this problem with a fitting kernel function allows to use SVMs with non-linearly separable data.

Computation Time

SVM has faced versatile attention in research for a long time now, starting with a paper by Vapnik [VC74] in 1974. Fields of application include face detection

[OFG97], text categorization [Joa97b] and many others. Training a SVM is a *quadratic programming* (QP) problem, which can in general be solved in $O(|E|^3)$. Different approaches exist for solving the optimization problem somehow more efficiently, such as *Sequential Minimal Optimization* (SMO) [Pla98], EvoSVM [Mie06], SVM$^{\text{light}}$ [Joa99] and the Cutting-Plane Algorithm by Joachims, which trains linear SVMs in linear time.

Despite its high requirements in terms of computation time and memory during training phase, SVM has been chosen to be evaluated in the scope of this thesis for its empirically well-known classification quality.

4.2. Custom Classification Approach

In the following, a custom classifier is developed, inspired by the kNN approach which has been explained in detail in Section 4.1.1. A fundamental issue of kNN is, that the algorithm stores the full set of training examples as its model and, in worst case, needs to compare a new example with all of these training examples, in order to classify. The in following presented approach attempts to mitigate this issue. Instead of storing the full training set, a condensed representation is stored. This representation consists of a set of representative vectors per class. An example for such a representative is the average vector of the class.

A similar approach is considered in [HK00]. However, [HK00] uses the cosine similarity measure instead of the Euclidean distance for vector comparison and only concentrates on the average vector as the representative. In contrast to that, the underlying thesis considers different representative generation techniques.

The following Section 4.2.1 motivates the developed approach further on basis of empirical results presented in Section 3.3.4. After that, the algorithm to transform an example set into its condensed representation is developed in Section 4.2.2. Finally, Section 4.2.3 presents three different approaches for generating a class representatives: The average and median vector are considered, as well as the technique of selecting well scattered points from the examples of a class. The latter approach is proposed in [GRS98] for cluster representation.

An empirical evaluation of the approach on basis of the described representative generation techniques can be found in Section 4.3.

4.2.1. Motivation

kNN is an appealing approach due to its simplicity and clarity. However, experiments – presented in Section 4.3 – turned out that it performs only moderately well in the scenario of tag prediction in micro-blogging systems. Furthermore, the technique developed to reduce the number of unique labels (Section 3.3.3) did not yield significant enhancements on kNN classification quality.

It is expected, that the observed deficiency is a result of the curse of high dimensionality, i.e. the sparseness of high-dimensional term vectors and the resulting short distance between examples of different classes:

The content of a micro-blogging entry is limited to 140 characters, which induces a maximum number of 70 terms, i.e. 70 term vector dimensions are non-zero. Compared to the number of distinct terms in the vocabulary of a user sample, which is in average 4415.60 (Appendix A), this can be considered a sparse vector population. Given two term vectors with no weights in common, a maximum Euclidean distance of 11.83 can be reached, which is considered low. This assumption of low distance between examples is also underlined by the evaluation of frequent term-based clustering in Section 3.3.3, where the average inter cluster distance turned out to be also low (about 5).

Motivated by these observations, a classification approach similar to kNN has been developed in the scope of this thesis: The approach relies on transforming a given training set into a condensed representation: For each class, a set of representative vectors are stored instead of storing the full training set. On the basis of these representatives, kNN classification is performed with $n = 1$. This strategy is expected to yield two enhancements in respect to classical kNN classification:

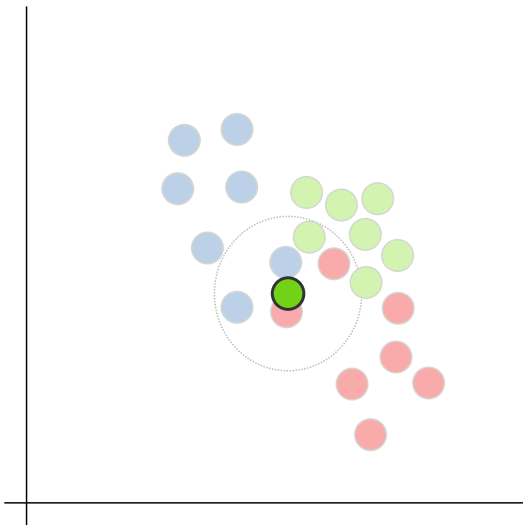
1. The worst case computation time during classification is independent from the number of seen training examples, similar to the Naive Bayes classifier (see Section 4.1.2).

2. The classification quality is expected to be better than for kNN, since representatives are expected to be more distant than examples in the training set.

Figure 4.3 visualizes the different approaches of kNN classification and proposed classifier. A setup with three different classes is shown (green, red and blue). The example to classify is shown in strong color of its ground truth label. The grey circle is only a guide. While kNN considers all training examples for classification, the developed classification approach takes only a representative per class into account. For $k \in \{1, 3\}$, kNN would classify the new example incorrectly as red. In contrast to that, 1NN on the class representatives (Figure 4.3b) would classify the new example correctly as green.

4.2.2. Algorithm

The classification approach proposed in this thesis works on basis of the kNN algorithm. Instead of searching for nearest neighbors in the full training set, only a smaller set of representative vectors per class are taken into account. This approach is formalized in following on basis of an example set transformation. First, the generation of a representative vector set from the examples of a class is defined.


(a) kNN

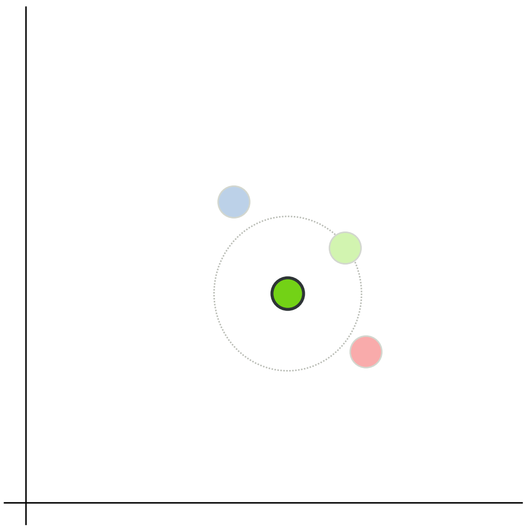

(b) Representatives

Figure 4.3.: kNN vs. representative based classification.

After that, this transformation is applied to all classes that occur in the training set, in order to transform it into a new example set. On this transformed example set, kNN is applied.

To ease equations $\mathcal{F} := \mathcal{X} \times L$ denotes the space of all possible examples.

Definition 24 (Representative function)**:**
Let $E \subset \mathcal{F}$ be an example set. E_l denotes all examples from E with label l, i.e. $\forall(\vec{x}_i, l_i) \in E : l_i = l \Rightarrow (\vec{x}_i, l_i) \in E_l$. A representative function in E is defined as

$$\mathrm{r} : \mathcal{P}(E_l) \to \mathcal{P}(\mathcal{F})$$

such that

$$\mathrm{r}(E_l) := \{(\vec{x}_s, l), \ldots, (\vec{x}_t, l)\}$$

subject to

1. $|\mathrm{r}(E_l)| \leq |E_l|$

2. $\forall l_i \in L : |\mathrm{r}(E_l)| \leq \mu$

for a fixed value μ.

Definition 25 specifies a general scheme for a function that generates a representative for a class. The representative is defined as a set of, potentially new, examples, calculated on basis of the original examples in the class. A class representative is intentionally not defined as a single vector, e.g. a centroid, but as a set of vectors. However, this set is restricted to be no larger as the original set of examples and to not exceed a defined size μ. The latter constraint ensures that the worst case classification time is independent of the number of training examples. Section 4.2.3 presents different approaches for generating representatives.

On this basis, an example set transformation is specified in Definition 25.

Definition 25 (Representative example set transformation)**:**
Given an example set E and its label set L. The transformation of E through a representative function r is defined as:

$$\mathrm{t_r} : \mathcal{P}(\mathcal{F}) \to \mathcal{P}(\mathcal{F})$$

given through

$$\mathrm{t_r}(E) := \bigcup_{l \in L} \mathrm{r}(E_l)$$

with E_l as specified in Definition 24.

The classification function is then defined on basis of such a transformation as

$$\mathrm{h}(x) := \operatorname*{argmax}_{l \in L} |\{(\vec{x}_i, l) \in \mathrm{N_k}(\vec{x}, \mathrm{t_r}(E))\}|$$

following the kNN approach on the set of all representative vectors. In scope of this thesis, only $k = 1$ is considered. Algorithm 3 presents a more practical description the example set transformation.

Algorithm 3 Representative example set transformation.

```
function ReprTransformation(example set $E$, label set $L$)
    $transSet := \emptyset$
    for $l \in L$ do
        $transSet := transSet \cup \mathrm{r}(E_l, l)$
    end for
    return $transSet$
end function
```

On this basis, the algorithm can be adapted by using arbitrary representative generation functions which conform to Definition 24. Furthermore, no new algorithm needs to be developed in order to perform the actual classification, but the kNN algorithm can be re-used.

4.2.3. Representative Calculation

Three different methods for calculating class representatives have been evaluated within this thesis. These methods are used to perform the example set transformation as described in Section 4.2.2.

First, the centroid based approach from [HK00] is considered, where the average vector of all the examples in a class is used a single representative vector. The second approach in consideration, chooses the median example of a class as its representative. Finally, the concept of *well scattered points*, as defined in [GRS98], is considered. Here, a fixed number of examples is chosen to represent a class.

In the following, these approaches for generating a class representative are described in further detail. The basis of each formalization is the representative function, which has been specified in Definition 24 as

$$\mathrm{r} : \mathcal{P}(E_l) \to \mathcal{P}(\mathcal{F})$$

with $E_l \subseteq E$ being all examples from $E \subset \mathcal{F}$ with label l.

Section 4.3 evaluates the proposed classification method together with the transformation functions described in the following. Figure 4.4 visualizes a class with three examples and each of the presented representative generation approaches.

Average and Median Representative

The average representative is calculated by the following function:

$$\mathrm{r}_{\mathrm{avg}}(E_l) := \{(\mathrm{avg}(E_l), l)\} \tag{4.15}$$

while avg calculates the average vector of a given set of vectors F as

$$\mathrm{avg}(F) := \frac{1}{|F|} \sum_{(\vec{x},l)\in F} \vec{x}$$

So, the representative set here consists only of a single new example, for which the feature vector is calculated as the average of all feature vectors in the class. The average representation is also used in [HK00].

The median representative function generates a set with only one example, too. In contrast to the average representative, where a new example is generated, the median representative function chooses one of the existing examples in the class:

$$\mathrm{r}_{\mathrm{median}}(E_l) := \{(\mathrm{median}(E_l), l)\} \tag{4.16}$$

with

$$\mathrm{median}(F) := \underset{\vec{x}}{\mathrm{argmin}} \sum_{(\vec{y},l)\in F} d(\vec{x}, \vec{y})$$

The chosen example has the smallest distance to all examples in the class. $d(\vec{x}, \vec{y})$ could be any distance measure on $\mathcal{X}$, but within this thesis, only the Euclidean distance (Definition 18) is considered.

Both presented representative functions obviously comply to Definition 24.

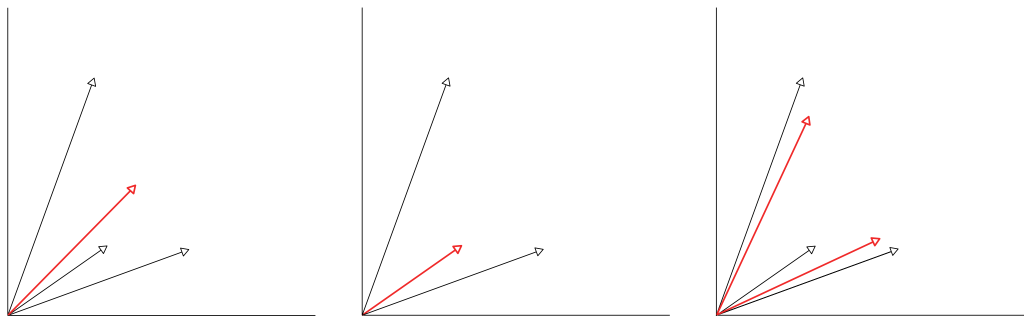

(a) Average (b) Median (c) Well scattered points

Figure 4.4.: Different types of class representatives. One class with 3 examples, chosen representatives in red.

Well Scattered Points

The technique of choosing *well scattered points* as a class representative originates from Guha et al. [GRS98]. There, the *CURE* clustering algorithm for large databases is described. CURE represents a cluster through a small number of points from it, in order to preserve the shape of the cluster to some degree. This idea is adapted to a class representative, in scope of this thesis. The adaption is obviously simple, since examples in a class can essentially be seen as a cluster.

Guha et al. observe that a single cluster representative, e.g. the average vector centroid, can describe small clusters of spherical shape quite well. But such a representative does not describe clusters of different shape accurately. Similarly, representing a cluster through all of its points is also not suitable: Both approaches result in incorrectly merged clusters in the CURE system. To avoid this problem, a cluster is represented by a set of points from it. These points are chosen to be well scattered within the cluster: They have the maximal distance among each other, i.e. lie on the cluster border, in order to conserve the cluster shape to some degree.

To mitigate the influence of outliers, the well scattered points are shrunk by a certain amount in the direction of the cluster average vector. Algorithm 4 presents the procedure of choosing well scattered points from a set of vectors. The algorithm has been slightly adapted to work as a representative function in this thesis.

The shown algorithm receives a set of feature vectors, the number of points to be generated and a shrinking factor $\alpha \in [0, 1]$. It first calculates the average vector, as defined in Equation 4.15. The average is then used to determine the first well scattered point as farthest away from it. Subsequent points are selected as far as possible away from all already selected ones. Finally, all selected points are shrunk in direction of the average by the fraction α.

The shrinking of well scattered points in direction of the cluster average does not

Algorithm 4 Algorithm to generate well scattered points

```
function WellScatteredPoints(vector set $c$, int $pointCount$, float $\alpha$)
    $tmp := \emptyset$
    $wsp := \emptyset$
    $avg := \mathrm{r}_{\mathrm{avg}}(c)$
    for $i \in \{1, \ldots, \min\{pointCount, |c|\}\}$ do
        $maxDist := 0$
        for $p \in c \setminus tmp$ do
            if $i = 1$ then
                $dist := \mathrm{d}(p, avg)$
            else
                $dist := \min\{\mathrm{d}(p, q) | q \in tmp\}$
            end if
            if $dist \geq maxDist$ then
                $maxDist := dist$
                $maxPoint := p$
            end if
        end for
        $tmp := tmp \cup \{maxPoint\}$
    end for
    for $p \in tmp$ do
        $wsp := wsp \cup \{p + \alpha(avg - p)\}$
    end for
    return $tmp$
end function
```

only mitigate the effect of outliers, but also influences to which degree the shape of the point set is conserved, together with the number of points to generate: By choosing more well scattered points, the original shape is conserved in more detail. A low α value leaves the selected points about at their original position. As a result, surface abnormalities are recorded more strongly. A larger α value moves the well scattered points more in direction of the average, mitigating the effect of surface abnormalities.

Figure 4.5 visualizes the idea of well scattered points and Figure 4.4c shows the technique in comparison to other presented representative generation techniques. On basis of the algorithmic description of the well scattered points technique (Algorithm 4), the representative function can be defined as

$$\mathrm{r}_{\mathrm{WSP}}(E_l) := \{(x_i, l) | x_i \in \text{WellScatteredPoints}(E_l, c, \alpha)\}$$

while c, α are the parameters for the algorithm to be defined in advance. This representative function conforms to Definition 24: Although being manipulated,

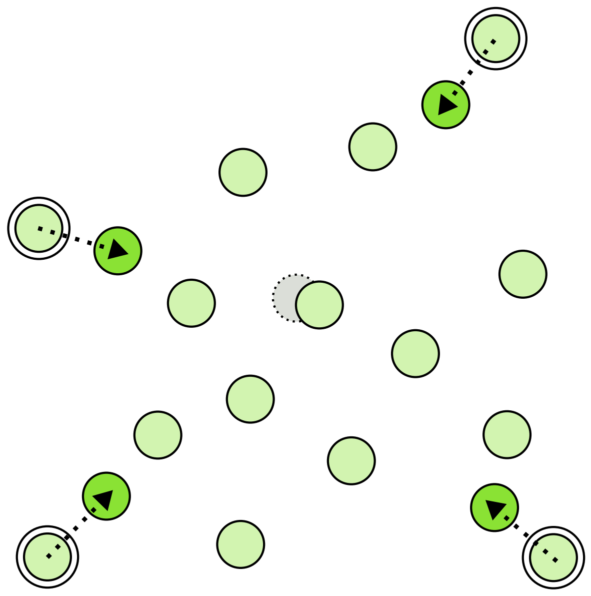


Figure 4.5.: 3 well scattered points (strong color) selected from 15 examples (pale). Examples with additional circles have been selected and have been shrunk in direction of the average vector (light grey).

the well scattered points reside in the same feature space as the input examples. The number of points is in maximum equal to the number of input examples. Furthermore it is overall bound by the parameter c. Choosing $\mu = c$ makes the maximum number of vectors condition hold.

4.3. Evaluation

Four classification techniques have been presented throughout this chapter, which are to be evaluated in following, regarding their feasibility for tag prediction in micro-blogging systems. Three methods have been selected from the standard repertoire of machine learning: kNN (Section 4.1.1), a simple lazy classifier, Naive Bayes (Section 4.1.2), which uses a probabilistic model, and the support vector machine approach (Section 4.1.3), which relies on the large margin principle. In addition to these, a custom classification approach has been developed in Section 4.2. This approach attempts to transform an example set into a condensed representation, consisting of representative vectors per class. On this basis, 1NN based classification is performed.

The selected classifiers are evaluated on basis of the multi-label transformation methods selected and developed in Section 3.2: This is firstly the power set method, as explained in [TK07], which transforms a multi-label data set into a single-label variant. This has been formalized by a classification function of the form

$$\begin{aligned} \mathrm{h}' : \quad & X \to L' \\ \mathrm{h}'(x) := \quad & \textstyle\bigwedge_{l \in h(x)} l \end{aligned}$$

in Section 3.2.1. Practically, this induces a transformation of the original multi-label data set into a single-label data set, by replacing each distinct set of labels of the form $\{a, b, c\}$ by a new, single label $a \wedge b \wedge c$. After this transformation, standard classifiers can be used on the example set.

Experiment results, anticipated in Section 3.2.2, proved the assumption correct, that this approach yields a large number of labels, resulting in only moderate classification quality. To solve this issue, a method for reducing the number of distinct multi-label sets has been developed in Section 3.3.3. This method replaces a number of similar label-sets with a new one, using frequent term-based clustering. Section 3.3.4 already evaluated the feasibility of the developed technique to reduce the number of generated power set labels significantly.

A leading question for the following evaluation is therefore, if the classification quality can be raised by applying the label reduction technique before the power set method?

In the following Section 4.3.1, the evaluation criteria for experiments in this thesis are presented. This includes the selection of two measures to determine classification quality: The classical *accuracy* measure and the *α-accuracy* as defined in [BLSB04]. Furthermore, the statistical test method of *cross validation* is introduced, which is used to assess generalization performance of classification methods.

After that Section 4.3.2 presents some general information of the experiment setup, including the topic of *evolutionary parameter optimization*, which has been used for the purpose of selecting classifier parameters.

Finally, performed experiment series are presented and evaluated: The first series (Section 4.3.3) measures multi-label classification quality for the three selected standard classifiers, on basis of the power set transformation method. Furthermore, the results are compared to classification on basis of reduced label sets. After that, the presented quality measures (accuracy and α-accuracy) are compared. Finally, the custom classification approach is evaluated in the same manner as for the standard classifiers, and the results of both experiment series are compared.

4.3.1. Evaluation Criteria

In order to assess the quality of a classifier, this thesis utilizes two quality measures: The classical accuracy and a variation of it, adjusted to the multi-label environment, the so-called α-accuracy. In order to assess generalization performance of a classifier, the statistical standard method of cross validation is used.

Quality Measures

The major criterion for evaluating classification approaches in this thesis is the classification quality. Two different quality measures are used throughout the experiments: The standard accuracy measure is generally used to evaluate classification quality. In addition to that, the α-accuracy, defined by Boutell et al. in [BLSB04], is used in order to evaluate the classification performance, with special attention to the multi-label scenario.

Definition 26 (Accuracy)**:**
The accuracy $\mathrm{acc}(\mathrm{h}, E)$ *of a classifier* $\mathrm{h} : \mathcal{X} \to L$ *on an example set* E *is defined as:*

$$\mathrm{acc}(\mathrm{h}, E) := \frac{|\{(\vec{x}, l) \in E |\, \mathrm{h}(\vec{x}) = l\}|}{|E|}$$

The accuracy measure determines the fraction of correctly classified examples. In contrast to that, the α-accuracy measures multi-label classification quality in a more fine-grained way: While the accuracy measure only distinguishes correct and incorrect classifications, α-accuracy also takes partly correct classifications into account. The following example motivates such a measure:

Given a multi-label example $e = (\vec{x}, M)$ as specified in Definition 4, for which the ground truth label set is $M = \{a, b, c\}$. A classification result $N_1 = \{a, c\}$ for this is incorrect in terms of the classical accuracy, although $\frac{2}{3}$ of the predicted label set was actually correct. Similarly, a prediction $N_2 = \{a, b, d\}$ would be considered incorrect by classical accuracy. The goal of a multi-label aware accuracy measure should therefore be to weight such partly incorrect classifications accordingly.

The α-accuracy fulfills these criteria and scores classification results based on their degree of correctness. Basically, two cases of partly correct classifications can occur:

Missed label	In the case $M - N \neq \emptyset$, not all labels from the ground truth set of the example have been assigned by the classifier. The classifier missed true labels.
False positive	Here, the classifier assigned additional labels which are not part of the true label set: $N - M \neq \emptyset$.

The α-score of a classification result considers both of these cases.

Definition 27 (α-score)**:**
Given two label sets $M, N \subseteq L$*, the* α*-score is defined as*

$$\mathrm{score}_\alpha(M, N) := \left(1 - \frac{\beta \cdot |M - N| + \gamma \cdot |N - M|}{|M \cup N|}\right)^\alpha$$

The parameters β and γ allow to weight false positive and missed labels differently. α is called the *forgiveness rate* in [BLSB04]. It permits to adjust the overall penalty of a partly incorrect prediction: A low α value makes the scoring less discerning, while $\alpha = \infty$ results in the score 0 for any classification error and 1 only for completely correct labeling, which is essentially the classical accuracy measure. It has to be noted that, for a sensible scoring, the constraints $\beta, \gamma \in [0, 1]$ must hold. Furthermore, Boutell et al. constraint $\beta = 1 \vee \gamma = 1$ and $\alpha \geq 0$.

On basis of this scoring function, a new notion of accuracy is defined for the multi-label case [BLSB04], which is called the α-accuracy in this thesis:

Definition 28 (α-accuracy)**:**
Given an example set E and a multi-label classifier $\mathrm{h} : \mathcal{X} \to \mathcal{P}(L)$*, the α-accuracy is defined as*

$$\mathrm{acc}_\alpha(E, \mathrm{h}) := \frac{1}{|E|} \sum_{(\vec{x}, M) \in E} \mathrm{score}_\alpha(M, \mathrm{h}(\vec{x}))$$

For experiments in this thesis, the classical accuracy is generally calculated, unless otherwise noted. The α-accuracy is calculated additionally for experiments involving the power set transformation method. The leading question to be answered by the α-accuracy in the scope of this thesis is if incorrect predictions by a classifier are completely wrong in general, or if partial incorrectness occurs. In order to asses this, the α scoring is simplified by choosing $\alpha = \beta = \gamma = 1$. This way, different types of misclassification are penalized equally and the resulting score is not scaled in terms of forgiveness:

$$\mathrm{score}'_\alpha(e) := 1 - \frac{|M \cap N|}{|M \cup N|}$$

The result is a 1 scoring for completely correct classifications and therefore in an accuracy value which is at least equal or larger than the classical accuracy. In case the α-accuracy is significantly larger than the classical variant, this indicates a fair amount of only partly incorrect classifications. Of two classifiers which yield a similar accuracy, the one with better α-accuracy values is considered to be of better quality in this thesis.

Cross Validation

Cross validation (often *X-validation*) is a method of statistical testing, used to asses the generalization quality of a classifier. For this purpose, multiple experiments are performed on different samples of a data set. In one round of X-validation, the data set is partitioned into training and test data. The training set is then used to learn a classification model, which is evaluated against the test data.

The most common form of this method is the *k-fold cross validation*: Here the sample data set is partitioned into k non-overlapping sub sets (folds). For each of these folds, a cross validation round is performed, where this fold is retained as test data, while the remaining $k-1$ sub sets are used as training data. The quality measures of all such experiments are typically averaged to assess generalization performance.

In order to ensure that each cross validation round yields somewhat representative results, the method of *stratified sampling* is used for cross validation in this thesis. This sampling strategy ensures that each class is properly represented in training and test data, i.e. each class occurs with the same fraction as it occurs in the full data set [WF05].

In general *10-fold cross validation* is used for experiments, since this is the most common setup, according to [WF05]. In case parameters are optimized using the technique of evolutionary parameter optimization (see next Section 4.3.2), the number of cross validation runs has been reduced, in order to shorten the run time of experiments. However, in order to present comparable results, dedicated experiments with the selected parameters and 10-fold cross validation have been run in addition.

4.3.2. Experiment Setup

In order to evaluate the chosen classification methods, five real-life data sets have been extracted from the Twitter web service, each based on a dedicated user. While four of these users were selected pseudo-randomly, one specific user was chosen, based on a priori knowledge of his tagging behavior. Details about the data set, the process of its extraction and a basic statistical analysis can be found in Appendix A.

The experiments in this thesis have been performed using the *RapidMiner*[3] framework for rapid data mining and machine learning. The general experiment setup is presented in following.

RapidMiner

All experiments in scope of this thesis have been run in the RapidMiner framework, using default operators delivered with the framework, unless otherwise noted. RapidMiner provides a comfortable interface for the creation and execution of data mining and machine learning experiments. Furthermore, it allows the user to develop custom operators using a plug-in mechanism. As far as possible, experiments were built using RapidMiner standard operators. Where necessary, new operators where implemented.

[3]`http://rapid-i.com/content/view/181/190/` (2010-07-24)

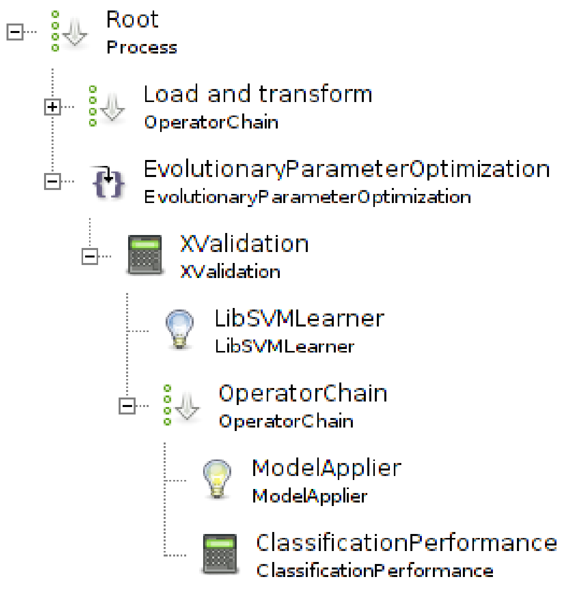


Figure 4.6.: Classification evaluation operator tree in RapidMiner

Figure 4.6 shows a prototypical experiment setup in the RapidMiner user interface which is prototypical for the evaluation experiments in this chapter. Experiments are structured in form of a tree of operators, while each operator receives the results produced by previous operators. In this specific case, the first operator loads the data set and performs the multi-label transformation, including label reduction. The following operator optimizes the parameters of the contained SVM classifier. For each generation of parameter values, the SVM classifier is run through cross validation, in order to measure its accuracy and assess generalization performance. A more detailed explanation of experiment setups can be found in Appendix B.

Parameter Optimization

Some classification methods can be parameterized in order to adjust their behavior to a given scenario. In the scope of this thesis, this holds for kNN and SVM (see Section 4.1). An appropriate set of parameter values for a given classifier on a given data set can be obtained through an optimization process using a classification quality measure as the objective function, which is to be maximized. Experiments serving this purpose are called *parameter optimization*.

The essential problem in the area of optimization problems is the search space size, which grows exponentially in the number n of parameters to optimize, i.e. $O(2^n)$. Even with a small amount of parameters, it is not feasible to explore the complete search space. Without a pre-defined, finite value set for each parameter, full exploration is even impossible.

A common approach to solve this problem at least approximately, is the technique of evolutionary parameter optimization, which is used throughout this thesis whenever parameters need to be optimized for an experiment setup. Instead of trying to explore the complete search space of parameter value combinations, an evolutionary algorithm is used here to maximize the accuracy of a desired classification function.

Evolutionary algorithms realize mechanisms, inspired by biological evolution [BS93], in order to select an optimal element from a given search space. An element of the search space $\vec{a}_i \in J$ is called an *individual*. An evolutionary algorithm works round-based, subsequently generating and evaluating populations $P(t) = \{\vec{a}_1(t), \dots, \vec{a}_n(t)\}$ of a pre-defined size n. The population generated in a specific round $t \in \{0, \dots, m\}$ of the algorithm is called a *generation*.

The first population $P(0)$ is typically initialized arbitrarily. Subsequent populations $P(t+1)$ are generated through *recombination*, *mutation* and *selection* of individuals from $P(t)$. The suitability of individuals is evaluated through a fitness function $f : J \to R$, which typically involves the objective function of the optimization problem, but is not necessarily equal to it. The end of an evolutionary algorithm is defined by a stop criterion, which usually involves a certain number of generations or a convergence threshold regarding the fitness of individuals.

A simplified and slightly adjusted version of the prototypical evolutionary algorithm shown by Bäck and Schwefel in [BS93], is presented in Algorithm 5. The algorithm receives an individual definition, which describes how individuals look like (e.g. number of variables, value ranges). At first, the initial population of individuals is generated. This generation is evaluated. The *while* loop generates subsequent generations by recombination, mutation and selection until the stop function returns *true*. Finally, the best individual from the last generation is returned.

In the case of classifier parameter optimization, an individual is a specific combination of parameter values for a specific classifier. For example C, γ for the SVM variant used within this thesis (see Section 4.3.2). The objective function here corresponds to the fitness function, i.e. fitness corresponds to classification accuracy.

For experiments in this thesis, the standard settings for evolutionary parameter optimization in RapidMiner were used:

- populations consist of five individuals
- the optimization process stops latest after 50 generations or if a generation does not yield fitness improvement in respect to its predecessor
- the Gaussian mutation operator is used, in combination with selection by tournament.

Algorithm 5 Prototypical evolutionary algorithm.

```
function EvolutionaryOptimization(individual definition $def$)
    $t := 0$
    $P(t) := \{\vec{a}_1(0), \dots, \vec{a}_n(0) | \vec{a}_i \in def\}$
    Evaluate($P(t)$)
    while $\neg$Stop($P(t)$) do
        $P'(t) :=$ Recombine($P(t)$)
        $P''(t) :=$ Mutate($P'(t)$)
        Evaluate($P''(t)$)
        $P(t+1) :=$ Select($P''(t)$)
        $t := t + 1$
    end while
    return Optimum($P(t)$)
end function
```

A detailed overview on the topic of evolutionary algorithms for parameter optimization can be found in [BS93].

Classifier Specifics

For experiments in this thesis, the following special settings apply to the used classifiers.

The kNN classifier needs a single parameter k, which determines the number of training examples taken into account for classification. This parameter has been determined by evolutionary parameter optimization as described. For each individual parameter value, cross validation is performed as described in Section 4.3.1. The operator shipped with RapidMiner has been used for kNN experiments.

Some problems occurred with the NaiveBayes operator shipped with RapidMiner in scope of experiments in Chapter 5: This operator produces a model, which cannot be updated with new classes, which occur in streaming experiments. Therefore, a custom Naive Bayes operator has been implemented, which supports this feature. Since the Naive Bayes does not provide any parameters, no optimization was necessary. Therefore, this classifier has been evaluated only with 10-fold cross validation.

The SVM classifier shipped with the RapidMiner framework has been used for the evaluation experiments. It is backed by LibSVM [CL01]. As recommended in [wHcCjL03], the C-SVC [SSMB00] implementation on basis of the Gaussian RBF kernel

$$\mathrm{k}_{\mathrm{Gauss}}(\vec{x}_1, \vec{x}_2) = \exp(-\gamma \|\vec{x}_1 - \vec{x}_2\|^2)$$

has been chosen. According to [KL03] this kernel behaves similarly to the linear SVM for some parameter combinations. Furthermore, the sigmoid kernel, with some parameter combinations, behaves similarly to RBF [tLjL03]. Therefore, this kernel is a good choice for initial experiments according to [wHcCjL03]. For more information on the kernel trick, see Section 4.1.3.

Evolutionary parameter optimization was used to determine the parameters for this SVM setup, which are the parameter of the Gaussian kernel γ and the cost parameter C for misclassification penalty in C-SVC. Because SVM training is quite computation time intensive, not only the number of cross validation runs had to be reduced to run experiments in reasonable time: In addition, the evolutionary parameter optimization settings were adjusted. The maximum number of generations in the evolutionary parameter optimization has been reduced from 50 to 20. Experiments with the standard settings had to be canceled after approximately 5 days of continuous running. Even with these settings, experiments with many examples about 1.5 days.

4.3.3. Experiments

This chapter evaluates classification based on the power set multi-label transformation method, described and selected in Section 3.2.1, without and with the label-reduction approach developed in Section 3.3.3. For this evaluation, the standard classifiers kNN, Naive Bayes and SVM (Section 4.1) have been used, as well as the custom class representative based classification approach (Section 4.2). The leading questions for this evaluation are:

1. How well do the selected classification approaches perform in the scenario of this thesis?
2. Does label reduction improve the classification quality, as expected?
3. Are misclassifications in general completely incorrect or do partial misclassifications occur?
4. How does the custom classification approach perform, in comparison to the selected standard classifiers, especially in respect to kNN?

In order to answer these questions, experiment results are presented in the following. These have been run on five user-based data sets, which were sampled from the Twitter web services (Appendix A).

Multi-Label Classification

The first experiment series is dedicated to evaluating the selected standard classifiers – kNN, Naive Bayes and SVM – on basis of the power set multi-label transformation method. This technique transforms a multi-label data set into a single-label variant by replacing each distinct label set with a unique, single label. Details on the transformation have already been presented in Section 3.2.1.

Figure 4.7 shows the accuracy of the evaluated classifiers for each of the five user samples, as assessed by 10-fold cross validation (Section 4.3.1). In the background, the blue bar visualizes the number of power set labels generated for the specific data set. The green bar shows the number of labeled examples, i.e. such that can be used for evaluation. Both in relation to the right value axis.

The chart shows, that classifiers perform generally worse on data sets with many labels (Reference and User 3), than on such with fewer labels. Furthermore, the number of available training examples seems to have influence to the classification accuracy. Both is expected. It has to be noted, that the data set User 4 consists of really few labeled examples (20), but also has only 4 different labels.

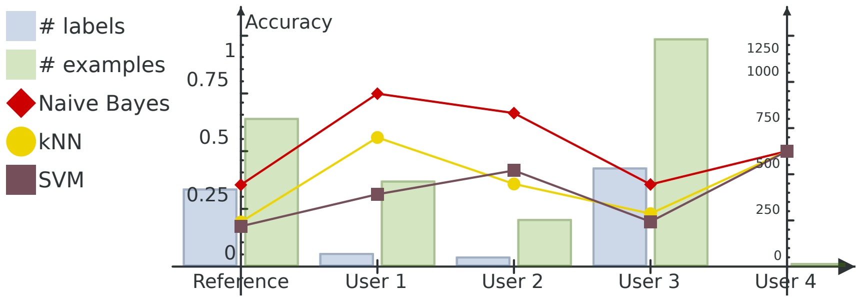


Figure 4.7.: Accuracy of the standard classifiers on power set labels.

Interestingly, the SVM performs not as well as other classifiers, although the determined parameters indicate severe fitting (see Table 4.1), according to [KL03]. Only on the data set User 2, SVM outperforms the kNN classifier. As expected in Section 4.1.2, the Naive Bayes classifier performs very well compared to the others, yielding an accuracy of up to 0.75 on a data set with few number of labels. Overall better performance on data sets with fewer labels already fortifies the assumption that label reduction will increase classification quality.

In order to assess if the label reduction method, developed in Section 3.3.3, is feasible to increase classification accuracy, a second experiment series has been run with the same setup, but involving label reduction. Figure 4.8 visualizes the results. The results of classification on the example set without label-reduction, presented in Figure 4.7, are given as a reference in pale color.

The experiment shows that reducing the number of labels indeed raises classification accuracy. This effect is more intensive on data sets with a large number of labels, as expected. As already presented in Section 3.3.4, the data sets Reference and User 3 experienced a significant reduction by 0.57 and 0.39. In contrast to that, for the data sets User 2 and 3 the number of labels was only reduced by a very small fraction. User 4 contains only 4 labels and no reduction is possible, therefore no effect to classification accuracy is visible.

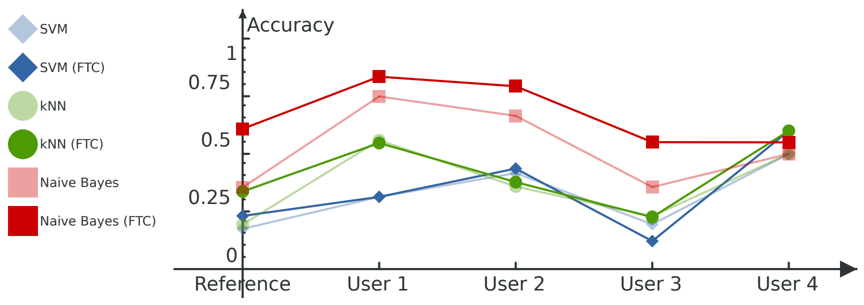


Figure 4.8.: Accuracy of the standard classifiers on reduced power set labels.

The Naive Bayes classifier is most sensitive to label reduction. The rationale for this is assumed to be, that with fewer labels, the corresponding conditional probability values (see Section 4.1.2) become more margined, which leads to a clearer division. The kNN classifier does not perform comparably better on the reduced label data sets. Still, a slight improvement is visible on the Reference data set. Although data set User 3 also experienced a significant label reduction, kNN does not perform significantly better. It is assumed, that this is a result of the data points not being linearly separable. In this case, kNN might not find a correct majority decision among the nearest neighbors. Details on the function of the kNN classifier have been presented in Section 4.1.1.

The optimized parameters - k for kNN, C and γ for the SVM (see Section 4.3.2) - are presented in Table 4.1.

		Reference	User 1	User 2	User 3	User 4
SVM	C	∞	∞	∞	∞	∞
	γ	$1.8e+7$	$2.2e+7$	$5.6e+7$	$5.6e+7$	$5.6e+7$
SVM (FTC)	C	∞	∞	∞	∞	∞
	γ	$6.2e+7$	$5.6e+7$	$2.2e+7$	$6.2e+7$	$5.6e+7$
kNN	k	221.00	70.00	221.00	70.00	221.00
kNN (FTC)	k	70.00	70.00	221.00	70.00	221.00

Table 4.1.: Overview on parameters for SVM and kNN, with and without label reduction.

Accuracy vs. α-Accuracy

In order to determine if classifications are generally incorrect or if there exist partial inaccuracies, for the experiments described in the last Section 4.3.3, the α-accuracy (see Section 4.3.1) has been evaluated as well. The alpha accuracy does not only distinguish between correct and incorrect classification, but is adjusted to the multi-label environment: It scores a classification result by its degree of correctness. This takes labels into account, which have not been predicted, but occur in the true label set and as such have been predicted false positively.

Table 4.2 presents the accuracy and α-accuracy values for the standard classifiers on the power set labeled data set and the data set with reduced labels. As can bee seen, both accuracy measures do not vary significantly. For the power set labels, the α-accuracy is occasionally better than the classical measure, indicating that there are some partly corrected classifications. On the reduced label set, the measures result in exactly the same values, at least for the first two decimal places. This appears to mirror the label reduction mechanism exactly: Multiple label sets with similar labels are replaced with a single frequent term set. For details on this method, please refer to Section 3.3.3.

		Reference	User 1	User 2	User 3	User 4	Avg.
		Power set labels					
Naive Bayes	(acc.)	0.35	0.75	0.66	0.36	0.50	0.52 ± 0.16
	(α-acc.)	0.38	0.75	0.68	0.38	0.55	0.55 ± 0.15
kNN	(acc.)	0.19	0.56	0.36	0.23	0.50	0.37 ± 0.14
	(α-acc.)	0.22	0.56	0.36	0.24	0.55	0.39 ± 0.15
SVM	(acc.)	0.17	0.31	0.42	0.19	0.50	0.32 ± 0.13
	(α-acc.)	0.18	0.31	0.42	0.20	0.55	0.33 ± 0.14
		Reduced power set labels					
Naive Bayes	(acc.)	0.61	0.83	0.79	0.55	0.55	0.67 ± 0.12
	(α-acc.)	0.61	0.83	0.79	0.55	0.55	0.67 ± 0.12
kNN	(acc.)	0.34	0.55	0.38	0.23	0.60	0.42 ± 0.14
	(α-acc.)	0.34	0.55	0.38	0.23	0.60	0.42 ± 0.14
SVM	(acc.)	0.23	0.31	0.44	0.12	0.60	0.34 ± 0.17
	(α-acc.)	0.23	0.31	0.44	0.12	0.60	0.34 ± 0.17

Table 4.2.: Accuracy values vs. α-accuracy.

Custom Classification Approach

Finally, an experiment series with the custom classification approach developed in Section 4.2 has been performed on both multi-label transformation approaches: Pure power set and label reduced power set transformation. The developed classifier performs 1NN classification on a condensed representative per class. In order to generate a representative for the examples of a specific class, three different approaches have been considered: Calculating the average vector, choosing the median example and choosing well scattered points.

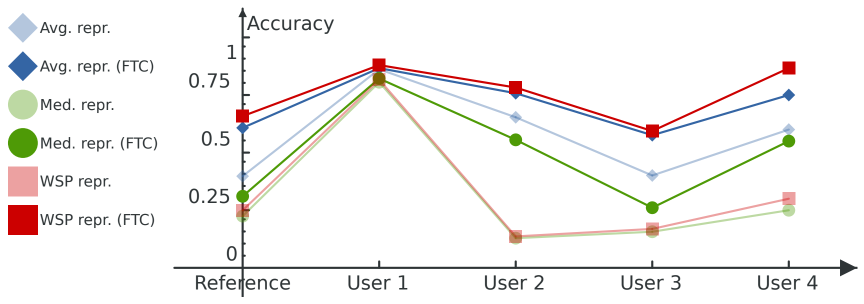


Figure 4.9.: Accuracy of the developed classification approach on reduced power set labels. Results on pure power set labels are in pale color.

Figure 4.9 visualizes the results of the different representatives. In pale color, the accuracy on basis of the pure power set method is shown, strong colors show the results on basis of the reduced power set labels. Results on basis of the average vector representative are shown in red, the median representative appears green, and the technique of well scattered points is shown in brown. Without label reduction, the average vector representative outperforms both others clearly. Also interestingly, the median and the well scattered points representatives are about equally accurate without label reduction.

As expected, all representative generation methods perform better with reduced number of labels. The average vector representative still shows top performance after the number of labels has been reduced. Interestingly, in this setup the well scattered points representative even slightly outperforms the average centroid. However, the generation of the average centroid involves less computational effort and is therefore considered further in this thesis. The tiny enhancement yielded by the well scattered points representative is not considered to compensate the higher computation time and storage consumption.

Figure 4.10 finalizes the evaluation, showing both classification approaches with the highest accuracy on the label reduced data sets: The Naive Bayes classifier from the set of standard classifiers and the custom representative based classification approach with average vector representative. Interestingly, both classification

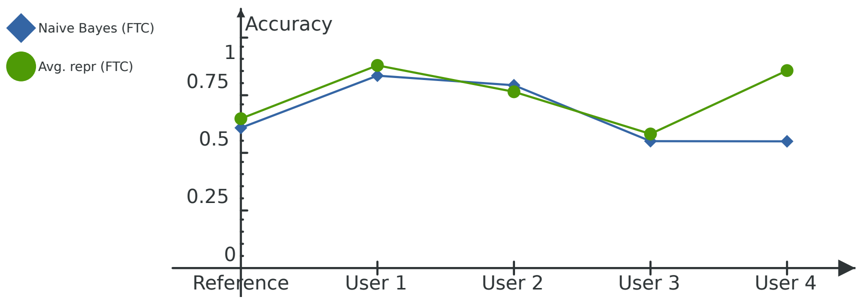


Figure 4.10.: Naive Bayes vs. custom classification approach on label reduced data sets.

approaches are about equally accurate on four of five data sets: The 1NN approach on class representatives outperforms Naive Bayes slightly on the data sets Reference, User 1 and User 3, while the Naive Bayes classifier performs slightly better on the User 2 data set. Interestingly, the difference in classification accuracy is much wider for the User 4, Naive Bayes has only an accuracy value of 0.55 while the classification approach developed in Section 4.2 yields 0.85. The result of this is an average accuracy of 0.75 ± 0.12 for the custom classifier and 0.67 ± 0.12 for Naive Bayes.

Chapter 5

Streaming

The problem of tag prediction in micro-blogging systems is a multi-label classification task, for which Chapter 3 selected the power set method, in order to transform it into a single-label problem. Furthermore, a custom approach for reducing the number of resulting labels has been developed in this chapter. Chapter 4 evaluated three standard classifiers - kNN, Naive Bayes and SVM - on this basis. In addition, a custom classification approach, based on kNN, applied on a condensed version of the data set, which involves the generation of a representative set of vectors per class. Experiments turned out, that the Naive Bayes approach, as well as the custom classification approach, perform best in the given scenario.

In Section 2.3.2, it has been noted that a classical batch algorithm could be used to implement tag prediction for micro-blogging in form of a web service. However, it is desired to create a stream classification approach instead, in order to perform anytime prediction.

Stream based data mining and machine learning are hot topics in research for the past years. Typical applications for data stream models include sensor data processing ([str02]) and log analysis ([BJC$^+$04]). The motivation for a stream processing approach is usually the large amount of data to be processed and the velocity of its occurrence, which renders it impossible to store the whole data set to perform multiple database runs or to keep it in memory for random access.

This motivation is not immediately evident in the environment of this thesis: Micro-blogging entries are considered on a per user basis and while some users produce a lot of entries per day, this amount of data does not reach dimensions where storage and multi-pass processing is unfeasible. However, the objectives defined in Section 2.3 prescribe the developed approach to be usable in a web service environment. It cannot be expected that the users of such a web service provide a sensible amount of example data to train a fully functional classifier. Furthermore, it is expected that users tagging behavior changes over time and the trained classifier should adjust to these changes. Therefore an anytime algorithm is desired, which is capable of classifying a new example at any given point in time, even if not with full accuracy.

For these reasons, it has been attempted to embed the classification approaches selected and developed throughout this thesis into a data stream environment, in order to approach the following demands:

1. Successively train the classifier in an anytime manner to allow recommenda-

tions whenever requested.

2. Detect changes in the tagging behavior of a user and adjust the classifier accordingly.

The subsequent structure of this chapter is as follows: The next Section 5.1 gives a general introduction into the stream data model used in this thesis and presents a prototype stream classification algorithm. This algorithm only provides a template, which is to be filled with three different model update strategies, in order to evaluate stream classification possibilities (Section 5.3) for tag prediction in micro-blogging services. The first strategy is very simple and deals as the base line: Here, a new classification model is trained for every stream window. The second approach attempts to update a once trained classification model continuously with new stream data. Finally, the third strategy is most complex and attempts to detect changes in users tagging behavior using the StreamKrimp algorithm by van Leeuwen and Siebes ([vLS08]), in order to adjust the classifier correspondingly. The StreamKrimp algorithm is described in detail in Section 5.2, then Section 5.3 describes the three stream classification approaches end presents an empirical evaluation of these approaches, in combination with the two best performing classification approaches from Chapter 4: Naive Bayes and the custom developed representative based classification technique.

5.1. Stream Basics

To specify stream based classification, Definition 6 already presented the formal definition of a data stream in Section 2.3.2. A data stream is a finite or infinite set of examples $S = \{e_1, \dots, e_n, \dots\}$, which are sorted in an ascending, typically temporal, order $\prec_S$, such that

$$\forall e_i, e_j \in S : i < j \Rightarrow e_i \prec_S e_j$$

holds.

To model the processing of data streams, a window approach is typically used. Such a *stream window* defines, which portion of a stream can be accessed by a processing algorithm at a given point in time, in order to model the infeasibility of random access to stream data. Definition 29 specifies the syntax for stream windows used within this thesis.

Definition 29 (Stream window):
A window $S_{i,j} := \{e_i, \dots, e_j | e_x \in S\}$ on a stream S is a set of $(i - j)$ consecutive examples from S, defined by two end points i and j. A single example from a stream is denoted by $S_i := e_i$.

Stream windows are occasionally denoted as a finite *sub-stream*. The evolution of a stream can be emulated by changing the window end points in a positive direction. In [GÖ03], different types of windows over data streams are characterized by several criteria as follows:

Movement	A *fixed window* is defined by two fixed end points, while two moving end points define a *sliding window*. In each step i and j are usually incremented by the same amount. A window where only one end point moves is called a *landmark window*.
Physical vs. logical	Windows on data streams can either be updated by time or by element count. *Physical*, or time-based, windows are updated after a certain amount of time elapsed. The number of new examples in the window may vary. Logical windows are, in contrast to that, updated whenever a certain amount of new examples arrive in the stream. The time between window updates may vary.
Update interval	If a window is updated on each arrival of a new example, this is called *eager re-evaluation*, while updates based on an example batch are denoted as *lazy re-evaluation*. The latter update procedure in general induces a *jumping window*, if the size of the update batch matches the window size. If the update batch exceeds the window size, a series of non-overlapping *tumbling windows* is produced.

Figure 5.1 visualizes the different described stream window approaches.

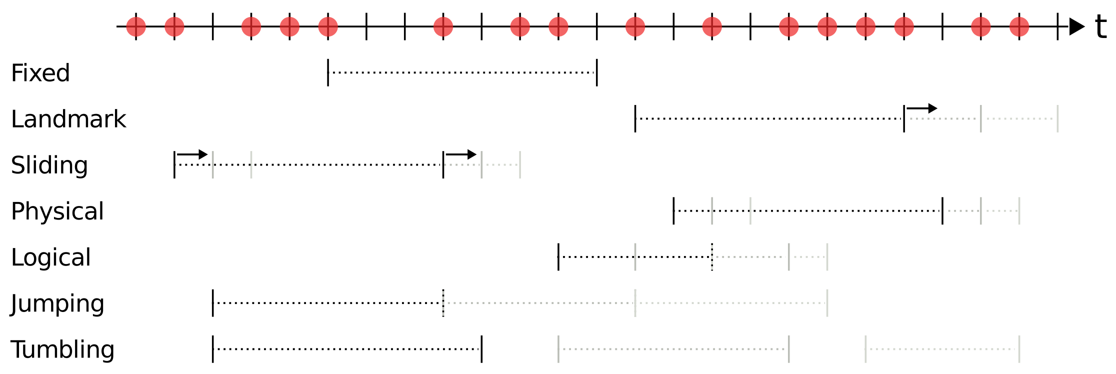


Figure 5.1.: Different stream window models

In scope of this thesis, a logical jumping window approach is considered: Whenever a defined number of examples have arrived, the classification model is to be updated. The decision for a jumping window is based on the label-reduction technique developed in Section 3.3.3, which relies on frequent item set mining. On this basis, a continuous update of the classification model is infeasible, since it prevents a proper label reduction. Furthermore, a time based update is considered infeasible, since the update frequency varies widely among users.

On this basis, Algorithm 6 presents a template for simple stream classification. This template is used to realize and evaluate three different stream classification approaches in the following. The function STREAMCLASSIFICATION() function retrieves a stream and a logical window size. It first initializes an empty model, since no examples have been received for training, yet. During the subsequent iteration over the stream, each new example is classified. The practical response of a predicted tag set to the user is indicated by the NOTIFY() function call. When ever *winSize* examples have been buffered, the classification model is updated within UPDATEMODEL(). This function is used later in this chapter to implement different model update strategies.

Algorithm 6 Template stream classification algorithm.

```
function StreamClassification(stream $S$, int $winSize$)
    $i := 0$
    $j := 0$
    $h :=$ EmptyModel()
    while $\neg$StreamEnd($S_j$) do
        Notify(h($S_j$))
        if $j = i + winSize - 1$ then
            UpdateModel($h, S, i, j$)
            $i := i + winSize$
        end if
        $j := j + 1$
    end while
end function
```

It has to be noted, that this algorithm suffers from the so-called *bootstrap* problem, i.e. no classification model is available until the first stream window is completely processed. Finding a solution to this problem is left for later research.

Based on the requirements presented in the introduction of this chapter, the StreamKrimp algorithm by van Leeuwen and Siebes [vLS08] has been selected in order to detect changes in tagging behavior of a user. STREAMKRIMP attempts to detect changes in the probability distribution of frequent item sets on a stream, based on the *minimum description length* (MDL) principle. In order to assess changes in tagging behavior, StreamKrimp is applied to the label sets in scope of this thesis.

Two main reasons motivate the choice of StreamKrimp: Firstly, it is expected that changes in the probability distribution of tag sets indicate a change of tagging behavior. Secondly, frequent item sets are already used for the label reduction technique developed in Section 3.3.3, so re-using them appears sensible.

In order to assess the feasibility of StreamKrimp to detect changes in the tagging behavior and therefore yield a potential classification quality enhancement, the corresponding model update strategy will be compared to two simply stream approaches: The first one returns a new model, trained on the recent stream window, on every call to UPDATEMODEL() call. The second simple variant trains a new model only on the first window and updates it with the training data of every subsequent window.

5.2. Detecting Changes in Streams

One essential goal of this chapter is to attempt detection of changes in the tagging behavior of a user. It is expected, that detecting such changes and adjusting classification models accordingly can yield an improvement of overall classification quality. In order to detect changes in the tagging behavior, the StreamKrimp algorithm by van Leeuwen ans Siebes ([vLS08]) is applied to the label sets.

The StreamKrimp algorithm attempts to detect changes in the probability distribution of frequent item sets in a stream. The algorithm utilizes the concept of minimum description length and a there implied coherence between probability distribution and optimal encoding. StreamKrimp works on a logical jumping window model, examining a fixed number of newly arrived examples at a time. The algorithm utilizes the Krimp algorithm ([SVvL06]) to detect the distribution of item sets in stream data.

In following, the Krimp algorithm is explained first, introducing the concept of encoding a transactional database through frequent item sets and the relation between optimal encoding and probability distribution. After that, the StreamKrimp algorithm is explained on that basis.

5.2.1. Krimp

The original motivation of the Krimp algorithm was to select an interesting subset of frequent item sets, because a frequent item set analysis yields typically huge number of item sets, including a large amount of redundant data. This behavior is closely related to the monotonicity property of frequent item sets (Section 3.3.1), i.e. if an item set is frequent, all its subsets are frequent, too. There exist several other approaches that attempt to prune sets of frequent item sets in order to make human inspection more comfortable, for example lift [ST96] and constraining item sets to be *closed* [Zak98].

In order to select a subset of frequent item sets, the Krimp algorithm attempts to encode the corresponding transactional database through it. Encoding in this case means, that a transaction is replaced by references to the item sets it consists of. To measure if a selected set of item sets is well chosen, Krimp utilizes the approach of minimum description length.

The minimum description length principle can roughly be described as follows: From a set of models $\mathcal{H}$, the optimal model $H \in \mathcal{H}$ is the one that minimizes

$$\mathrm{L}(H) + \mathrm{L}(D|H) \tag{5.1}$$

where L denotes a function which returns the length in bits, which are necessary to describe a given object. That means $\mathrm{L}(H)$ is the bit-length of the description of the model H and $L(D|H)$ is the length of a database D when being encoded using the model H. If the sum of the description length of the model and the data encoded in the model is minimal, the model is considered optimal. Extensive information on the principle of minimum description length can be found in [Grü05].

The Krimp algorithm attempts to encode a given transactional database through frequent item sets mined from it. That means, a set of frequent item sets is chosen, which allows to replace every transaction in the database by a set of references to frequent item sets. The goal is to find a subset of all frequent item sets, which encodes the given database optimally. In the sense of MDL, the model here consists of a set of frequent item sets and some kind of code on it. A transactional database can then be encoded by this model as defined in following. In order to satisfy MDL, the subset of frequent item sets is to be found, which minimizes the description length as defined in Equation 5.1.

Encoding a Database

In order to define the encoding of a transactional database on the basis of frequent item sets, Siebes et al. specify the term *cover* differently, compared to the definition given earlier in Section 3.3.1:

Definition 30 (Item set cover)**:**
Let $\mathcal{C} \subseteq \mathcal{I}$ be a set of frequent item sets from a transactional database D. $\mathcal{C}$ is called an item set cover for D, iff for each transaction $(t, I_t) \in D$, a subset $\mathcal{C}(t) \subseteq \mathcal{C}$ exists for which

1. $I_t = \bigcup_{I \in \mathcal{C}(t)} I$
2. $\forall I_x, I_y \in \mathcal{C}(t) : I_x \neq I_y \Rightarrow I_x \cap I_y = \emptyset$

holds. It is said that $\mathcal{C}(t)$ covers t.

Definition 30 specifies the conditions that must be fulfilled by a subset of item sets $\mathcal{C} \subseteq \mathcal{I}$ in order to encode a transactional database D: $\mathcal{C}$ must be such that each transaction $(t, I_t) \in D$ can be replaced by references to a number of item sets in C, without loss of information. In order to define the actual encoding of a database, the principle of a *coding scheme* is introduced in Definition 31.

Definition 31 (Coding scheme):
A coding scheme CS for a transactional database D is a tuple $(\mathcal{C}, \mathrm{S})$, where $\mathcal{C}$ is an item set cover for D and S *is a function* $\mathrm{S} : D \rightarrow \mathcal{P}(\mathcal{C})$ *such that* $\mathrm{S}(t)$ *covers t.*

So, a coding scheme consists of a set of item sets $\mathcal{C}$, which covers a database and a function S, which maps each transaction to the set of item sets which are necessary to encode it. Obviously, this construct can be used to encode the transactional database, for example by assigning a unique integer number to each item set and replacing each transaction with the corresponding numbers determined by S.

However, Krimp does not search for an arbitrary coding set, but the one which yields the smallest compression in terms of MDL. In order to measure the size of the desired encoded database, a correspondence between codes and probabilities is exploited: If P is a probability distribution on a finite set K, a unique code exists for K, such that for the length of an element $L(k), k \in K$

$$\mathrm{L}(k) = -\log(\mathrm{P}(k)) \tag{5.2}$$

holds. This code does not only exist, but is also known to the be optimal for any dataset which is drawn according to P. A coding set actually induces a probability distribution on its set of frequent item sets through the mapping function S. The distribution is determined by the relative frequency of an item set to be used to cover a transaction.

$$\frac{\mathrm{freq}(I)}{\sum_{I' \in \mathcal{C}} freq(I')}$$

Replacing this probability into Equation 5.2 and adding up over all item sets yields the size of the database when being encoded by an optimal code on the item sets:

$$\mathrm{L}_{(\mathcal{C},\mathrm{S})}(D) = -\sum_{I \in \mathcal{C}} freq(I) \log\left(\frac{\mathrm{freq}(I)}{\sum_{I' \in \mathcal{C}} freq(I')}\right) \tag{5.3}$$

While this does not yield a concrete code, it allows to calculate the size of the encoded database when being encoded with such a code, which allows to calculate the part $\mathrm{L}(D|H)$ of the MDL formula in Equation 5.1.

Encoding a Model

$L(D|H)$, from the MDL formula in Equation 5.1, can now be calculated, given an optimal code over item sets. Still the term $L(H)$, which denotes the description length of the encoding model, is to be calculated. One part of an encoding model

H has already been described in the last section: A coding scheme. However, the code itself is still missing.

One way to describe a model would be to use a specific *encoding / decoding table*, which maps each item set from a coding scheme to all transactions which need the item set to be encoded. But, listing all transactions encoded with a given item set would lead to quite a large code table, utilizing at least $n \log(n)$ bits for a database of size n: n times $\log(n)$ bits for the pointer to each transaction. In addition, such a table does not yet describe the actual code in terms of a code word corresponding to an item set.

To ship around these issues, Siebes et al. introduce the simple algorithm Cover() (Algorithm 7). The algorithm presumes an arbitrary order $<$ on the given set of item sets $\mathcal{C}$. Furthermore, it requires that $\mathcal{C}$ contains at least all singleton item sets:

$$\forall I \in \mathcal{I} : |I| = 1 \Rightarrow I \in \mathcal{C}$$

Such a set of item sets is denoted as a *coding set* (Definition 32).

Definition 32 (Coding set)**:**
An ordered set of item sets, which contains at least all singleton item sets, i.e. $|I| = 1$*, is called a coding set. Its order, referred to as coding order is denoted by* $<$*.*

Algorithm 7 is not only well-defined on any coding set $\mathcal{C}$ and any transaction (t, I_t) over the same items, it also defines a unique code table (Definition 33), given a fitting order in the coding set.

Algorithm 7 The algorithm encodes a transaction uniquely on basis of a coding set, by recursively selecting the first matching item set until the full transaction is covered.

```
function Cover(coding set $\mathcal{C}$, transaction $(t, I_t)$)
    $S := \min\{I \in \mathcal{C} | I \subseteq I_t\}$
    if $I_t \setminus S = \emptyset$ then
        $Res := S$
    else
        $Res := S \cup \text{Cover}(\mathcal{C}, (t, I_t \setminus S))$
    end if
    return $Res$
end function
```

Definition 33 (Code table)**:**
Given a coding set $\mathcal{C}$ *and a set of codes* K*.* $CT \subseteq \mathcal{C} \times K$ *is a code table iff*

- $\forall (I_x, k_x), (I_y, k_y) \in CT : I_x = I_y \Rightarrow k_x = k_y$
- $\forall (I, k) \in CT : L(k) = -\log(\mathrm{P}(I))$

where $\mathrm{P}(I)$ *is the probability of* I *in the code scheme for database* D*, induced by* $\mathcal{C}$ *and* Cover()*.*

In terms of minimum description length, a code table is a model, which can be used to encode a given database. In order to calculate the MDL formula (Equation 5.1), the size of such a model needs to be calculated. The size of the right side of the code table is obvious: Each assigned code has a specific length, denoted by $-\log(\mathrm{P}(I))$, which must only be summed up for all codes. Leaves the left side: In order to encode the item sets used for encoding the database, Siebes et al. define the so called *standard encoding* (Definition 34), which consists of a code table that only utilizes singleton item sets.

Definition 34 (Standard encoding)**:**
The standard encoding for an item set I *for a given database* D *over* $\mathcal{I}$ *is the encoding defined by the coding set* $\mathcal{C}_{St} := \{\{i\} | i \in \mathbb{I}\}$ *of all singleton item sets.*

Obviously, the standard encoding is the same for all potential coding sets over a certain database. It therefore makes the encodings comparable. On this basis, the description length for a code table CT, i.e. a model, can be defined as:

$$\mathrm{L}(CT_{\mathcal{C}}) = \sum_{I \in \mathcal{C}:\mathrm{freq}(I) \neq 0} (\mathrm{L}_{\mathrm{st}}(I) + \mathrm{L}_{\mathcal{C}}(I)) \tag{5.4}$$

where $\mathrm{L}_{\mathrm{st}}(I)$ denotes the length of the item set I when being encoded by the standard encoding. $\mathrm{L}_{\mathcal{C}}(I)$ determines the length of the code assigned to an item set I in the code table CT.

This finally yields the full description length needed to calculate the MDL formula in Equation 5.1:

$$\mathrm{L}_{\mathcal{C}}(D) = \mathrm{L}_{(\mathcal{C},\mathrm{S_C})}(D) + \mathrm{L}(CT_{\mathcal{C}}) \tag{5.5}$$

The description length of the encoded database has been presented in Equation 5.3, while the description length for the encoding model itself has been presented in Equation 5.4.

Minimal Coding Set Heuristic

Now that the calculation of the description length has been specified in Equation 5.5, the problem of finding a code for a given database is tackled. The goal of the presented algorithm is to find a coding set $\mathcal{C}$ for which $L_{\mathcal{C}}(D)$ is minimal. Since almost any arbitrary subset of the set of frequent item sets is allowed as a coding set, the size of the search space is bound by $O(2^n)$, where n is the number of frequent item sets. Furthermore, a coding set has been defined as being ordered (Definition 32), so a fitting order needs to be found in addition to the item sets themselves. This yields an upper bound of $O(n!)$ for possible orders. Resulting in the size of the search space to be bound by $O(2^n n!)$.

Algorithm 8

```
function NaiveCompression(items $\mathcal{I}$, item sets $\mathcal{C}$, database $D$)
    $CS := \text{Standard}(\mathcal{I}, db)$
    $\mathcal{C} := \mathcal{C} \setminus \mathcal{I}$
    $candidates := \text{CoverOrder}(\mathcal{C}, D)$
    while $candidates \neq \emptyset$ do
        $cand := \max(candidates)$
        $candidates := candidates \setminus \{cand\}$
        $canCS := CS \cup \{cand\}$
        if $\text{L}_{\text{canCS}}(D) < \text{L}_{\text{CS}}(\text{D})$ then
            $CS := canCS$
        end if
    end while
    return $CS$
end function
```

In [SVvL06], four different algorithms are defined for the purpose of finding an optimal coding set, while this thesis considers only the very basic version to determine the general feasibility of the StreamKrimp approach for the problem of detecting changes in tagging behavior. The greedy approach NaiveCompression() (Algorithm 8) starts with the standard coding set and subsequently tests if adding a specific item set yields a better compression. If the description length decreases, the item set is kept, otherwise it is discarded.

The crucial point in Algorithm 8 is to determine the place in the code table, where a new item set should be inserted. Determining the optimal place could for example happen on basis of the Cover() function, but this would yield another huge search space to be explored. Therefore the *standard order* of item sets in a coding set is defined by Siebes et al.

Definition 35 (Standard order)**:**
Given a database D and a set of item sets $\mathcal{C}$. $\mathcal{C}$ is in standard order, iff for all $I_1, I_2 \in \mathcal{C}$ holds

- $|I_1| \leq |I_2| \Rightarrow I_1 \preceq_C I_2$
- $|I_1| = |I_2| \wedge \text{support}(I_1, \mathcal{D}) \leq \text{support}(I_2, \mathcal{D}) \Rightarrow I_1 \preceq_C I_2$

Siebes at al. concede that the standard order (Definition 35) is just a heuristic, but any better ordering technique would involve far more computational effort. In Algorithm 8, the CoverOrder() function returns a coding set on basis of standard order. The NaiveCompression() algorithm is used in this thesis, in combination with the StreamKrimp algorithm, to detect changes in the probability distribution of label sets on a stream.

5.2.2. StreamKrimp

The heuristic Krimp algorithm, presented in the last Section 5.2.1, attempts to determine the set of frequent item sets, which compress a given database optimally in a minimum description length sense. Siebes et al. have shown empirically in [SVvL06], that this approach works rather well. The StreamKrimp algorithm, which is to be explained in the following, is based upon Krimp. It utilizes the correspondence between optimal codes and probabilities, noted in Equation 5.2, to determine changes in the probability distribution of frequent item sets on a stream. The basic idea here is, that if the optimal coding set changes on subsequent sub-streams, a change in the distribution is inherent.

More formally, it is assumed, that a given stream S consists of a, potentially infinite, set of sub-streams $S = S^1 S^2 S^3 \ldots$, for which holds that each S^i is drawn i.i.d. from a distribution P_i over $\mathcal{P}(\mathcal{I})$ and $\forall i \in \mathbb{N} : P_i \neq P_{i+1}$. The StreamKrimp algorithm attempts to identify the S_i, i.e. to detect subsequent sub-streams of S, such that each sub-stream conforms to a dedicated distribution, which is different from the distribution of its predecessor and successor sub-streams.

In order to achieve this goal, the StreamKrimp algorithm must essentially fulfill the following two requirements.

1. Detect the distribution on a sub-stream.

2. Detect the point where this distribution changes.

In the following, it is described how StreamKrimp honors these requirements.

Detecting Distributions

Siebes et al. first consider the problem of detecting distributions on finite streams. Equation 5.2 showed, that a probability distribution can be reflected through optimal coding. Therefore, the problem can be reformulated as follows:

A set of consecutive sub-streams on a finite stream $S = S^1 \ldots S^k$ is to be detected, such that

$$\sum_{i=1}^{k} \mathrm{L}(CT^{opt}, S^i)$$

is minimized. $\mathrm{L}(CT^{opt}, S^i)$ denotes the length of the encoded code set, consisting of S_i and its optimal code table CT^{opt}, as described in Equation 5.5. Practically that means to partition the given stream, such that the over all description length of S is minimized.

However, this formulation assumes that the whole stream S is known. It is generally not possible to project this minimization problem directly to the infinite case, because there is no guarantee that an optimal partitioning of a finite stream $U = ST$ coincides with an optimal partitioning for S on the S-part of U [vLS08].

StreamKrimp therefore concentrates on a locally optimal solution and utilizes the fact that the empirical probability distribution converges against the real distribution: For a stream S, which is i.i.d., drawn from a single distribution Q,

$$\forall I \in \mathcal{I} : \lim_{n\to\infty} P(I|S_{1,n}) = Q(I)$$

holds. Here, $P(I|S_{1,n})$ denotes the empirical probability for an item set $I \in \mathcal{I}$ on the first n items of stream S. $Q(I)$ denotes the items real probability distribution. From this observation and the correspondence between probability and optimal coding (Equation 5.2) follows, that the optimal coding table on stream S converges as well.

In order to measure when a code table can be considered as converged, Siebes et al. introduce the *improvement rate* measure, which is specified as follows:

Definition 36 (Improvement rate)**:**
The improvement yielded by a code table CT_{n+k}, generated on a sub-stream $S_{1,n+k}$, in comparison to its predecessor CT_n is defined by

$$\mathrm{IR}(CT_n, CT_{n+k}) = \frac{|\mathrm{L}(S_{1,n}, CT_n) - \mathrm{L}(S_{1,n}, CT_{n+k})|}{\mathrm{L}(S_{1,n}, CT_n)}$$

Practically, when the improvement rate becomes small in an absolute sense, it may be concluded that CT_n has converged and that the distribution of the sub-stream has been detected to a sufficient degree. Algorithm 9 provides a function, which is used in the StreamKrimp algorithm for the purpose of detecting a converged code table.

Algorithm 9 converges a Krimp code table on a stream S, starting from position *start*. The *blockSize* determines the jumping window size. The Krimp() function here wraps NaiveCompression() (Algorithm 8) or any other Krimp algorithm described in [SVvL06], and performs the mining of item sets more frequent than sup_{min} with a suitable algorithm, e.g. FPGrowth.

Detecting Distribution Changes

Once the distribution of a sub-stream has been detected, the next requirement is to detect when this distribution changes on the stream. On the basis of the

Algorithm 9 Converging a Krimp code table on a stream.

```
function FindCodeTableOnStream(stream $S$, int $i$, int $blockSize$, int $sup_{min}$, float $maxIR$)
    $n := blockSize$
    $CT :=$ Krimp$(S_{i,i+n}, sup_{min})$
    $ir := \infty$
    while $ir > maxIR$ do
        $n := n + blockSize$
        $newCT :=$ Krimp$(S_{i,i+n}, sup_{min})$
        $ir :=$ ComputeIR$(CT, newCT)$
        $CT := newCT$
    end while
    return $codeTable$
end function
```

convergence of code tables, deduced in the last section, the naive idea would be to generate coding tables on consecutive stream chunks and compare their encoding quality to the previously converged code table. However, this would consume a fair amount of computation time. Therefore, Siebes et al. apply a statistical test, which compares the encoded size of a subsequent transaction block to the size expected for a block of this size.

After a code table converged, the sub-stream used to converge it is not immediately discarded. Instead, random chunks of the defined block size are sampled from it and their encoded size is computed. After a lower and upper leave-out fraction is removed from the resulting set of sizes, the smallest and largest sizes are taken as boundaries for the encoded size of subsequent chunks. If the size of a new stream chunk, encoded with the converged code table, does not break with the expected boundaries, the block is considered to belong to the previously detected distribution. Otherwise, the full test of converging a new code table is applied in order to determine if a change of the distribution is evident. The statistical test is defined for being used in the StreamKrimp algorithm in form of the so-called *code table difference* measure.

Definition 37 (Code table difference)**:**
iven code tables CT_1, CT_2 converged on two subsequent sub-streams S^1, S^2. The code table difference (CTD) is given by

$$\mathrm{CTD}(CT_1, CT_2) := \frac{\mathrm{L}(S^2, CT_1) - \mathrm{L}(S^2, CT_2)}{\mathrm{L}(S^2, CT_2)}$$

On basis of these considerations, the StreamKrimp algorithm is defined in the following.

StreamKrimp

Based on the Algorithm 9, which detects a distribution on a stream, and the CDT measure, StreamKrimp is shown in Algorithm 10. The algorithm subsequently detects the item set distribution of finite sub-streams on basis of the minimum description length principle.

Algorithm 10 StreamKrimp algorithm

```
function StreamKrimp(stream $S$, int $blockSize$, int $sup_{min}$, float $maxIR$,
float $leaveOut$, float $minCTD$)
    $i := 1$
    $CT_i :=$ FindCodeTableOnStream($S, blockSize, sup_{min}, maxIR$)
    $pos :=$ EndPos($CT_i$)
    while $pos <$ SizeOf($S$) do
        SkipBlocks($S, CT_i, pos, blockSize, leaveOut$)
        $candCT :=$ FindCodeTableOnStream($S, pos, blockSize, sup_{min}, maxIR$)
        if ComputeCTD($S, CT_i, candCT$) $> maxCTD$ then
            $i := i + 1$
            $CT_i := candCT$
            $pos :=$ endPos($CT_i$)
        else
            $pos := pos + blockSize$
        end if
    end while
    return $CT$
end function
```

The StreamKrimp algorithm (Algorithm 10) converges an initial code table (CT) on the received stream S, using Algorithm 9. Once the initial code table is found, it tries to skip as many stream blocks as possible, using the statistical test described in the previous section. If this test fails, a new code table *candCT* is converged, starting at the current stream position. If the code table difference of CT and *candCT* exceeds $maxCTD$, a distribution change is assumed to be detected and *candCT* becomes the new actual code table. In case the stream ends, the last code table is returned. However, this point will not be reached on infinite streams.

Siebes et al. have shown empirically in [vLS08], that the StreamKrimp algorithm is indeed capable of detecting changes in the distributions of frequent item sets on streams. In the following, the StreamKrimp algorithm is embedded into the stream classification algorithm used within this thesis (presented in Section 5.1) to detect changes in the tagging behavior of a user.

5.3. Stream Based Classification

Section 5.1 proposed a template algorithm for stream classification. The Algorithm 6 attempts to train an initial classification model on the first stream window. This model is then used to classify examples on subsequent windows, and updated on the ground truth labels afterwards of the examples afterwards. The template algorithm defined a function UpdateModel(), which is responsible for updating the classification model. In following, this function will be used to realize three different model updating strategies, in order to complete the definition of stream classification approaches.

The following Section 5.3.1 presents the three strategies used to evaluate stream classification in the area of tag prediction in micro-blogging services. After that, experiment results are presented in Section 5.3.2.

5.3.1. Stream Classification Models

The template stream classification Algorithm 6 describes how stream classification works overall in this thesis. The algorithm definition is repeated here to ease reading.

Algorithm 11 Template stream classification algorithm.

```
function StreamClassification(stream $S$, int $winSize$)
    $i := 0$
    $j := 0$
    $h :=$ EmptyModel()
    while $\neg$StreamEnd($S_j$) do
        Notify(h($S_j$))
        if $j = i + winSize - 1$ then
            UpdateModel($h, S, i, j$)
            $i := i + winSize$
        end if
        $j := j + 1$
    end while
end function
```

The function UpdateModel() is meant to be realized using different classification model update strategies. The first strategy presented in this thesis is very simple and should deal as the baseline in experiments. Here, on each stream window a classifier is trained from scratch. The second, also simple, update strategy trains only a single model on the first stream window and updates this one on every subsequent window. The third update strategy is more complex and involves the described StreamKrimp algorithm (see Section 5.2). The approach of StreamKrimp is used, in order to detect portions of the underlying stream, which appear sensible for training a classification model. Furthermore, StreamKrimp should detect changes in the tagging behavior, in order to denote when a new model needs to be trained.

Simple Stream Classification

In order to asses if the StreamKrimp based stream classification approach can yield an enhancement, two simple stream classification approaches are used as the base line in this thesis:

In the first stream classification approach, the UPDATEMODEL() function always returns a new model, based on the training data in the last stream window. This approach is in the following denoted as the *simple stream classification approach.* It is expected, that this approach yields the worst classification quality, since each classification model returned by the update strategy is only backed by very few training examples.

The second baseline approach, denoted as the *continuous stream classification approach*, trains a new model on the first data window. Every subsequent call to UPDATEMODEL() updates the received model using the training data available in the current stream window.

Updating a Naive Bayes classification model is an easy task, since it only involves creating new counters for potentially unseen classes. For the custom class-representative based classification approach, which has been presented in Section 4.2, this task at first appears a little more complex. However, this a fallacy, since the average class representative turned out to be most accurate (see Section 4.3.3). The average vector is simple to update, as long as the number of examples contained in the average are known.

Updating the classification model is expected to inhere one significant problem: The label-reduction approach (Section 3.3.3) relies on determining frequent item sets among the label sets. Several such label sets are then replaced by a common frequent label set. For example, the label sets $M_1 = \{a, b, c\}$, $M_2 = \{a, c\}$ and $M_3 = \{a, c, d\}$ might all be replaced by M_2. In the streaming scenario, it might occur, that on a window $S_{i,j}$ the label set M_2 is chosen, but on a different window, one of the others. This would reduce the idea of label reduction to absurdity.

It is expected that the, in the following presented, StreamKrimp based classification approach yields better results, for two reasons: Firstly, it attempts to detect a number of subsequent stream windows which appear sensible for training a new classification model. Secondly, it attempts to detect when a classification model is outdated, i.e. the tagging behavior changes, and trains a new model.

Krimp Based Classification

In Section 5.2, the StreamKrimp algorithm has been explained, which attempts to identify changes in the probability distribution of item sets on a stream. In the following, this approach is embedded into the template stream classification algorithm. The StreamKrimp model update strategy attempts to detect changes in a users tagging behavior and adjust the classification model accordingly.

Algorithm 12 shows the StreamKrimp based realization of the UpdateModel() function. The purpose of this function is to update a classification model, based on the current and potentially previous stream windows. The function uses the classification model h to store additional information needed for StreamKrimp.

Algorithm 12 StreamKrimp based model update strategy.

```
function KrimpUpdateModel(model $h$, stream $S$, int $i$, int $j$)
    if IsEmpty($h$) then
        $h := \textsc{InitModel}(S_{i,j})$
        $h.CT := \textsc{Krimp}(\text{TagSets}(S_{i,j}))$
        $h.start := i$
        $h.conv := false$
    else if $\neg h.conv$ then
        $h' := \textsc{InitModel}(S_{h.start,j})$
        $h'.CT := \textsc{Krimp}(\text{TagSets}(S_{h.start,j}))$
        $h'.start := h.start$
        if $\textsc{ComputeIR}(h.CT, h'.CT) \leq maxIR$ then
            $h'.conv := true$
            $h'.bounds := \textsc{CalculateBounds}(h'.CT, \text{tagSets}(S_{h'.start,j}))$
        end if
        $h := h'$
    else if $\neg\textsc{BoundsHold}(h.bounds, \text{tagSets}(S_{i,j}))$ then
        $CT' := \textsc{Krimp}(\text{tagSets}(S_{i,j}))$
        if $\textsc{ComputeCTD}(\text{tagSets}(S_{i,j}), h.CT, CT') > maxCTD$ then
            $h := \textsc{InitModel}(S_{i,j})$
            $h.CT := CT'$
            $h.start = i$
            $h.conv = false$
        end if
    end if
    return $h$
end function
```

The model update strategy shown on Algorithm 12 makes use of the principles of StreamKrimp. In order to store information of the current Krimp state, the classification model is used. The algorithm first checks if the model is initialized at all. If this is not the, a new model is trained and a Krimp code table is generated on the basis of the label sets in the current stream window. If a model exists, but the corresponding code table is not yet converged, the code table is attempted to be converged in the manner of StreamKrimp. A new classification model is trained on the examples used for the code table. In case the code table is already converged, the statistical test of StreamKrimp is applied to determine if a change in the probability distribution of tag sets is evident. If this test fails and the code table difference between the current code table and a newly generated one exceeds $maxCTD$, the new code table is kept and a new classification model is trained. Otherwise, the converged code table still reflects the current tag set distribution and is therefore kept, as well as the current classification model.

5.3.2. Evaluation

The presented stream classification approach is in following evaluated in combination with all three proposed model update strategies, as well as both classifiers, which yielded the best performance in Section 4.3.3: Naive Bayes and the custom classification approach on basis of class representatives. Evaluation takes place on the example sets sampled from the Twitter web service, as described in Appendix A. The major goal of the evaluation is to assess if the presented stream classification approach is feasible to realize tag prediction in micro-blogging services or if a batch learning approach has to be utilized for this task..

Beside the classification accuracy of the stream based approaches, the following questions are leading for the evaluation:

- A general loss of accuracy is assumed to attend stream based classification, compared to classical classification environments. How accurate are the selected classifiers in a stream environment?
- It has been noted in Section 5.1, that the presented stream classification approach suffers from bootstrap problem. How much does this effect influence the average accuracy?
- Does the StreamKrimp algorithm detect changes in users' tagging behavior and how do these relate to classification accuracy?

The following section describes the setup for experiments in this section. Subsequent sections evaluate different experiments to answer the questions noted above.

5.3.3. Experiment Setup

Both classifiers, the Naive Bayes classifier and the custom 1NN classification on FTC cluster representatives, have been tested with all three presented stream classification approaches: The simple stream classification approach, the continuous stream classification approach and the StreamKrimp based model update strategy. The setup of the experiments in this chapter is very similar to the setup described in Section 4.3, where different classifiers have been evaluated on the basis of the multi-label to single-label transformation approach. Therefore, only changes to this setup are described in following.

Both of the classifiers selected in Chapter 4, Naive Bayes and the custom-developed average class representative based approach, do not require any parameters to be optimized. However, the stream classification algorithm requires at least the window size to be set, the StreamKrimp based variant requires even more parameter values. Although Siebes et al. provide a recommendation for these values in [vLS08], evolutionary parameter optimization, as described in Section 4.3.2, has been used to determine fitting values for the case of tag prediction in microblogging systems. The following list gives an overview on the parameters:

$blockSize$ For all stream classification strategies, this parameter is to be optimized. The target value range has been limited to $[1, 1000]$, in order to not exceed values sensible for the targeted web service environment and to ease the optimization process. In [vLS08], this setting is proposed to be set to the number of items that can occur in the stream. This is not possible in the underlying case, since tags that might occur in the future are not known beforehand, in practice.

sup_{min} The minimum support for frequent item set mining is the only parameter that has not been optimized. Since the label reduction approach (Section 3.3.3) relies on all item sets to be mined, this setting has been kept. This also satisfies the need of the Krimp algorithm to have all singleton item sets available for generating a coding set, as described in Definition 32 in Section 5.2.1.

$leaveOut$ A leave out value of 0.01 is suggested in [vLS08]. However, this value makes sense only if a large number of stream blocks are involved in the converging process of a StreamKrimp code table. Since the number of involved blocks is expected to be small, it has been chosen to optimize this parameter as well. The value range here is $[0, 0.5]$.

$maxIR$ The proposed value for the maximum improvement rate is 0.02. Because it is unknown how StreamKrimp will behave with varying items on stream blocks, this parameter has been included in the optimization process with a value range of $[0, 1]$.

$minCTD$ The minimum code table difference defines how much better a new code table must compress a stream block to induce a change of distribution.

For the same reasons as for $maxIR$, this parameter has been included in the optimization process.

The resulting set of six experiment setups – all three stream processing methods with each classifier – have been executed for all five sampled user data sets independently (Appendix A). Since the stream data model consists of an ordered set of examples, no cross validation (see Section 4.3.1) has been applied. In order to assess classification accuracy, the multi-label data set has first been prepared by the label reduction approach and then transformed using the power set transformation, to generate ground truth labels. During the stream classification, this process is also applied to the original label sets, on the basis of the current stream window. A large fraction of partly correct classifications is expected, due to the difference between applying label reduction to the full example set and only on a stream window. Therefore, the α-accuracy has been selected as the major quality criterion. Details on this measure, which can cope with partly correct classifications, can be found in Section 4.3.1.

Classification Quality

To assess the α-accuracy of the Naive Bayes classifier and the custom-developed 1NN approach on the basis of class representatives, for both of them experiments have been run in the RapidMiner framework, embedded in each of the three presented stream classification models.

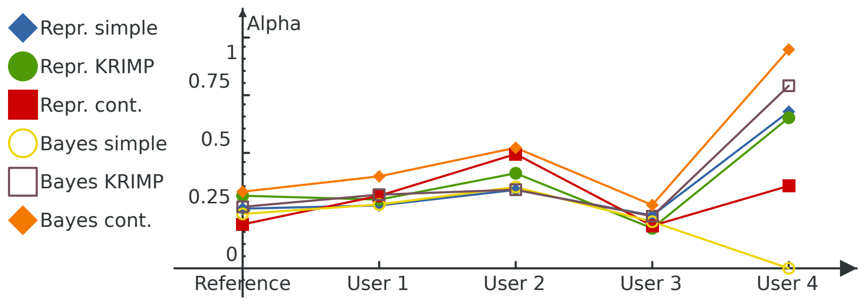


Figure 5.2.: Overview on tested stream classification methods. *Simple* denotes the simple stream processing approach, while *KRIMP* refers to the method involving StreamKrimp. The cluster representative 1NN classification approach is referred to a *FTC*.

Figure 5.2 indicates that all three stream classification approaches fail to yield an acceptable α-accuracy. None of the presented approaches was capable of producing accuracy significantly better that 0.5. Only for the data set User 4, which only contains 4 labels and very few examples, better values could be achieved. For those data sets with many labels, the classifiers even score only about 0.25. Although this is a disappointing result, this leads to the conclusion that none of the presented stream classification approaches is capable to solve the problem of tag prediction in micro-blogging services to a satisfying degree. The best result was generally achieved by the Naive Bayes classifier in combination with the continuous stream approach.

Bootstrap Effect

Section 5.1 noted, that the developed stream classification approaches suffer from the so-called bootstrap effect. Since no classification model is available on the first stream window, the classification accuracy (and α-accuracy) is always 0. Table 5.1 compares the measured α accuracy values with and without bootstrap effect for some of the experiments, i.e. for calculating the average α-accuracy of the *no boot* values, the first stream window has been left out.

		Reference	User 1	User 2	User 3	User 4	Avg
		Representative based classification					
KRIMP	full	0.26	0.43	0.31	0.16	0.43	0.32 ± 0.10
	no boot	0.29	0.44	0.32	0.17	0.43	0.33 ± 0.10
cont.	full	0.12	0.30	0.38	0.17	0.36	0.26 ± 0.10
	no boot	0.18	0.30	0.39	0.20	0.36	0.29 ± 0.08
		Naive Bayes					
KRIMP	full	0.22	0.19	0.30	0.21	0.36	0.25 ± 0.06
	no boot	0.26	0.19	0.33	0.23	0.36	0.28 ± 0.06
simple	full	0.21	0.53	0.31	0.19	0.07	0.26 ± 0.15
	no boot	0.22	0.53	0.35	0.22	0.07	0.28 ± 0.15

Table 5.1.: Stream classification bootstrap effect.

It can be noted, that the bootstrap effect is indeed visible, although it is not really strong. Overall, the StreamKrimp based stream classification approach yields the best results in combination with the class representative based approach developed in Section 4.2. Still, with an α-accuracy of 0.33 ± 0.10, this is not acceptable for a practical implementation of a tag recommendation service.

Krimp Based Stream Model

It is interesting to see if the Krimp based stream classification model can in fact detect changes on a stream and if this effect is visible within the classification quality of the embedded classifier. To assess this, the classification log of subsequent stream windows is presented for two user data sets in Figure 5.3 and Figure 5.4.

The α-accuracy is shown on subsequent stream windows for the custom average class representative 1NN classifier, in combination with the Krimp based stream classification model. An addition, a line represents the convergence state of the current Krimp code table. A low value indicates that the current code table did not converge, yet. A high point indicates a converged code table.

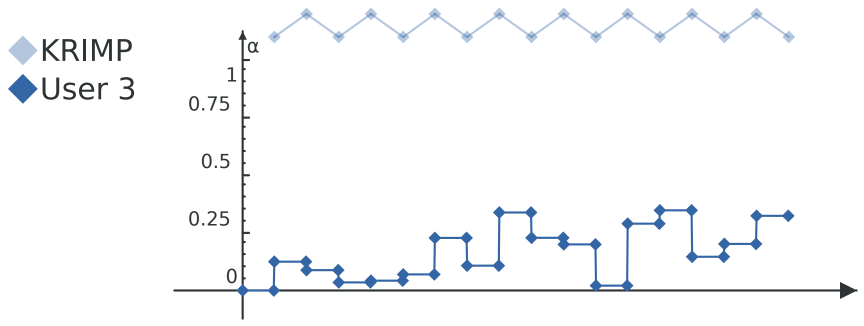


Figure 5.3.: Krimp stream classification log for User 3.

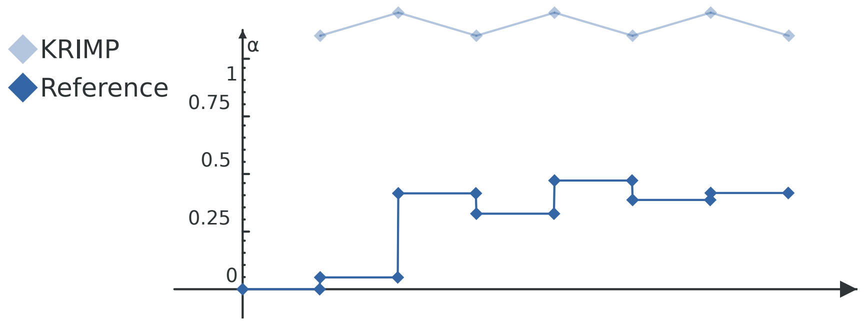


Figure 5.4.: Krimp stream classification log for Reference.

It can be noted, that the Krimp based change detection has no visual effect. Code tables generally take one stream window for converging and are abandoned again on the subsequent window, while the classification quality does not appear to be related to this frequency. Since the StreamKrimp parameters have been selected using a parameter optimization process, it can be assumed that the shown setting is optimal for the underlying case. Therefore, either the chosen stream model is not feasible to work with StreamKrimp, or the selected classifiers are not prone for changes in the probability distribution of tag sets.

Chapter 6

Conclusion

The following chapter concludes this thesis and gives an outlook on potential future research in the area of tag prediction in micro-blogging systems. The following Section 6.1 summarizes the results yielded throughout this thesis. After that Section 6.2 summarizes some detected problems and gives an impression on potential future research approaches. Finally, Section 6.3 completes this thesis with a bottom line.

6.1. Summary

The task of tag prediction in micro-blogging systems is a problem from the area of multi-label classification. The preceding thesis selected a data set transformation approach to tackle this multi-label classification problem using classifiers from the standard repertoire of machine learning. To enhance classification quality on basis of the selected transformation approach, a technique has been developed, which attempts to reduce the number of generated labels on the basis of frequent term-based clustering. In addition to that, a custom classification approach was presented, which relies on a class representative based condensation of the training data set, to which kNN is applied. Finally, it was attempted to embed the two most promising classification approaches into a data stream model, while three different stream classification approaches were evaluated.

In Chapter 3, the problem of tag prediction in micro-blogging systems has been identified as a multi-label classification problem. Several common approaches to tackle this problem using a transformation into a single-label problem, originally summarized in [TK07], have been presented and evaluated. The evaluation yielded, that a method, denoted as the *power set transformation method* in this thesis, appeared most feasible in the underlying case. This technique transforms the multi-label data set into an equivalent single-label variant by replacing each distinct label set with a unique, single label. Experiments with the selected standard classifiers, kNN, Naive Bayes and SVM, underlined the stated assumption, that this transformation technique would yield a large number of labels for actively tagging users, which influences classification accuracy negatively. In order to mitigate this problematic effect, a technique to reduce the number of labels produced by the power set transformation method has been developed.

The developed label-reduction approach is based on the technique of frequent term-based clustering, presented by Beil et al. in [BEX02]. This clustering technique attempts to cluster text documents on the basis of frequent term-sets, i.e. item sets, to ship around problems with the high dimensionality of the term vector model (see Section 3.1.1). In addition, frequent term-based clustering yields a description for each cluster, in form of a frequent term-set that is common to the documents in the cluster. The developed label reduction approach applied this clustering technique to the tag sets of the original multi-label data sets of micro-blogging entries. The tag sets contained in a certain cluster were then replaced by the cluster description yielded by FTC, i.e. a frequent term-set among the tag sets. Empirical evaluation showed, that FTC is capable of generating clusterings of reasonable quality. While Beil et al. suggested two different FTC algorithms, only the flat clustering variant has been evaluated for label reduction within this thesis. Furthermore, [BEX02] proposed a second overlap measure in order to generate sensible clusterings, which has not been considered in scope of this thesis.

The suitability of the developed label-reduction approach, for mitigating the classification accuracy problem involved with a large number of labels, has been shown empirically in Chapter 4. Experiments have been presented using the three standard classifiers, each on basis of data sets yielded by the pure power set transformation method compared to using the label-reduction technique. As expected, the classification accuracy could be increased significantly.

In addition to the standard classifiers, a custom classification approach has been developed and evaluated, which is inspired by [HK00]. This approach uses the well-known kNN method on a condensed training data set. Condensation is achieved by replacing all examples of a certain class by a smaller, representative set of vectors. Different methods for generating such a representative have been evaluated: The average vector, the median vector and the well-scattered points technique by Guha et al. ([GRS98]). Empirical evaluation showed that this classification approach, in combination with the average vector representative generation, is capable of outperforming even the best standard classifier, Naive Bayes, in many cases.

Finally, Chapter 5 presented a stream classification model, to develop an anytime algorithm for being used in a tag-recommendation web service, and three different model update strategies for it: Two simple approaches, where the first trains a new model on each stream window and the second updates a once trained model constantly on newly arriving data. In addition to that, a more complex method has been developed on basis of StreamKrimp ([vLS08]), which attempts to detect the probability distribution of tags in a micro-blogging stream and, more importantly, to detect changes in this distribution. The goal was to train models more effectively and to adjust them to changes in a users tagging behavior. Empirical evaluation rated none of the presented approaches as feasible for being used in practice, since none of them yielded an average α-accuracy above 0.33. While the most accurate approach was the StreamKrimp based technique, it could not be assessed that the changes detected in the probability distribution of tags relate to the classification quality on corresponding stream windows.

6.2. Future Research

Two different areas touched by this thesis provide special potential for future research: The developed label reduction approach and the stream prediction approach. Directions for future research are discussed in further detail in following.

6.2.1. Label Reduction Approach

The developed label reduction algorithm relies on frequent term-based clustering, presented by Beil et al. in [BEX02]. In the scope of this thesis, only the flat FTC version has been evaluated. Wurst and Kaspari describe an approach for multi-objective frequent term set clustering in [KW07]. They study clusterings based on frequent term sets, which are optimized in a multi-objective way, resulting in hierarchical clusterings that can e.g. used as navigation structures in tagging systems.

An interesting topic for future research would be to develop a way of utilizing these hierarchical clustering techniques in the scope of the label reduction technique. A crucial point with both approaches would be the selection of feasible cluster nodes from a hierarchical clustering, since such a clustering is commonly overlapping. In addition, the approach by Wurst and Kaspari does not yield a single clustering, but a set of Pareto optimal clusterings, from which a feasible one must be selected first.

6.2.2. Stream Classification

As presented in Section 5.3.3, the quality of the developed stream classification approaches, based on Naive Bayes and class representative based kNN classification, is not satisfactory. Most interestingly, the StreamKrimp based approach did not show any obvious relation between detected tag distribution changes, trained classification models on this basis and classification quality. One attempt to raise classification accuracy could be, to implement further enhancements to Krimp, presented by Siebes et al. in [SVvL06].

Section 5.1 noted that the presented stream classification approaches suffer all from the so-called bootstrap problem, since none of them has a classification model ready on the first window. One approach to this problem could be to extract potential tags from not tagged micro-blogging entries in an early stage of the processing. For example, frequent term sets of the full term vectors could be used. However, this would not fully solve the bootstrap issue, since for the first status update, no information from the user is available at all. Beside simple random selection of a word from this post, one could involve knowledge on the tagging behavior of related users. This methodology could also help enhancing the classification accuracy in the stream environment, if related users are chosen with similar tag behavior as the user in question.

6.3. Bottom Line

The developed batch classification approach for the purpose of tag-prediction in micro-blogging environments produces fully satisfying results. It has been shown, that the selected multi-label transformation technique is feasible in the underlying case and that classification accuracy can be significantly increased by applying the developed label-reduction approach on basis of FTC. The developed classifier, which works in a kNN manner on a condensed training set, turned out to even outperform the best evaluated standard classification approach, Naive Bayes. Both classification approaches are similarly modest in terms of computation time and memory utilization, which qualifies them for being used in the web service environment.

In contrast to this, the developed stream classification approaches did not yield the expected classification accuracy with neither of the analyzed classification approaches. This leaves the problem of tag-prediction on a stream basis still open for future research.

Appendix A

The Data Set

The data sets used for experiments throughout this thesis has been extracted from the Twitter web service. In following the process of data extraction is described in Section A.1. After that, some basic statistical analysis is performed in Section A.2.

A.1. Data Set Extraction

Twitter provides different ways for accessing its data, mainly micro-blogging entries and certain portions of user data. Beside the HTML based web interface for end-users, a web service *application programming interface* (API) is provided. This API is mainly meant for client applications, providers of services based on Twitter and research purposes. The example data used for experiments in this thesis has been extracted through this web service API, which is described in further detail in following.

A.1.1. Twitter Web Service API

The Twitter web service API follows the concept of *representational state transfer* (REST), similar to the architectural concept described by Fielding in [Fie00]. REST describes the architectural concept for distributed hypermedia, i.e. a web service architecture which utilizes the syntactical and semantical concepts of the *hyper text transfer protocol* (HTTP).

In the sense of REST, the Twitter web service API is mainly defined through resource identifiers in form of HTTP *uniform resource locators* (URLs). A web service consumed sends an HTTP request to such an URL in order to retrieve or manipulate information.

URLs are well-known from web browsing, an example for a Twitter web service URL is shown in Figure A.1. Above the URL, the syntax is explained: An URL consists of a protocol identifier (`http` in this case), a host specifier which can either be a network address or a host name – as in this case – and optionally a path specifier. Below the URL, the Twitter web service semantics is explained. The host name denotes the web service end point. The first path element indicates which version of the web service API the client wants to utilize for communication.

After that, the object type to be accessed is noted. In the presented case, this is a collection of status messages in form of a user timeline, i.e. micro-blogging entries by a specific user in reverse chronological order. The final path element specifies the user ID for which entries should be fetched and the desired response format, XML[1] in the presented case.

$$\overbrace{\texttt{http}}^{\text{protocol}}\texttt{://}\overbrace{\underbrace{\texttt{api.twitter.com}}_{\text{end point}}}^{\text{host}}\overbrace{\texttt{/}\underbrace{\texttt{1}}_{\text{ver.}}\texttt{/}\underbrace{\texttt{statuses/user\_timeline}}_{\text{resource type}}\texttt{/}\underbrace{\texttt{12345}}_{\text{ID}}\texttt{.}\underbrace{\texttt{xml}}_{\text{format}}}^{\text{path}}$$

Figure A.1.: Structure of a Twitter web service URL

As defined by the REST architecture, the shown URL does not identify a deterministic set of values, but refers to a concept, describing the data to expect. In the presented case, this is a set of status messages by the given user in reverse chronological order. The actually returned values will most likely vary, when being requested at two different points in time. Without any further options defined by a client, a HTTP `GET` request to the presented URL will return the 20 most recent micro-blogging entries by the desired user. Using options in form of HTTP `GET` parameters, the client can influence the returned data further. For example, by specifying the parameter `count`, the client specifies the number of statuses to be returned. It is important to note, that, conforming to the REST architecture, a request to the URL may also return an empty set. This, for example, happens if the given user did not produce any entries, yet. Furthermore, an error response might be returned, for example in case the parameters provided by the client where faulty or the service produced an error.

The Twitter web service provides access to a variety of other sets of resources and also allows the manipulation of such sets, given valid authentication and the authorization. This includes the creation of new entries, editing of user data and more. An extensive documentation of the provided methods can be found in the Twitter API documentation[2].

A.1.2. Data Extraction and Sampling

To assess the qualification of machine learning techniques developed within this this, example data has been extracted from Twitter using the web service API described in Section A.1.1. It is obviously not possible to run experiments on

[1]`http://www.w3.org/XML/` (2010-05-17)
[2]`http://apiwiki.twitter.com/Twitter-API-Documentation` (2010-05-17)

the complete Twitter data base (see Section 2.2.3), due to time, memory and processing power constraints. Furthermore, a user-centric approach is desired in this thesis. Therefore, a heuristic sampling mechanism has been applied during data extraction, in order to retrieve representative user data sets.

Population Definition

The desired data set has been extracted from the database of Twitter status messages, taking the related user into account (see Section 2.1). In order to retrieve the example data, users have been selected pseudo-randomly from Twitter and all possible micro-blogging entries written by them have been fetched. To ensure that the extracted data set provides micro-blogging entries written by real world users, only users fulfilling the following constraints have been taken into account:

1. The user must have written at least 2000 entries.
2. A minimum of 100 other users must follow the user in question.
3. The user must follow at least 50 other users.

Restriction 1 ensures, that the user provides enough example data to run experiments on. With criterion 2, it is attempted to exclude spam accounts from the population. Such users follow as much other people as possible, to gain their attention for advertisement purposes. A human can detect such behavior easily and would not follow the spamming user. Therefore, a reasonable amount of followers is a good indicator against spam accounts. Finally, a user must follow at least 50 other users in order to qualify for the population (condition 3). This requirement is meant to exclude bot accounts. The status messages of such accounts are generated through a computer program. Depending on their service, such bots can have quite some followers, but do usually not follow many users. An example for such a bot is GetForecast[3].

From all users fulfilling the defined constraints, it has been tried to extract a representative sample.

Sampling

Since it is infeasible to extract the whole population from the Twitter database, a heuristic has been defined for extracting a sufficiently random sample.

To select random users, the overall public timeline of Twitter has been used, which presents the most recent micro-blogging entries over all users at a given point of time. Due to the huge number of users on Twitter, it is expected that fetching this collection of statuses at random point in time inheres enough entropy to be reasonably random.

[3]`http://twitter.com/getforecast` (2010-05-17)

On this background, the algorithm shown in Figure 13 has been developed to sample data of 50 distinct users from the Twitter database. Each user found in the public timeline has been checked for the presented constraints. If the user appeared feasible, his / her user data has been stored in a local MySQL[4] database in order to extract micro-blogging entries for this users afterwards.

Algorithm 13 Algorithm to sample users from Twitter

```
function SampleData
    $sample := \emptyset$
    while $|sample| < 50$ do
        $latest := fetchLatestStatuses()$
        $users := extractUsers() \setminus sample$
        for $user \in users$ do
            if conditionsFulfilled($user$) then
                $sample := sample \cup \{user\}$
            end if
        end for
    end while
    return $sample$
end function
```

The generated population consists of more than 140,000 micro-blogging entries. A simple SQL query for micro-blogging entries that contain the # char – which indicates a tag – revealed, that approximately 10,000 micro-blogging entries contain tags.

From the 50 fetched user, four have been randomly selected to deal as example data for experiments. These are denoted with *User x* ($x \in \{1, \ldots, 4\}$) through experiment evaluation. In addition, the Twitter data of the thesis author has been chosen as a reference data set, because of the a priori knowledge about the active tagging.

A.2. Data Set Statistics

To gain an insight into the sampled data set, the collected data has been analyzed from a statistical point of view. Leading questions are:

- How do users tag their statuses?
- How are terms and tags distributed within user sample?

[4] `http://mysql.com/` (2010-07-22)

- Of what size are the dictionaries of the users?
- How sparse are word vectors created from micro-blogging entries?
- Does the tagging behavior of users change over time?

The presented statistics have almost exclusively been calculated in the Rapid-Miner framework. However, some statistics were extracted directly from the database using SQL statements.

A.2.1. Tagging Behavior

The fraction of tagged statuses (see Section A.1) indicates, that tagging is overall not very common among Twitter users. To verify this impression, the fraction of labeled examples was calculated on a per-user basis over the selected sample.

	Reference	User 1	User 2	User 3	User 4	Avg.
# entries	1047	3155	2654	3195	3199	2549.8 ± 931.84
# tagged	805	466	258	1236	20	557 ± 476.60
Fraction	0.77	0.15	0.1	0.39	0.006	0.28 ± 0.31

Table A.1.: Tagging behavior statistics. Number of overall entries (# *entries*) and tagged statuses.

The results in Table A.1 do not confirm the impression that in general only a fraction of 7% of the statuses are tagged. Instead, users show a brought diversity in tagging behavior. Some exist with a much higher as well as such with a much lower rate. While User 3 tagged 39% of his / her micro-blogging entries, User 4 labeled only 0.6%. As expected, the Reference sample turns to tag very actively, with a fraction of 0.77 tagged statuses. The average and standard deviation values underline the impression that users tagging behavior differs widely.

For the desired tag prediction system, the data of users who participate more actively in tagging were expected to yield classification results. Such users offer a larger amount of training data, which, on the one hand, promises better classification quality than few training data. On the other hand, this could also lead to a tight separation boundary between classes and therefore to worse classification quality. Experiments presented in Section 4.3.3 confirmed both ideas: User samples with many training examples do yield better classification performance, but a large number of tags mitigates this effect.

A.2.2. Dictionary and Term Vector Size

To get an idea of the distribution of words used in a users timeline, term vectors have been generated from each user samples, as defined in Section 3.1.1. The overall occurrence of each term has been measured and the Euclidean length of the term vectors has been analyzed as an indicator for their sparseness.

The number of occurrences of a term in an example (status) has been used as the term vector values. Table A.2 presents and compares the dictionaries sizes on a per-user basis in the first data row.

The sample User 4 has a very brought dictionary, while the Reference user provides smallest number of words. This could be the result of users writing in differently sized domains. The reference user writes statuses mostly about open source development, specifically the PHP programming language. Manual inspection of the examples in User 4 yielded that the domain if this user are job offers in various fields, what confirms the thesis.

The average number of words in a users dictionary is low, compared to common numbers for such cases, i.e. 10,000 noted by Beil, et al. in [BEX02].

	Reference	User 1	User 2	User 3	User 4	Avg.
# terms	2449	5730	3389	4332	6128	4415.60 ± 1532.50
Max. length	8.145	2.646	13.379	23.0	6.325	10.70 ± 7.89

Table A.2.: Dictionary size and term vector length statistics

In addition, the maximum Euclidean length of the generated term vectors for each user is presented in Table A.2. The typical assumption that term vectors are sparse holds, even for User 3: Given an Euclidean length of 23, a maximum of 529 unique words could have occurred in the underlying status. Compared to 4332 words in the dictionary of this user, this can be considered sparse. Manual inspection of the affected example has turned out, that it actually consisted of 23 times the same word. The next lower vector length for this user is 16.03, which is still high compared to the other users. But again, this status consisted of 15 times the same word and 1 different one. The first entry with roughly sensible content has a length of 8.94, which fits into the average and confirms the assumption.

Considering the limited size of 140 characters per status, 4900 is an upper bound for the word vector length. Such an example would consist of 70 different characters divided by spaces. Given the dictionary sizes, it can be concluded that micro-blogging entries must therefore always lead to sparse term vectors.

A.2.3. Term Distribution

Figure A.2 shows the term distribution for User 3. Terms are sorted by their occurrence frequency on the domain axis. The absolute overall term occurrence is noted on the value axis. The presented distribution is prototypical for all samples and optically appears to be following Zipf's law, as one would naturally expect for text content.

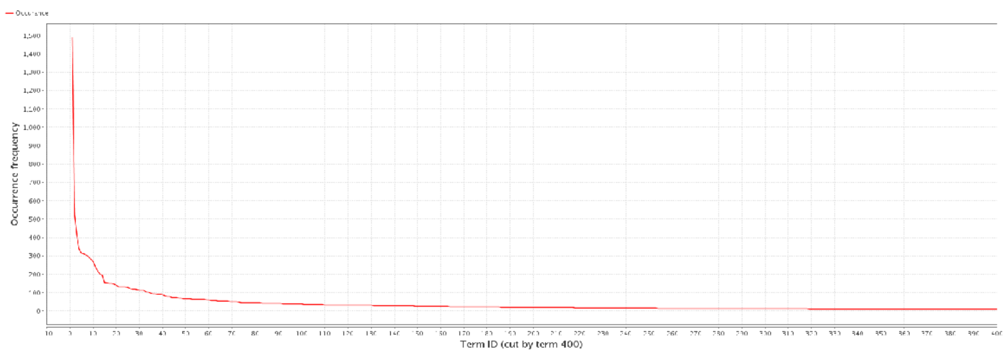


Figure A.2.: Term distribution statistics

A manual inspection of the terms most frequently used by the analyzed user samples revealed, that Twitter has a small custom slang. For example, the most frequent term for User 3 is `rt`, which is the abbreviation for *Re-Tweet*, i.e. the quotation of another users status. This term is also frequent for 2 other user samples.

Parts of URLs, such as `http` and `com`, occur frequently among four of the five analyzed samples. Due to the size limitation of statuses, it is common practice in micro-blogging services to cut down the character length of URLs as far as possible. This typically works by using *URL shortening services*. Such services create a tiny but cryptic URL for which the web server just responds with the HTTP redirect response `301 Moved Permanently`, including the desired destination. An example is `http://tinyurl.com/ykgf3kv` which redirects to `http://schlitt.info/opensource/blog/0716_convert_from_to_opendocument.html`. Using the short URL in this case saves 48 characters. The shown TinyURL[5] service is frequently used by two analyzed users, indicated by the term occurance frequency of the term `tinyurl`.

It is interesting to note, that typical internet slang acronyms like `lol` do not occur as frequently as one would intuitively expect. For User 4 the term resides on frequency rank 9 with 283 occurrences, but none of the other users samples has this term among top 10. Other internet slang terms are not present in the top 10 of frequent terms for the samples.

[5]`http://tinyurl.com` (2010-05-17)

A.2.4. Tag Distribution

To gain an insight into the distribution of tags in a user sample, term vectors have been generated from the tags only. Table A.3 presents the following data on a per user basis:

#	refers to the number of unique tags
Max occ.	denotes the maximum occurrence of a single tag
Max length	specifies the maximum length of a tag term vector
# occ. 1	refers to the number of tags that occur only once.

	Reference	User 1	User 2	User 3	User 4	Avg.
#	357	76	56	489	4	196.40 ± 213.68
Max occ.	93	110	97	144	8	88.75 ± 57.86
Max length	2.24	2.65	8.00	23.00	1.00	7.38 ± 9.14
# occ. 1	208	37	29	321	2	119.40 ± 138.96

Table A.3.: Tag occurrence statistics

The number of unique tags fulfills the intuitive impression based on the users tagging behavior: Users who participate more actively in tagging use more tags than those who do not tag frequently. Of the used tags, a fraction of about 50% occurs only once in the sample. This confirms the motivation for the tag-prediction system developed in this thesis. The distribution of tags in the sample of User 1 is visualized in Figure A.3. Again, this visually appears to be a distribution roughly following Zipf's law. The tag distributions for other users look similar.

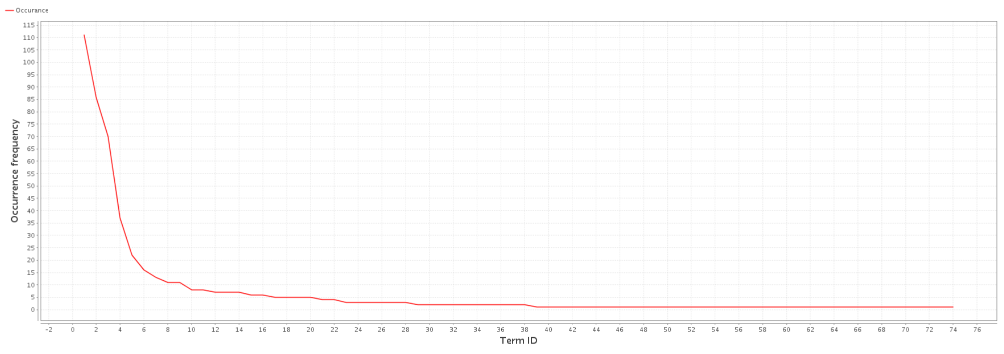


Figure A.3.: Tag distribution statistic

Appendix B

Experiments

In following, selected experiment setups and results for the experiments performed throughout this thesis are presented.

Experiment setups are shown as developed in the RapidMiner framework. RapidMiner uses a tree structure to combine so-called operators. An operator can encapsulate arbitrary actions on its given input data, like manipulation, creation of new data or calling subordinate operators with potentially manipulated input data. Operators are executed by an *in order* traversal of the operator tree.

Data in RapidMiner is handled in form *input output* (IO) objects, which can contain arbitrary data structures, depending on their type. Examples for IO objects are example sets — of different types — or classification models. These IO objects are maintained in a container, which is submitted to each operator, subsequently. Operators may consume IO objects from the container or add new ones. This way, operators can retrieve IO objects generated by their predecessors, if it has not been consumed by an intermediate operator.

B.1. Custom Operators

Whenever possible, the standard operators shipped with the RapidMiner framework have been used to realize experiments throughout this thesis. However, in some cases no suitable operators were available, so that custom ones have been implemented. These are

- an operator to realize *frequent term-based clustering* as described in [BEX02] (see Section 3.3.3)
- an operator that realizes the hierarchical version of the FTC algorithm, although not used within this thesis
- a custom Naive Bayes operator which is can be updated with new, unseen classes during stream learning
- an operator for calculating inter and intra cluster distances (see Section 3.3.4)
- several operators around the Krimp and StreamKrimp algorithms, used for stream classification experiments (see Section 5.2)

- an operator to calculate the Rand index of two clusterings (see Section 3.3.4)
- an operator realizing the custom classification technique on basis of class representatives (Section 4.2).

These operators are currently only available through an internal subversion repository of the *Artificial Intelligence Group* of TU Dortmund, but might be made available as open source software after this thesis has been published.

B.2. Basic Experiment Setup

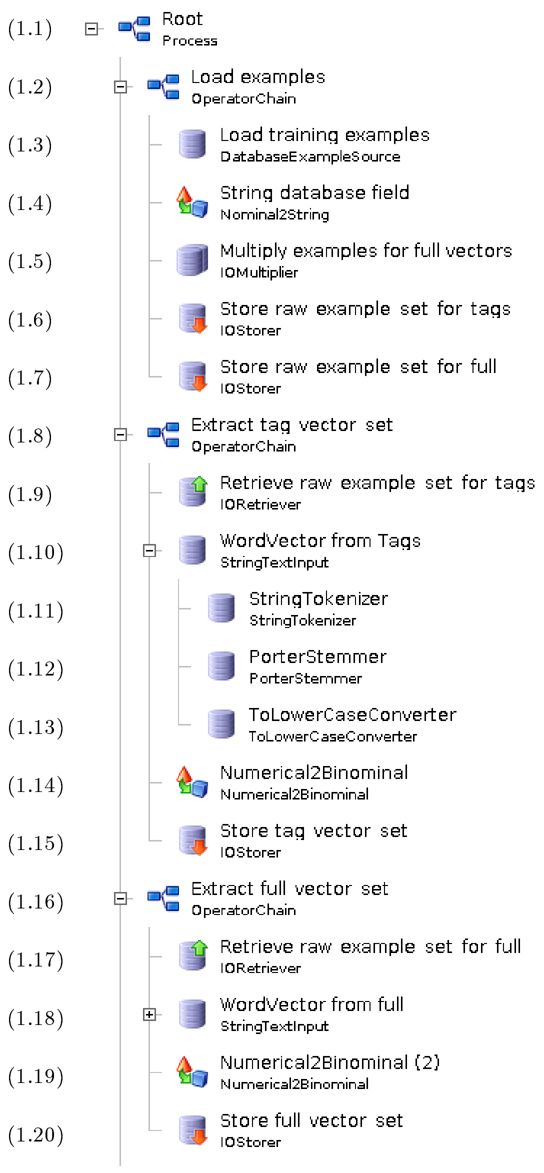

(1.1) Root
Process
(1.2) Load examples
OperatorChain
(1.3) Load training examples
DatabaseExampleSource
(1.4) String database field
Nominal2String
(1.5) Multiply examples for full vectors
IOMultiplier
(1.6) Store raw example set for tags
IOStorer
(1.7) Store raw example set for full
IOStorer
(1.8) Extract tag vector set
OperatorChain
(1.9) Retrieve raw example set for tags
IORetriever
(1.10) WordVector from Tags
StringTextInput
(1.11) StringTokenizer
StringTokenizer
(1.12) PorterStemmer
PorterStemmer
(1.13) ToLowerCaseConverter
ToLowerCaseConverter
(1.14) Numerical2Binominal
Numerical2Binominal
(1.15) Store tag vector set
IOStorer
(1.16) Extract full vector set
OperatorChain
(1.17) Retrieve raw example set for full
IORetriever
(1.18) WordVector from full
StringTextInput
(1.19) Numerical2Binominal (2)
Numerical2Binominal
(1.20) Store full vector set
IOStorer

Figure B.1.: Basic experiment setup.

Figure B.1 shows the RapidMiner operator tree used as the basis in all subsequent experiments. It loads a desired user data sample from the database and performs basic pre-processing operations, in order to produce feasible data structures for the desired experiments: One example set consists of the term vectors of micro-blogging entries and a second example set consists of a tag vector for every example. This representation has been chosen, since RapidMiner does not natively support the multi-label environment.

The presented operator tree consists of three basic steps, composed in form of operator sub-trees:

1. Loading of the micro-blogging entries for a specific user (1.2).
2. Extraction of tag vectors from micro-blogging entries in (1.8).
3. Extraction of full term vectors in (1.16).

Loading of the micro-blogging entries involves the actual load operation from a relational database (1.3) and conversion of the nominal field containing the micro-blogging entries into a string text field (1.4). After that, two copies (1.5) of this example set are put into dedicated memory storages (1.6, 1.7) for later retrieval in each of the following operator trees.

Pre-processing for the tag example set (1.8) and full term vectors (1.16) is pretty similar. First, the corresponding version of the micro-blogging entry example set are fetched from the memory storage (1.9 and 1.17). After that, an operator consumes this example set and generates a new version, consisting of the desired term vectors (1.10 and 1.18). For simplicity reasons, this sub-tree is collapsed in 1.18, since it basically duplicates 1.10. The term vector generation involves string tokenizing (1.11), splitting the text into tokens at each non-alphanumeric character for the full term vectors and extracting every sequence of alphanumeric characters that is preceded by a # character for the tag vectors. After that, the resulting terms are stemmed and converted to lower-case, as described in Section 3.1.2.

On the resulting term vector example set, each numerical field is the converted to a binominal fields, in order to weight term vectors by binary occurrence (1.14 and 1.19). Finally, the example sets are stored in memory again for later usage in experiments in the operators 1.15 and 1.20.

Storing of IO objects in a dedicated memory storage removes them from the global IO object container. This cleans up the overall process data and avoids conflicts of that form, that an operator accidentally chooses the wrong IO item, if multiple of the same type are present. For example if two example sets are present.

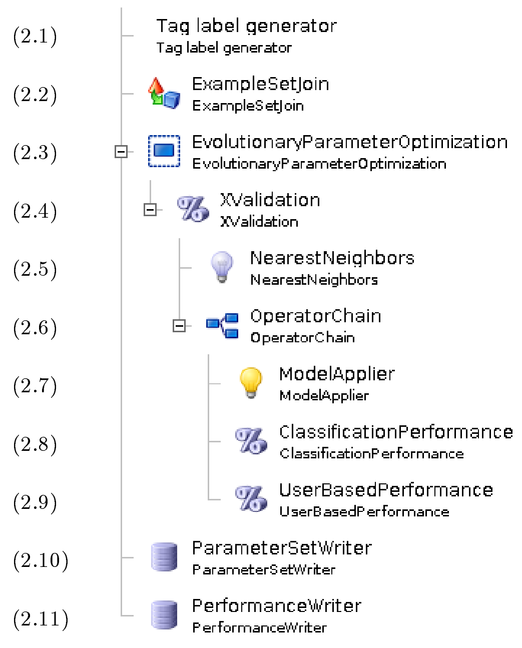
This basic pre-processing is performed for every subsequently presented experiment and therefore left out in the following.

B.3. Multi-Label Experiments

The first experiment series in this thesis consisted of the evaluation of the three selected standard classifiers – kNN, Naive Bayes and SVM (see Section 4.1) – and the custom, class representative based, classification approach (Section 4.2) on basis of the power-set transformation method. This approach to tackle the multi-label classification problem relies on an example set transformation, where each distinct label set is replaced by a new, unique, single label. For example, the label set $\{a, b, c\}$ is replaced by $a \wedge b \wedge c$.

The second experiment series is rather similar, but involves the label reduction approach, developed in Section 3.3.3. This approach performs frequent term-based clustering on the multi-label sets and replaces a number of them with a common, frequent subset. This reduces the overall number of labels generated by the power set method.

B.3.1. Experiment Setup



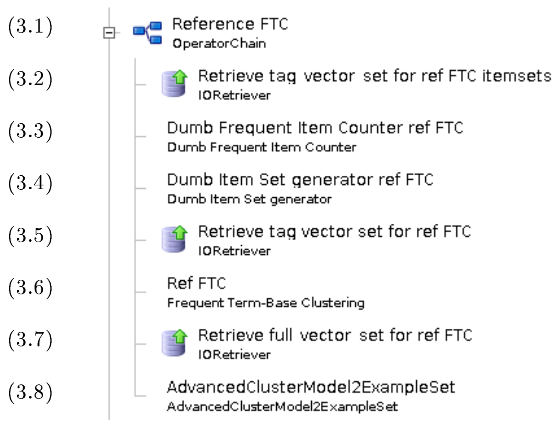
```
(2.1)   Tag label generator
          Tag label generator
(2.2)   ExampleSetJoin
          ExampleSetJoin
(2.3)   EvolutionaryParameterOptimization
          EvolutionaryParameterOptimization
(2.4)     XValidation
            XValidation
(2.5)       NearestNeighbors
              NearestNeighbors
(2.6)       OperatorChain
              OperatorChain
(2.7)         ModelApplier
                ModelApplier
(2.8)         ClassificationPerformance
                ClassificationPerformance
(2.9)         UserBasedPerformance
                UserBasedPerformance
(2.10)  ParameterSetWriter
          ParameterSetWriter
(2.11)  PerformanceWriter
          PerformanceWriter
```

Figure B.2.: Power set classification experiment setup.

The setup of the kNN evaluation experiment is prototypically shown in Figure B.2. Before the shown operators, the basic experiment setup shown in Figure B.1 is placed. The first experiment operator (2.1) generates an example set with a single label attribute, containing the power set labels. The example set join operator (2.2) merges the full term vector example set with the power-set labels.

The next operator sub-tree (2.3) performs evolutionary parameter optimization as described in Section 4.3.2. For each generated individual, i.e. parameter value set, the trailing operator tree is executed to evaluate the fitness. In the shown case, the parameter k for the kNN classifier is optimized. During evaluation of the fitness of a parameter set, stratified cross validation (Section 4.3.1) is used to asses the generalization performance of the operator (2.4). The cross validation process trains a kNN classification model (SVM and Naive Bayes analogous) on the training fraction of the data set in a first step (2.5). The second step consists of applying the trained model (2.7) and the quality evaluation using classification accuracy (2.8) and α-evaluation (2.9). Information about these quality measures can be found in Section 4.3.1.

After the evolutionary parameter optimization process is finished, the found optimal parameter value set is written to disc (2.10), as well as the yielded performance (2.11).



(3.1) Reference FTC
OperatorChain

(3.2) Retrieve tag vector set for ref FTC itemsets
IORetriever

(3.3) Dumb Frequent Item Counter ref FTC
Dumb Frequent Item Counter

(3.4) Dumb Item Set generator ref FTC
Dumb Item Set generator

(3.5) Retrieve tag vector set for ref FTC
IORetriever

(3.6) Ref FTC
Frequent Term-Base Clustering

(3.7) Retrieve full vector set for ref FTC
IORetriever

(3.8) AdvancedClusterModel2ExampleSet
AdvancedClusterModel2ExampleSet

Figure B.3.: Label reduced classification experiment setup.

In order to run the same experiment, but with label-reduction involved, only the label-generation operators (2.1 and 2.2) are removed and replaced by the operator sub-tree shown in Figure B.3. The operators 3.3 and 3.4 mine all item sets from the tag sets. After that, frequent term-based clustering is applied to these (3.6). Finally, operator 3.8 maps the reduced labels to the full term vectors.

B.3.2. Experiment Results

Figure B.4 shows the classification accuracy yielded by the different classification approaches in the experiment presented in the last Section B.3.1. Table B.1 compares results for the power set label experiments and label reduced experiments, in terms of accuracy and α-accuracy.

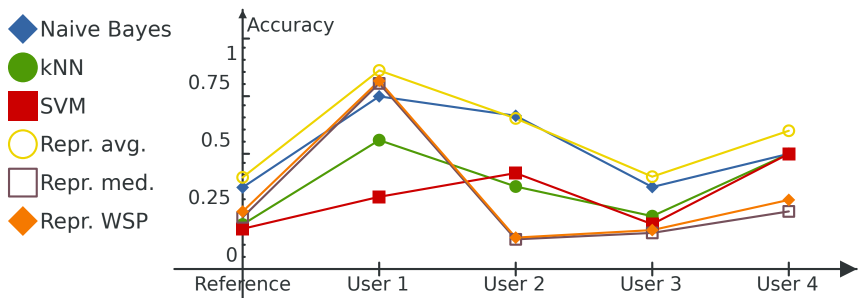


Figure B.4.: Classification accuracy on power-set labels

			Reference	User 1	User 2	User 3	User 4	Avg.
Power set labels	Naive Bayes	(acc.)	0.35	0.75	0.66	0.36	0.50	0.52 ± 0.16
		(α-acc.)	0.38	0.75	0.68	0.38	0.55	0.55 ± 0.15
	kNN	(acc.)	0.19	0.56	0.36	0.23	0.50	0.37 ± 0.14
		(α-acc.)	0.22	0.56	0.36	0.24	0.55	0.39 ± 0.15
	SVM	(acc.)	0.17	0.31	0.42	0.19	0.50	0.32 ± 0.13
		(α-acc.)	0.18	0.31	0.42	0.20	0.55	0.33 ± 0.14
	Repr. avg.	(acc.)	0.40	0.86	0.65	0.40	0.60	0.58 ± 0.17
		(α-acc.)	0.46	0.87	0.71	0.43	0.65	0.62 ± 0.16
	Repr. med.	(acc.)	0.22	0.81	0.13	0.16	0.25	0.31 ± 0.25
		(α-acc.)	0.29	0.81	0.18	0.21	0.28	0.35 ± 0.23
	Repr. WSP	(acc.)	0.25	0.82	0.14	0.17	0.30	0.33 ± 0.25
		(α-acc.)	0.30	0.82	0.19	0.21	0.30	0.36 ± 0.23
Reduced labels	Naive Bayes	(acc.)	0.61	0.83	0.79	0.55	0.55	0.67 ± 0.12
		(α-acc.)	0.61	0.83	0.79	0.55	0.55	0.67 ± 0.12
	kNN	(acc.)	0.34	0.55	0.38	0.23	0.60	0.42 ± 0.14
		(α-acc.)	0.34	0.55	0.38	0.23	0.60	0.42 ± 0.14
	SVM	(acc.)	0.23	0.31	0.44	0.12	0.60	0.34 ± 0.17
		(α-acc.)	0.23	0.31	0.44	0.12	0.60	0.34 ± 0.17
	Repr. avg.	(acc.)	0.61	0.87	0.76	0.58	0.75	0.71 ± 0.11
		(α-acc.)	0.61	0.87	0.76	0.58	0.75	0.71 ± 0.11
	Repr. med.	(acc.)	0.31	0.82	0.56	0.26	0.55	0.50 ± 0.20
		(α-acc.)	0.31	0.82	0.56	0.26	0.55	0.50 ± 0.20
	Repr. WSP	(acc.)	0.66	0.88	0.78	0.59	0.87	0.76 ± 0.11
		(α-acc.)	0.66	0.88	0.78	0.59	0.87	0.76 ± 0.11

Table B.1.: Accuracy and α-accuracy on power set and reduced labels

B.4. Stream Classification Experiments

In order to assess stream classification capabilities, a similar experiment setup as the one shown in the previous Section B.3 has been used. The following sections present an exemplary setup and the results yielded from stream classification experiments.

B.4.1. Experiment Setup

The setup of the stream experiment series is similar to the experiment presented in Section B.3.1. The data preparation phase is the same as shown in Figure B.1. Before the operator chain shown in Figure B.5, a reference FTC run is preformed in Figure B.3, in order to create the reduced power set transformed labels as a reference.

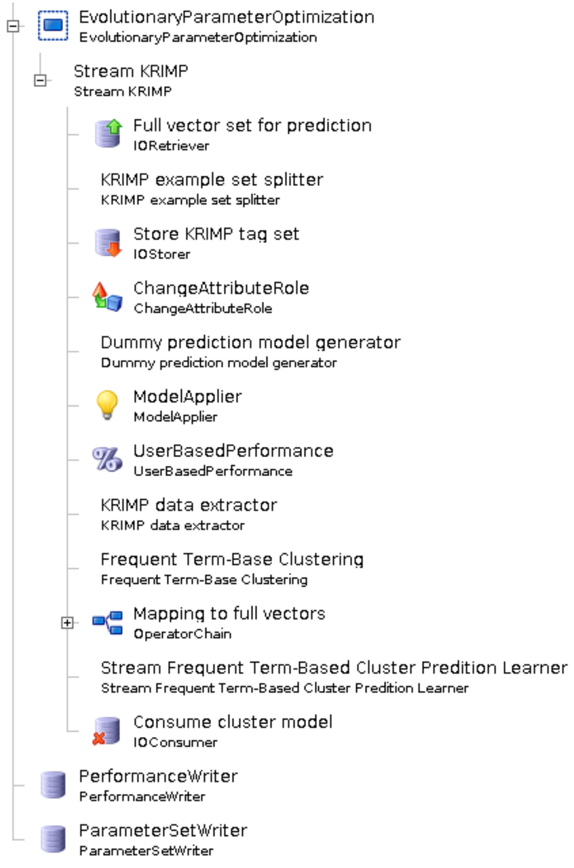
The presented operator chain appears a bit complicated, due to the fact that three different label attributes are needed:

1. The ground truth label (reduced and power set transformed)
2. the predicted label
3. the label to train on a given stream window.

It has to be noted, that the first and last label attributes are not the same. The first one consists of applying the power set transformation method to the full example set, so as the label reduction technique. These ground-truth labels must not be trained by the stream classifiers. For this purpose, the last label attribute us used: It contains the labels extracted from the current stream window, using the power set transformation and label reduction technique.

Since RapidMiner does not allow three label attributes for an example set, a trick is used here: In the stream classification process, the first and last noted label attributes each become a cluster attribute alternately. The corresponding other one then becomes the label attribute. In the initial state, the ground truth labels are stored in a cluster attribute.

In case of stream classification, the evolutionary parameter optimization (5.1) does not only need to optimize the classifier parameters, but also those of the StreamKrimp algorithm (e.g. *blockSize*). Detailed information on these parameters can be found in Section 4.3.2 and Section 5.3.3. There is a single operator sub-tree below the optimization process, lead by the StreamKrimp operator (5.2). From the standard IO object container, this operator receives the tag vector example set. The operator is responsible for two things: Firstly, it splits the tag vector set into stream windows and submits only one chunk per classification round to the subsequent operators. Secondly, it performs the actual Krimp operations, i.e. detection of the tag distribution and whenever this distribution changes. See Section 5.2 for details.



(5.1) EvolutionaryParameterOptimization
EvolutionaryParameterOptimization

(5.2) Stream KRIMP
Stream KRIMP

(5.3) Full vector set for prediction
IORetriever

(5.4) KRIMP example set splitter
KRIMP example set splitter

(5.5) Store KRIMP tag set
IOStorer

(5.6) ChangeAttributeRole
ChangeAttributeRole

(5.7) Dummy prediction model generator
Dummy prediction model generator

(5.8) ModelApplier
ModelApplier

(5.9) UserBasedPerformance
UserBasedPerformance

(5.10) KRIMP data extractor
KRIMP data extractor

(5.11) Frequent Term-Base Clustering
Frequent Term-Base Clustering

(5.12) Mapping to full vectors
OperatorChain

(5.13) Stream Frequent Term-Based Cluster Predition Learner
Stream Frequent Term-Based Cluster Predition Learner

(5.14) Consume cluster model
IOConsumer

(5.15) PerformanceWriter
PerformanceWriter

(5.16) ParameterSetWriter
ParameterSetWriter

Figure B.5.: Stream classification experiment setup.

Inside the StreamKrimp operator, the full term vector set is retrieved in order to perform classification (5.3). The next operator splits the full vector set according the given stream window, so classification can take place on it. The current stream window on the tag vector example set is stored for later usage in 5.5. After that, the label attribute trick is applied, changing the role of the attribute containing the ground truth labels to be a label attribute (5.6).

Operator 5.7 creates a dummy prediction model, which predicts *unknown* for every label, in case no valid prediction model is found in the IO object container. This is only the case on the first window, where the bootstrap effect (see Section 5.1) occurs. Starting with the second stream window, a valid prediction model is present, trained later on in the stream process. The operators 5.8 and 5.9 perform the actual classification and evaluation, compared to the ground truth labels.

The next operator (5.10) extracts the item sets mined within the StreamKrimp operator 5.2, followed by frequent term-based clustering (5.11) for label reduction. The operator sub-tree 5.12 contains the label mapping, which has already been shown in Figure B.3. Inside this chain, the label attribute trick is also applied in reverse direction, so that the ground truth labels are not overwritten during the label mapping. Finally, the classification model for the current stream window is generated on these labels (5.13). In the shown setup, this is the custom class representative based classifier for which details can be found in Section 4.2. Finally, the FTC cluster model is removed from the IO object container (5.14).

After the parameter optimization process finished, the optimal parameter values and the performance they yielded are stored to disc.

B.4.2. Experiment Results

Figure B.6 visualizes the α-accuracy yielded by each of the evaluated stream approaches, using the two classifiers which performed best on the original data set (Naive Bayes and the custom approach developed within this thesis). Table B.2 presents the visualized data in numbers (α-accuracy).

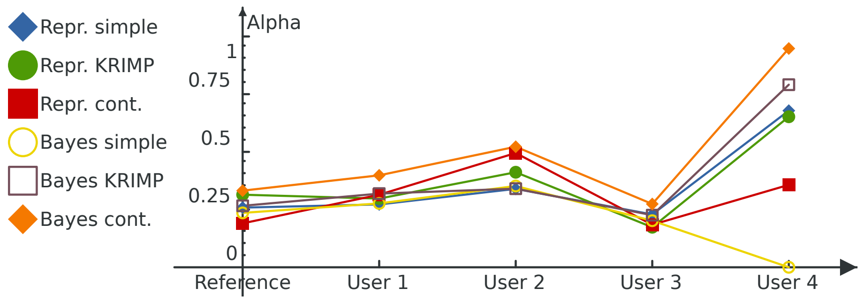


Figure B.6.: Stream classification performance

	α-accuracy	Reference	User 1	User 2	User 3	User 4	Avg
	simple	0.26	0.27	0.34	0.23	0.68	0.36 ± 0.17
Repr.	KRIMP	0.31	0.30	0.41	0.17	0.65	0.37 ± 0.16
	cont.	0.19	0.31	0.50	0.18	0.36	0.31 ± 0.12
	simple	0.24	0.28	0.35	0.20	0	0.21 ± 0.12
Bayes	KRIMP	0.27	0.32	0.34	0.23	0.79	0.39 ± 0.21
	cont.	0.33	0.40	0.52	0.27	0.95	0.49 ± 0.24

Table B.2.: Stream classification performance

Bibliography

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases (VLDB)*, pages 487–499. MorganKaufmann, 1994.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison Wesley, 1986.

[Azu02] Francisco Azuaje. A cluster validity framework for genome expression data. *Bioinformatics*, 18(2):319–320, 2002.

[BEX02] Florian Beil, Martin Ester, and Xiaowei Xu. Frequent term-based text clustering. In *KDD*, pages 436–442. ACM, 2002.

[BJC$^+$04] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, and Ophir Frieder. Hourly analysis of a very large topically categorized web query log. In *27th ACM SIGIR Conf.*, pages 321–328, July 2004.

[BLSB04] M. R. Boutell, J. B. Luo, X. P. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, September 2004.

[BS93] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[DC00] Susan T. Dumais and Hao Chen. Hierarchical classification of web content. In *SIGIR*, pages 256–263, 2000.

[DTMV05] Sotiris Diplaris, Grigorios Tsoumakas, Pericles A. Mitkas, and Ioannis P. Vlahavas. Protein classification with multiple algorithms. In Panayiotis Bozanis and Elias N. Houstis, editors, *Panhellenic Conference on Informatics*, volume 3746 of *Lecture Notes in Computer Science*, pages 448–456. Springer, 2005.

[ELM03] Susana Eyheramendy, David D. Lewis, and David Madigan. On the naive bayes model for text categorization, November 01 2003.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[FH02] V. Franc and V. Hlavac. Multi-class support vector machine. In *ICPR*, pages II: 236–239, 2002.

[Fie00] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of Califormia, Irvine, USA, 2000.

[GÖ03] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 32(2):5–14, June 2003.

[GQ03] Teresa Gonçalves and Paulo Quaresma. A preliminary approach to the multilabel classification problem of portuguese juridical documents. In Fernando Moura-Pires and Salvador Abreu, editors, *EPIA*, volume 2902 of *Lecture Notes in Computer Science*, pages 435–444. Springer, 2003.

[GRS98] Guha, Rastogi, and Shim. CURE: An efficient clustering algorithm for large databases. *SIGMODREC: ACM SIGMOD Record*, 27, 1998.

[Grü05] Peter Grünwald. *A Tutorial Introduction to the Minimum Description Length Principle.* MIT Press, March 2005.

[GS04] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 22–30. Springer, 2004.

[GZ03] Bart Goethals and Mohammed Javeed Zaki, editors. *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA*, volume 90 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2003.

[HK00] Eui-Hong (Sam) Han and George Karypis. Centroid-based document classification: Analysis and experimental results. *Lecture Notes in Computer Science*, 1910:424–??, 2000.

[HMS02] Andreas Hotho, Alexander Maedche, and Steffen Staab. Ontology-based text document clustering. *KI*, 16(4):48–54, 2002.

[HPYM04] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov*, 8(1):53–87, 2004.

[HRGM08] Paul Heymann, Daniel Ramage, and Hector Garcia-Molina. Social tag prediction. In Sung-Hyon Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat-Seng Chua, and Mun-Kew Leong, editors, *SIGIR*, pages 531–538. ACM, 2008.

[HTF01] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning.* Springer, July 2001.

[JFY09] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, 2009.

[JGZ04] Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2004.

[JMH+07] Robert Jäschke, Leandro Balby Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *LWA*, pages 13–20, 2007.

[Joa97a] T. Joachims. A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization. In *Proceedings of International Conference on Machine Learning*, 1997.

[Joa97b] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report LS VIII-Report, Universität Dortmund, Dortmund, Germany, 1997.

[Joa99] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[KL03] S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation*, 15(7):1667–1689, 2003.

[KTV08] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multi-label text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD 2008 Discovery Challenge*, 2008.

[KW98] M. Kirsten and S. Wrobel. Relational distance-based clustering. *Lecture Notes in Computer Science*, 1446:261–??, 1998.

[KW07] Andreas Kaspari and Michael Wurst. Multi-objective frequent termset clustering. In Alexander Hinneburg, editor, *LWA*, pages 133–140. Martin-Luther-University Halle-Wittenberg, 2007.

[LA99] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *KDD*, pages 16–22, 1999.

[LH03] Boris Lauser and Andreas Hotho. Automatic multi-label subject indexing in a multilingual environment. In Traugott Koch and Ingeborg Sølvberg, editors, *ECDL*, volume 2769 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2003.

[LO03] Tao Li and Mitsunori Ogihara. Detecting emotion in music. In *ISMIR*, 2003.

[McC99] Andrew McCallum. Multi-label text cassification by EM. *AAAI'99 Workshop on Text Learning.*, 1999.

[Mie06] Ingo Mierswa. Evolutionary learning with kernels: a generic solution for large margin problems. In Mike Cattolico, editor, *GECCO*, pages 1553–1560. ACM, 2006.

[MM04] Paul McNamee and James Mayfield. Character N-gram tokenization for european language text retrieval. *Inf. Retr*, 7(1-2):73–97, 2004.

[MM05] Ingo Mierswa and Katharina Morik. Automatic feature extraction for classifying audio data. *Machine Learning*, 58(2-3):127–149, 2005.

[OFG97] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *CVPR*, pages 130–136, 1997.

[Pla98] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research (MSR), April 1998.

[Por80] M. F. Porter. An algorithm for suffix striping. *Program*, 14(3):130–137, 1980.

[Ran71] W. M. Rand. Objective criteria for the evaluation of clustering methods. *American Statistical Association Journal*, 66(336):846–850, 1971.

[Ris] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*.

[SOHB07] Sanjay C. Sood, Sara H. Owsley, Kristian J. Hammond, and Larry Birnbaum. Tagassist: Automatic tag suggestion for blog posts. 2007.

[SS00] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135, 2000.

[SSMB00] B. Scholkopf, A. J. Smola, K. R. Muller, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

[ST96] Abraham Silberschatz and Alexander Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):970–974, December 1996.

[ST00] Noam Slonim and Naftali Tishby. Document clustering using word clusters via the information bottleneck method. In *SIGIR*, pages 208–215, 2000.

[str02] Fjording the stream: An architecture for queries over streaming sensor data. February 26 2002.

[SVvL06] Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. Item sets that compress. In Joydeep Ghosh, Diane Lambert, David B. Skillicorn, and Jaideep Srivastava, editors, *SDM*. SIAM, 2006.

[SWY75] G. Salton, A. Wong, and A. C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18:229–237, 1975.

[TBHG00] Pang-Ning Tan, Hannah Blau, Steven A. Harp, and Robert P. Goldman. Textual data mining of service center call records. In *KDD*, pages 417–423, 2000.

[TCCL07] Meng-Hsiun Tsai, Jun-Dong Chang, Sheng-Hsiung Chiu, and Ching-Hao Lai. Identification of marker genes discriminating the pathological stages in ovarian carcinoma by using support vector machine and systems biology. In Marcus Randall, Hussein A. Abbass, and Janet Wiles, editors, *ACAL*, volume 4828 of *Lecture Notes in Computer Science*, pages 381–389. Springer, 2007.

[TK07] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *IJDWM*, 3(3):1–13, 2007.

[tLjL03] Hsuan tien Lin and Chih jen Lin. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods, May 21 2003.

[VC74] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979).

[vLS08] Matthijs van Leeuwen and Arno Siebes. Streamkrimp: Detecting change in data streams. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *ECML/PKDD (1)*, volume 5211 of *Lecture Notes in Computer Science*, pages 672–687. Springer, 2008.

[WF05] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann, 2005.

[wHcCjL03] Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification chih-wei hsu, chih-chung chang, and chih-jen lin, October 29 2003.

[WK96] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[Yan99] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1-2):69–90, 1999.

[YP97] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning*, pages 412–420, 1997.

[Zak98] Mohammed J. Zaki. Theoretical foundations of association rules, February 08 1998.

[ZZ07] M. L. Zhang and Z. H. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, July 2007.