Research Paper

Clustering of Mailing List Discussions for Information Retrieval

Steffen Lortz

2010/07/19

Knowledge Systems Laboratory, Prof. Dr. Takashi Yukawa
Department of Electrical Engineering
Nagaoka University of Technology

Knowledge Engineering Group, Prof. Dr. Johannes Fürnkranz
Technische Universität Darmstadt
Fachbereich Informatik

Abstract

Recently, threaded discussion communities, such as web forums and mailing lists, have become increasingly popular with Internet users. Millions of registered members converse about leading-edge scientific issues or just about trivial matters of their every day life. This makes data bases of these communities a valuable source for information, but their colloquial texts are hard to search and summarize with traditional retrieval methods. For that reason, several attempts have been made to optimize established techniques of topic detection and tracking for use with casual language. If the underlying topics could be made visible to information retrieval systems and to human users alike, it would enable greater benefits from the huge amount of hardly structured data. This work proposes a general approach to make up data from mailing list community archives. It utilizes techniques from traditional topic detection and tracking and optimizes them for texts, written in casual language. Another particular focus has been on the proper labeling of generated clusters with comprehensible names to enable user interaction with the structured data. Further, the effect of searching a mailing list by topic in contrast to common retrieval on a document basis is analyzed. Finally, experiments on a prototype implementation show, that developed enhancements in fact increase the quality of topic recognition as well as grouping threaded discussions by their implicit subject can improve information retrieval, especially for human users.

Contents

1 Introduction **1**

2 Related Work **3**

3 Clustering of Mailing List Discussions **4**
3.1 Application Overview 4
3.2 Preprocessing 5
3.3 Clustering Algorithm 7
3.4 Similarity Criteria 10
3.5 Cluster Labeling 12
3.6 Cluster Retrieval Method 14
3.7 Displaying Results 15

4 Experiments **17**
4.1 Corpus 17
4.2 Evaluation Methodology 18
4.3 Results 20

5 Conclusion **24**

1 Introduction

During the last years, threaded discussion communities have become an important part of the Internet. Every day, millions of users are talking about a wide variety of topics using web forums or mailing lists. For example the big English community Offtopic.com currently has a user base of about 220,000 members, who have already written more than 130,000,000 posts in 4,500,000 threads and they add over 30,000 new posts every day [1]. The huge amount of content created by users of those platforms is a valuable source of information, but it is also hard to summarize and search. High quality information retrieval algorithms for common web pages often rely on proper format and language to extract important features from a text. However posts in web forums or on mailing lists are often full of spelling mistakes and use slang words or uncommon abbreviations. Nevertheless, threaded discussion communities can provide up to date information and insights into topics not covered by the news or other web pages. This can be interesting, in particular for news reporters looking for stories or specialists gathering information about their field of work, but also for everybody who just wants to spend some quality time on reading about a topic of interest. Therefore, existing methods for information retrieval have to be adapted to enable successful work in that new domain.

Today, most Internet users are familiar with using web forums or mailing lists to ask for advice or just talk about everyday matters. Users of the same forum provider or mailing list are called its community and each arbitrary conversation in such communities is represented as a thread. That is an initial email to a mailing list or a post in a forum, which proposes a question or thought as a subject for discussion and a sequence of replies to it. Although the discussed matters within one thread are usually closely related to each other, it is not given that a single thread is just about one topic. Also, the same topic may be spread over separate threads. For example a conversation about destroyed crops in the garden might well turn into a discussion about how to get rid of snails after they have been identified as the cause of the initial problem. On the other hand, people with the same or a similar issue might not be able to find the existing thread and just open a new one about the same topic.

A well established field of information retrieval, called "Topic Detection and Tracking" (TDT), addresses the issue of associating documents with a certain topic. It performs quite well on its original domain, the articles of Internet news portals. The advantage of TDT over a usual web search like that of Google [2] is, that documents are grouped together based on their implicit topic rather than being delivered as an unstructured list of texts, matching a few query words. For example, a reporter writing an article about an accident on an oil rig needs to get information, not only about the event itself, but also about related subsequent events like the environmental issues caused by the leaking oil. If there are a lot of documents, it may be difficult or impossible for him to manually search news portals to find all coherent articles. So he could make use of TDT to get an automatically composed group of articles instead, which are all related to the topic of the particular oil spill disaster. Hence, TDT provides a more structured view on the

data than simple full text search algorithms. Instead of an unstructured document list, information mining can be done based on topics.

When Topic Detection and Tracking was first introduced as a competition by the National Institute of Standards and Technology (NIST) [3], it was aiming at organization of news articles. Although being written in natural language, these are usually free of language errors or ambiguities in their content. They also have a clear structure, mention time and place of events, repeat important facts or refer to related documents. All those qualities greatly simplify the process of TDT on news articles. For threaded discussion communities on the other hand, the language is much more casual and often full of spelling and grammar mistakes. The users do not pay much attention to their writing style or have a low language proficiency, for example because they are no native speakers of the community's conversational language. Additional use of slang words or unusual abbreviations makes some threads hard to understand, even for unexperienced human readers. Besides the language, subjects of threads might be unrelated to the topic or just provide it implicitly. "Just another lie of a politician" for example could express the anger a person feels about a recent event in politics, but it is not clear what the discussion really is about, until the full text is known. Further, a thread might be about different topics simultaneously or change its topic over time, like stated in a previous example. Because of that, TDT on an Internet forum or a mailing list is a more sophisticated problem than that on news articles. However, if it succeeded, users could get a good overview over the usually huge amount of data present in a big discussion community and they would not have to rely on search engines, that require complicated queries to return good results. If the topics derived from a document collection were also named automatically, the TDT approach could provide a completely different view on the discussion data.

In this work, a general approach to make up data from mailing list community archives is developed. The proposed method utilizes different techniques of information retrieval and TDT for classifying threaded discussions on mailing lists into topic based clusters. It combines different preprocessing steps of posts with a clustering algorithm, that takes into account not only text similarity but also user similarity of threads. Further, the created clusters are named and can be searched for by using a ranked retrieval method. Finally, results are displayed to a user in an output style typical for search engines or alternatively, the mailing list archives can be viewed "by topic" in addition to the already provided views "by thread", "by subject", "by author" or "by date".

The remainder of this document is structured as follows. First, related work in the field of information retrieval is presented. Afterwards, the clustering creation, cluster labeling and the retrieval method are explained and evaluated. Problems that occurred during the development are also mentioned in that section. Finally, a conclusion states benefits from and weaknesses of the current solution and shows possible ways to improve it in future work.

2 Related Work

One study about TDT on news articles especially important to this work has been done by Makkonen et al. [4]. They tried to achieve better results by using document similarity measures based on implicit data in addition to pure text comparison. Examples of such meta information for a certain event are the time of its occurrence, its location and names of persons involved. However, extracting such data from casual language is even more challenging than reading it from well-formed news texts.

Previous works about TDT in email have been done by Gabor Cselle [5] and Arun Surendran [6]. Both developed applications, which semantically group emails together and provide the user with a named list of those clusters. Surendran especially focused on the process of cluster naming while using simple term frequency inverse document frequency (TF-IDF) based similarity measures for clustering. Cselle tried to use meta data of emails, similar to the method of Makkonen, to improve cluster quality, but also spent considerable effort on cluster naming. Both approaches were user centric, which means that they always had a single mailbox owner in mind, whose personal preferences should be adhered to. That often results in cluster names where certain properties like names of active senders are emphasized. For a general service on a public platform like a mailing list, such information is unimportant, because usually no personal relationship to the other authors exists. Nevertheless, many of their ideas can also be adapted for use on publicly accessible mailing list archives.

Mingliang Zhu et al. [7] published a conference paper about TDT for threaded discussion communities with a focus on large bulletin board systems. They assume different weights for terms, depending on their position in a thread. Thus, words in a subject line are emphasized while late words in a long text are ignored. The advantage of this method is faster indexing and comparison, because less words are considered. In addition, Zhu assumes that users usually reply to threads, which match their particular interests. Thus, if a user, who usually only replies to threads about cars, it should be safe to assume, that a new thread in which the user is active also is about cars. Other than the previous studies, Zhu does not provide a user interface to the resulting clusters, neither does he name them.

Besides these works, which are mainly cited in this document, there are some other notable studies. Efficient clustering and on-line-clustering algorithms are introduced and compared in [8, 9]. Beringer [9] also proposes an efficient, noise canceling document comparison method. More about processing natural language texts for TDT can be found in [10]. Its focus is on proper noun recognition, which is exceptionally hard in texts, that are not properly cased. The National Institute for Standards and Technology and Manning et al. [3, 11] suggest different approaches for evaluation. The latter also provides a lot of general information about clustering and ranked retrieval algorithms and paradigms. In addition to those publications, many tools have been utilized for stemming, part of speech tagging and nominalization [12, 13, 14, 15, 16, 17].

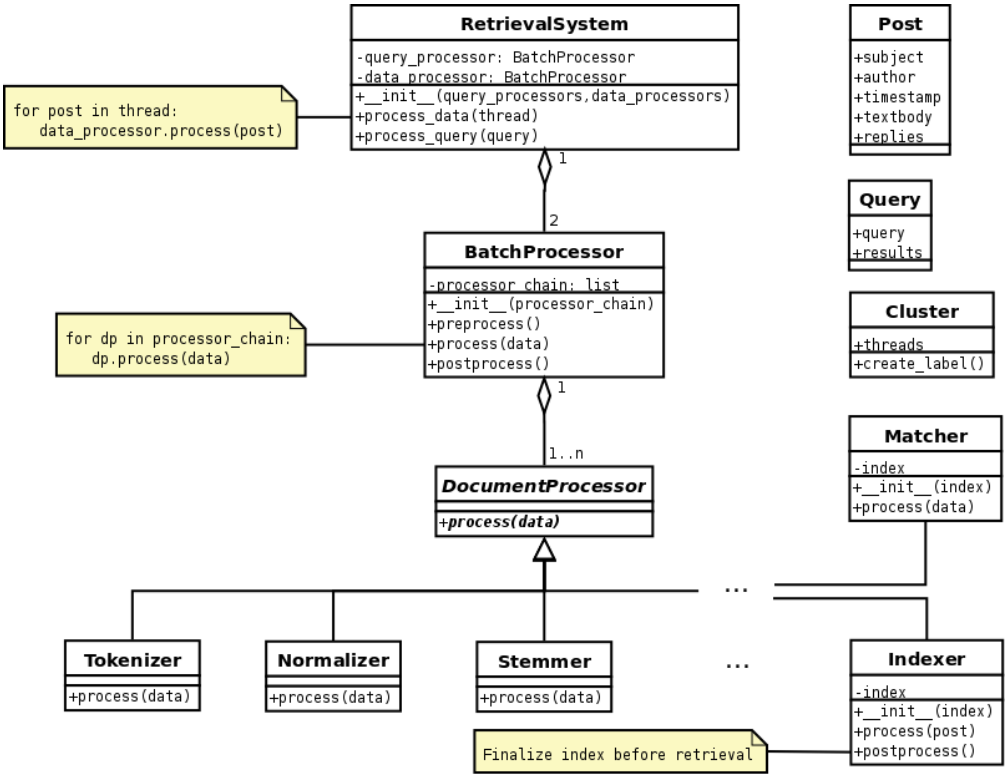

Figure 1: The information retrieval system's program architecture.

3 Clustering of Mailing List Discussions

This section introduces the clustering algorithm and its prototype implementation. At first, it is necessary to define some vocabulary related to the domain of threaded discussion communities. Every time a user says something, it is considered a post. A thread is a usually coherent sequence of posts, of which the first post is called entry and all subsequent posts are called replies. A post always has a subject, an author, a time-stamp, denoting when it was committed, a text-body and a list of direct replies. Direct replies to a post are those, which were created using the "reply to" option of that specific post. If a data set meets these requirements, the algorithm explained in this section is applicable to organize it into topic-based clusters.

3.1 Application Overview

The basic application architecture of the prototype is shown in figure 1. For the implementation prototype, an information retrieval system consists of a data processor and a query processor. Both are defined as chains of document processor modules, which perform data transformation, preprocessing and finally indexing or query matching.

Therefore, they work on queries and posts alike, to assert consistent transformations of both. The document processors will be applied in the same sequence, that is given by their position in the processor chain. This architecture has been chosen, because it eases the composition and testing of different setups. Except for indexers, all processors developed for this prototype are stateless. That means, that all operations on a single data item are done without any global knowledge about other documents. Nevertheless, the framework supports three processing phases: Preprocessing, which is done just once when the system is started, processing, which is called once for every data item, and finally post-processing, which is called just before a query is processed after new posts were added. Thus, it would be possible to support stateful document processors in the current framework as well, although it has not been necessary so far, except for indexers.

The basic document units, which are indexed, clustered and retrieved, are threads. Although preprocessing is done post-wise, threads are never split up, since much information would become incomprehensible or misleading, if it was separated from its original context. Thus, only entries are referenced in the document index, which is used to build a clustering from. The whole thread structure still can be accessed by recursively traversing the replies of those entries. Queries are represented as a simple line of free text and have their retrieved results attached by the matcher. Finally, clusters are just collections of threads, that have an additional method to create their own name. Depending on the utilized retrieval algorithm, threads and clusters can have one or more additional objects for similarity measures attached.

The implementation prototype has been developed and tested using archive data from a technical mailing list [18]. Several reasons led to that decision, such as the free and easy availability of complete archive files and the proficiency of this work's author with the discussed topics. However, with a suitable parser, any source for threaded discussions could be used, as long as their posts come with subject, author, time-stamp, text-body and direct replies. In general, the application framework provides a flexible base for the experiments of this work and should also be customizable enough, to serve for similar projects in the future.

3.2 Preprocessing

As stated during the introduction, texts in threaded discussion communities usually are sloppily written. Thus, normalizing their style and removing noise is especially important, before they are clustered. After parsing the document collection into a suitable data structure, there are a couple of document processors, which clean up and modify the subject and text-body of posts and queries alike, to maximize the efficiency of the information retrieval system. Those are introduced in this section.

Noise Removal Emails usually contain a lot of uninformative noise, which, if not removed, would alter a thread's characteristics. Examples for such noise are greetings and signatures, advertisements from mail service providers, citations of earlier posts in the

same thread, PGP-keys and many more. Especially in short emails, noise can have a lot of influence. Thus, a filter removes all lines which are likely to be not relevant from a posts text-body before further processing begins. Since also valuable information might be filtered, a proper balance between noise removal and data loss has to be established.

Tokenization The tokenizer used in this study splits text on whitespace and on any non-alphanumeric character, if it produces words of at least two characters in length. For example "well-formatted" would be split into the tokens "well" and "formatted", whereas "U.S.A." would remain unchanged. There have been mainly two reasons to split on other characters than whitespace. First, due to the informal writing style in email, commas and periods are not always followed by a whitespace, e.g. "I said before,that should be fixed". Second, for a technical mailing list, there are many relevant terms, that are embedded into longer strings like filenames or version numbers, for example "/usr/bin/firefox". Hence, if words were not split on any special character, the amount of compound terms, that do not contribute to indexing or search would increase a lot.

Normalization Normalization processors try to reduce variances in between words of text-bodies of different threads. Procedures include transforming every word to lower case, removal of all non-alphanumeric characters and also by filtering words, which are too short or too long to be valuable search candidates. Stop-words, i.e. very frequent words in all threads, are also removed during a normalization step. The list of stop-words contains about 200 terms and is made up by uninformative words, taken from a predefined stop-list provided by [12] and the most frequent words in the corpus. Whether numbers or words containing numerals should be removed like in [5] is an arguable question, since version numbers seem to be quite important in a mailing list discussing about computer software. However, experiments suggested, that numbers with more than one numeral should also be ignored.

Stemming To further increase the chance of matching words with the same meaning but different shapes, e.g. "fails" and "failing", stemming is applied. By doing so, both words from the example would be reduced to the common stem "fail", thus establishing a relation between each other's threads. For this prototype, the Porter2-stemmer [12] has been used.

One issue with mailing list discussions, that cannot be compensated for is that some email clients or users show bad habits, which break the usual thread structure. If starting an intentionally new thread, some users tend to reply to an existing thread to have the receiver's address set. Then they remove title and text and enter their own content. Since the email client will correctly set the header references for a regular reply, the post will be added to the thread to which the user had replied. This is called "thread hijacking" in the community and leads to totally independent discussions within a single thread. Also, there are some mail clients that do not set header references if replying to a thread. In this case, one formal thread will be split up into many parts. It is not

always possible to find all parts, which belong together, especially for human readers. While the first issue mentioned is a problem to TTD, since topics cannot be properly separated, the second one showed to be handled well by the clustering algorithm itself.

After the posts have been cleaned and normalized, they are ready for indexing and clustering. Figure 2 shows, how a usual email body is transformed.

> Brad Brent wrote:
> > You could use Skype. It is free and should do well for you.
> > Just give it a try.
> >
> > Cheers!
> > BB
>
> I have just a **coupl**e of **question**s for those of you, who have **use**d **Skype**.
> Is it **free** to make **intern**ational **call**s, **even though** it **seem**s they
> are **free**, is this **true**?
> Are they **upfront** on what their **ad**s **say** or not...?
> I have **never use**d **Skype**, so I would like to know **befor**e I get myself
> into a **problem**...
>
> –
> Jack Jones
> Powered by Debian/GNU/Linux
> by Ubuntu ver 9.10 Karmic
> Registered Linux User No. 0815
> GPG Fingerprint: AFE2 B222 E11B 8F9F C226 EFAC 7D39 30A5 3212
> Public Key available from subkeys.pgp.net

Figure 2: Effect of preprocessing on a post's text body. Only the parts marked bold will be indexed and everything else will be discarded. 'Skype' will be further normalized to lower case 'skype'. For this example, preprocessing reduced the amount of words relevant to indexing from 116 to 21.

3.3 Clustering Algorithm

The clustering algorithm used for the prototype is group-average agglomerative clustering (GAAC) as it is presented in [11] with a few adaptions. These were necessary to be able to use the algorithm repeatedly, as new threads are added to the document collection over time. Although starting with an initial clustering does not create as good results as the construction from scratch, it assigns a new thread to its closest existing cluster or creates a new cluster for it, which is basically the same as other on-line-algorithms do [9, 5]. The pseudocode is shown in algorithm 1. Beginning from an initial clustering, the algorithm will merge the two most similar clusters in each iteration until their similarity is below a predefined threshold or only one cluster is left. Given this behavior, GAAC naturally is a hard clustering scheme, because it assigns every thread

Algorithm 1 The hierarchical agglomerative clustering algorithm.

1: **for** $n = 0$ to $N - 1$ **do**
2: **for** $i = n + 1$ to $N - 1$ **do**
3: $C[n][i].sim \leftarrow sim(n, i)$
4: $C[n][i].index \leftarrow i$
5: $C[i][n].sim \leftarrow sim(n, i)$
6: $C[i][n].index \leftarrow n$
7: **end for**
8: $I[n] \leftarrow 1$
9: $P[n] \leftarrow SimPriorityQueue(C[n])$
10: **end for**
11: **for** $k = 0$ to $N - 2$ **do**
12: $k_1 \leftarrow argmax_{\{k:I[k]=1\}} P[k].Max().sim$
13: **if** $P[k_1].Max().sim < threshold$ **then**
14: $break$
15: **end if**
16: $k_2 \leftarrow P[k_1].Max().index$
17: $index[k_1].extend(index[k_2])$
18: $I[k_2] \leftarrow 0$
19: $P[k_1] \leftarrow []$
20: **for all** $i \in \{i : I[i] = 1 \wedge i \neq k_1\}$ **do**
21: $P[i].delete(C[i][k_1])$
22: $P[i].delete(C[i][k_2])$
23: $C[i][k_1].sim = sim(i, k_1)$
24: $P[i].insert(C[i][k_1])$
25: $C[k_1][i].sim = sim(i, k_1)$
26: $P[k_1].insert(C[k_1][i])$
27: **end for**
28: **end for**
29: **for** $i = N - 1$ downto 0 **do**
30: **if** $I[i] = 0$ **then**
31: $index.delete(i)$
32: **end if**
33: **end for**

Where N denotes the size of $index$, I is used to keep track of which clusters are still available for merging, C contains precomputed similarities for all pairs of clusters in $index$ and P holds a priority queue for each cluster, which contains the similarities to other clusters, sorted by similarity. $argmax_{set}exp$ returns the parameter of set for which exp evaluates to the biggest value and $sim(a, b)$ computes the similarity of two clusters, given as their position in $index$. Finally, $threshold$ defines the lower bound for document similarity, below which further merging of clusters is stopped and the clustering is finished.

to one and to only one cluster, in contrast to a soft clustering, that allows for multiple memberships. An initial clustering either consists of singleton clusters or it is given as a previously created clustering with additional new threads. The used similarity criteria will be introduced in section 3.4.

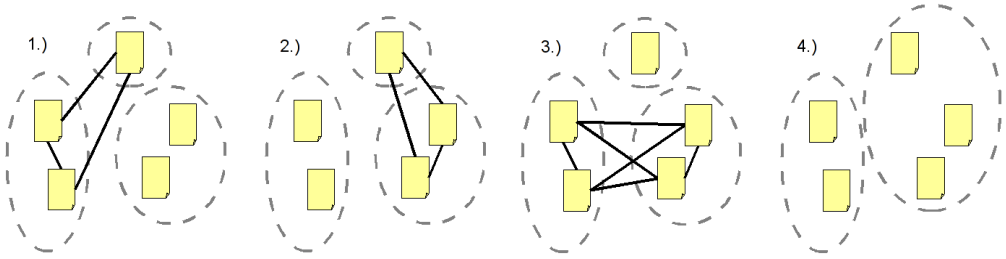

Figure 3: The GAAC document comparison strategy. Trough 1 to 3, the average distances of the document pairs depicted by the black lines are computed. The two clusters for which this distance is minimal are merged in step 4.

GAAC has been chosen for different reasons. Manning [11] recommends this clustering algorithm, because it produces the best results for most applications. To determine the best merge candidates, all similarities of pairs of documents in the prospective cluster are considered, as shown in figure 3. So the cluster, which has the best average similarity in between all of its own threads, will be created. Compared to other systems, like single link clustering, which merges the two clusters which have the most similar members, the clustering created by GAAC is not as sensitive to the distribution of documents in their similarity vector space. GAAC is also a hierarchical agglomerative clustering, which means that it combines documents and clusters into bigger clusters until a certain criterion is met. Thus, we do not need to specify a number of clusters in advance, which would be necessary for flat clustering algorithms like K-means. Further, GAAC is deterministic and repeated executions of tests will always yield the same results, which is crucial to optimize the system's parameters. A drawback of hierarchical agglomerative clustering algorithms is their high time complexity, which is at least quadratic in the number of documents compared. Though the complexity of $\Theta(N^2 logN)$ should be just a minor issue for document collections of moderate size, if an efficient variant with a cache for comparisons is used, as it is the case for the prototype implementation.

Before deciding for GAAC, an adaptive K-means algorithm as described in [9] was used and turned out to be unbearably slow. Most time was consumed by computations of the similarity between clusters and threads, which could not be cached appropriately. While K-means usually could be quite efficient, the on-line-version of that algorithm needs too many comparisons to determine whether new clusters should be added or old ones should be removed. Possibly, the efficient similarity measure introduced by Beringer in the same report, would have sped up the clustering process. Though, it just uses approximates to the real documents, thus the results may have varied from an exact

matching. Since the implementation of that method would have been complicated, too, there have been too many risks, because of which that approach has not been tested.

3.4 Similarity Criteria

The similarity criteria used to compare documents are a very important part of clustering algorithms, since they mainly decide the distribution and quality of the resulting clusters. According to [7], TF-IDF based text similarity measures are the most common in traditional TDT on news articles. Considering the casual writing style and incomplete, implicit information in conversational language, additional features should be exploited to complement the pure text similarity. [5, 7] suggest, that certain users preferably talk about certain topics. Therefore, another similarity measure, derived from the active users in a thread, is also used in the prototype.

To compare threads based on their textual content, the words from the cleaned and normalized text-bodies are transformed into TF-IDF vector representations. For each term t and each thread d, the number of occurrences of t in d, denoted as term frequency $tf_{t,d}$, is determined and saved into a vector attached to the thread. That vector has as many entries as there are different words in the document collection. Those, which do not appear in a thread, yield a value of 0 in the respective term frequency vector. Words from the subjects of each post are also added to the same vector, emphasized by the amount of posts, which use the respective term in their title. Since subjects are sometimes changed to account for topic transitions in the thread, this makes the most frequently used subject the most important. In addition, a global vector is maintained, that holds for every word t the document frequency df_t, which denotes the number of threads, that t occurs in. The document frequency vector is used to normalize the term-frequency vectors in a way, that a word, which appears in nearly every thread, does not weight as much as a more distinctive term, that is very rare throughout the document collection. The resulting weight for a term t in a thread d, is computed as follows:

$$tfidf_{t,d} = tf_{t,d} \times log\frac{N}{df_t}$$

where N is the total number of documents in the document collection.

The prototype implementation maintains for each thread d a length-normalized vector containing $tfidf_{t,d}$ for all words t in the document collection. Length-normalized in this case means, that the vectors are divided by their own length to become unit vectors. Given document representations as introduced, the group-average similarity measure defined in [11] can be used to compute cluster similarity. For two clusters ω_i, ω_j with the respective length N_i, N_j, the group-average similarity is defined as

$$\mathsf{SIMGA}(\omega_i, \omega_j) = \frac{\left(\sum_{d \in \omega_i \cup \omega_j} \overrightarrow{d}\right)^2 - (N_i + N_j)}{(N_i + N_j)(N_i + N_j - 1)}. \quad (1)$$

where $\overrightarrow{d}$ are the unit-vector representations of the documents in both clusters.

If the above similarity measure is used to implement $sim(a, b)$ of the clustering algorithm on page 8, it will create text based clusterings. However, the author similarity has not been considered yet. Zhu's approach [7] to determine thread similarity based on user data is almost the same as for text based similarity. He introduces a user frequency $uf_{u,d}$, which denotes for every thread d and user u the number of posts, that u wrote. As for TF-IDF, the user frequency is complemented by a global thread/user frequency tuf_u for each user u, which counts the total number of threads, that u has participated in. The two values are then combined into a single weight, which emphasizes locally active users over users, who virtually answer every thread. The user frequency-inverse thread/user frequency (UF-ITUF) for a thread d and a user u is defined as

$$ufituf_{u,d} = uf_{u,d} \times log\frac{N}{tuf_u}$$

where N is the total number of threads in the document collection.

As for the TF-IDF values, the UF-ITUF values are stored in vectors respectively. These are also length-normalized to become unit-vectors and thus can be used with equation (1). Now, the clustering algorithm could also be used to cluster threads by their author similarity. Next, the two methods need to be combined.

Makkonen, who did multi-feature comparison on news articles [4], used the weighted sum of similarities for content, time, location and persons, to cluster articles. His method is simple and easy to implement. However, first attempts to use the same approach to combine TF-IDF and UF-ITUF measures resulted in worse clusterings than the pure text similarity. Zhu [7], in contrast, used a two phased comparison as depicted in figure 4, where he first builds clusters with text based similarity and then checks for author similarity of just a few remaining merge-candidates, which were close, but not sufficiently similar to already have been combined. He notes, that user similarity is valuable to emphasize merges in question, but not to decide them in general. This seems obvious, if very short threads, e.g. such without replies, are created by the same user. The author similarity for those always will be very high, such that even with a low weight, it gets a big influence on clustering decisions, which led to the worse results with Makkonen's method. Hence, Zhu's method to involve author similarities into the decision process of the clustering algorithm is used in the prototype implementation.

Cselle also tried to utilize timestamps [5] to improve clustering decisions by telling that topics without recent replies are obsolete and not to continue, but stated that this approach did not improve their results significantly. In contrast to the private conversations of one email user, the topics of discussions in a mailing list could last much longer or be reactivated after a long time, if another user encounters the same or a similar problem. So it is likely, that time similarity is even less important to TDT in discussion communities and thus, not considered in this study.

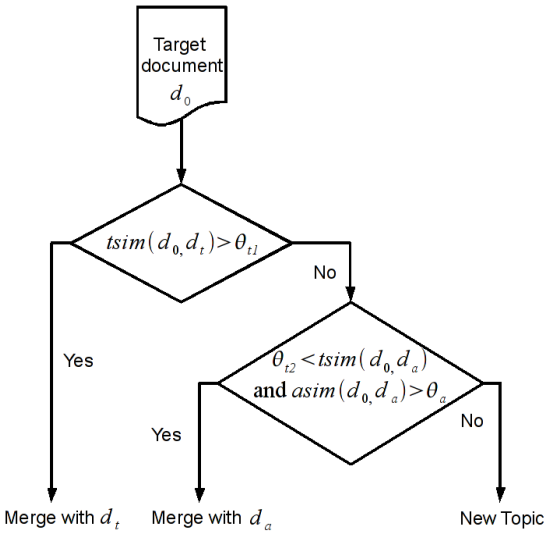

Figure 4: Two-phase merging decision. d_0 is the cluster to be merged, d_t is the cluster most similar to d_0 in terms of text, while d_a has the best author similarity to d_0. θ_{t1}, θ_{t2} with $\theta_{t2} < \theta_{t1}$ and θ_a are threshold values, defining the minimum necessary similarity to merge two clusters.

Given the definitions of this section, the function $sim(a, b)$ of the clustering algorithm on page 8 can be implemented and thus, the clustering algorithm is complete. It now can create clusterings of a collection of threads, based on text similarity and author similarity.

3.5 Cluster Labeling

When clusters of threads are presented to users of discussion communities, they need to be automatically labeled with a textual name to represent their inherent topic. Otherwise users would have to read subjects or even the content of assigned threads, to know what a cluster is about. In this section a method for automated cluster labeling is proposed. To assign a label to a cluster, we distinguish two cases:

1. If the cluster only consists of a single thread and this thread's subject contains at least a given number of descriptive words, it is set as the clusters title with prefixes like "OT", "RE", "FWD" and so on removed. For small clusters, this produces better results than automated label construction, since they usually do not have enough words in their text body to have a title determined statistically. Further, a human crafted subject is usually easier to understand than a list of terms.

2. If the cluster contains more than one thread or the subject words of the single thread are not descriptive, all nouns, numbers and verbs from all subjects and text-bodies in the cluster are collected and ordered by relevance. However, words

POS-Tagger	Errors	Error Probability
Stanford POS-tagger	52	7%
TreeTagger	61	8.2%
NLTK POS-tagger	262	35%

Table 1: Test results for part-of-speech taggers. The errors are out of 741 words in the test set.

of length three or less are filtered out, since they often do not have a descriptive meaning. To measure a words relevance, the cluster's centroid TF-IDF vector is used together with weights depending on whether the word is part of a thread's subject or of a certain type, like "verb", "number" or "noun". The word with the highest score and a number of words with a weight of at least half of it are collected as the most relevant words. They are then nominalized and ordered by their relative position in texts. Finally, a cluster label is constructed as a whitespace separated string of these nouns.

Also according to [5, 6], the created title for complex clusters is entirely constructed from nouns. However, it has shown during experiments, that sometimes nouns present in a cluster alone could not properly describe its topic. For example a cluster with various threads about problems with hibernation did only contain verbs like "to hibernate" or "to suspend", not the related noun "hibernation". Therefore the prototype uses a nominalization algorithm to convert verbs to nouns, where the WordNet-database [13] is used to find possible candidate nouns. These are then filtered by suffixes, that denote actors and the remaining words are emphasized based on their ending. For example when nominalizing "to install", possible candidates might be "installer", "installment", "installation". "installer" will be filtered, because the suffix "-er" denotes an actor, which usually does not reflect the action expressed with "to install". Further, the ending "-ion" will be emphasized, because doing so often led to good results in experiments. Thus, "installation" will be chosen as nominalization of "to install".

To distinguish the type of a word, a part-of-speech (POS) tagger is used. POS taggers usually rely on proper language and capitalization to extract the role of a word from its context. Hence, information about the precision of available taggers does not apply for texts from the document collection. To find a suitable POS tagger, three [14, 15, 16] have been tested tagging the entries of 100 threads and manually checked for their performance. Only three word groups important to cluster labeling have been taken into account: Nouns, proper nouns and verbs. The test results for the POS-tagger test are shown in table 1. Of a total of 741 relevant words, the Stanford POS-tagger tagged only 52 wrong. Although it made the least errors, it still has trouble to distinguish nouns and proper nouns if they are wrong cased. To further increase the success ratio of POS tagging, only the most frequently assigned tag for each word stem within the cluster to label is considered.

```
seamonkey plugins, Size: 5
"SM 2.0rc1+No Plugins On Karmix Beta 64 Bit"
"Seamonkey Addons"
"[Karmic] SeaMonkey + 64bit +plugins"
"[Karmic] SeaMonkey + 64bit +plugins"
"[Ubuntu Karmic Beta] Java + SeaMonkey 2.x"

sound, Size: 4
"9.04 sound issues"
"No more sound with 2.6.28-15 (Jaunty)"
"No sound after upgrade...???"
"sound&video"
```

Figure 5: Examples of generated cluster labels and subjects of their contained threads.

To sort the final list of words within the title-string, the method proposed by [5] is used. When indexing the words, the index of the first occurrence $fo_{w,d}$ of each word stem w in a thread d is stored. Later, this data is used to compute a relation over all pairs of words w_1, w_2 in a single cluster c as follows:

$$w_1 \preceq w_2 \Leftrightarrow \sum_{d \in c} compare(w_1, w_2, d) \geq 0 \tag{2}$$

$$compare(w_1, w_2, d) = \begin{cases} 1, & \text{if } w_1 \in d \wedge w_2 \in d \wedge fo_{w_1,d} < fo_{w_2,d}, \\ -1, & \text{if } w_1 \in d \wedge w_2 \in d \wedge fo_{w_1,d} > fo_{w_2,d}, \\ 0, & \text{otherwise.} \end{cases}$$

In the final title of a cluster c, a word w_1 is placed before w_2, if and only if $w_1 \preceq w_2$ with the definition in equation (2) holds. This is, if w_1 usually occurs before w_2 in text-bodies of the cluster.

The labeling method proposed in this section is cluster internal, which means that words are chosen independently from other clusters. Although the resulting names may not be as distinctive as those of differential labeling, they proved to be sufficiently descriptive in [5, 6]. Provided with proper titles, clusters can ease search for certain topics or increase the efficiency, with which document collections are navigated by human users. Examples of generated cluster labels are shown in figure 5.

3.6 Cluster Retrieval Method

Now, that we have a topic-based clustering of our document collection, where each cluster has a descriptive name, users may retrieve clusters instead of single threads when looking for information via a search engine. The ability to search the discussion data in general

is vital to the usefulness of technical mailing lists or forums, because many users do not only ask questions, but search for possible answers, in advance. It can be advantageous to search for clusters, because the search results are already structured or some informative threads are returned in clusters, which would not have been rated relevant to the query itself. In this section, a method for matching queries to clusters is explained.

To match clusters to a user query, a centroid TF-IDF vector is maintained for each cluster, derived as the average of the TF-IDF vectors of all contained threads. As stated in the "preprocessing" section 3.2, queries have to be processed in the same way as threads, to convert the words to an identical shape. Without preprocessing, matching would not succeed, since the query's words might not be found in a cluster's TF-IDF vector. The normalized query is then simply transformed to its vector representation and compared with all available clusters. Comparison is done using cosine similarity. For vectors representing a query $\overrightarrow{q}$ and a cluster $\overrightarrow{c}$, the cosine similarity is defined as:

$$sim(\overrightarrow{q}, \overrightarrow{c}) = \frac{\overrightarrow{q} \cdot \overrightarrow{c}}{|\overrightarrow{q}||\overrightarrow{c}|} \tag{3}$$

where the numerator is the dot-product of both vectors and the denominator is the product of their Euclidean lengths.

The denominator should length-normalize the vectors to unit-vectors, but since all used vectors are already normalized by definition, equation (3) can be simplified to

$$sim(\overrightarrow{q}, \overrightarrow{c}) = \overrightarrow{q} \cdot \overrightarrow{c},$$

the simple dot-product of the two vector representations. All threads of clusters with a similarity above a certain threshold are then added to the queries result set. However, if no cluster is sufficiently similar, the clusters with the best scores are returned, because having any results proved to perform better in experiments than not returning anything at all.

With the prototype implementation of the clustering algorithm, cluster labeling and a retrieval method, all requirements for a cluster based retrieval system for human users are met. However, it still lacks a proper display, to be useful for real applications.

3.7 Displaying Results

Finally, a user interface, that enables human interaction with the generated clusterings is required, to put all pieces together. An interface to view and to browse clusterings could also be used to display results of a cluster retrieval operation. With the prototype implementation, a web-page representation of a clustering index or of search results can be easily generated. The design of that page resembles the standard look and feel of the mailing list archive, that was used for experiments [18]. Thus, it should be intuitive for most users, especially if it was used as an additional view on the archive data collections.

(a) View on the index. **(b)** A cluster representation

Figure 6: Different pages of the web-page representation.

The interface shows an index with the named clusters and for each a list of subjects of contained threads, such that users can check for a clusters content, if the name alone does not suffice. This design is taken from best-practices for search engines introduced in [11]. The clusters and threads themselves allow for easy navigation through links, such as “previous” and “next” for “cluster”, “thread” and “post”. Screenshots of the view on a cluster index and a cluster are shown in figure 6.

4 Experiments

In this section, the algorithms introduced in the last section are evaluated using the prototype implementation. First, the corpus and the different evaluation methods are explained. Then, the results of the experiments are presented.

4.1 Corpus

To evaluate the prototype's performance, predefined test data is necessary. Though, even after a thorough search in the Internet, no accessible test set for TDT in threaded discussion communities had been found. Since [5, 7, 6] also created their own corpus, it seems that no official test set exists, so one had to be created. The corpus for the experiments has been obtained from the archives of one popular mailing list about the Ubuntu operating system [18]. It consists of threaded discussions within two months in 2009 and 2010, which are mostly about technical matters like computer hardware and software.

The target clustering and query-answer pairs have been created manually, hence reflect the authors subjective opinion about what are right clusters or answers. However, since users of the system always will be subjective and thus, diverge in their opinions, even imperfect test data should not affect the validity of the results. Query-answer pairs are decided by checking entries, which usually contain a direct or indirect question. If the remainder of the thread answers it, a query is paired with the thread's identifier as one possible answer. The text of this query is composed like it was written by a user with the same problem looking for a solution. If the problem repeats in other threads, they are also added to the query's answer set. Target clusterings are mostly constructed by intuition. In the majority of cases, clusters are about one certain software application or protocol, for example "Firefox", "Samba", but also general topics exist, which discuss problems like "sound issues" or "wireless network". The created test data has then been divided into a training set and a test set, as shown in table 2. The big time gap in between the two months exists to makes sure, that no more thread used for training is still active in the test set and thus, influences the evaluation results.

	Training Set	**Test Set**
Time span	October 2009	April 2010
Posts	2551	2759
Threads	425	376
Posts per thread	6	7.3
Queries	101	103
Target clusters	187	171
Threads per cluster	2.27	2.2

Table 2: Corpus statistics.

4.2 Evaluation Methodology

Clustering algorithm The National Institute of Standards and Technology (NIST) provides detailed guidelines [3] to evaluate TDT on news articles. Those also can be applied to measure the quality of a clustering derived from threaded discussion data. They distinguish two tasks of TDT:

1. *New Topic Detection (NTD)* is defined to be the ability to tell from any document in a stream of documents, whether it starts to discuss a new unknown topic. In our case of threads being the smallest organizable document unit, every thread can either evaluate to "yes", it starts a new topic, or "no", it does not start a new topic.

2. *Topic Tracking (TT)* is defined to be the task of associating all documents in a stream of documents to topics, which are already known to the system. With respect to the NTD task, a topic is known to the system, if a document already has started it. Thus, a topic is defined by the first thread that discusses it. For each pair of topic and thread, the system has to decide whether the thread discusses the topic ("yes") or not ("no"). With the definition of a topic given in this paragraph, threads that start a topic obviously discuss it, too. Hence, if a thread evaluates to "yes" in the NTD task, it also evaluates to "yes" for its associated topic in TT.

Detection quality is defined as a single detection cost, derived from error probabilities for misses and false alarms. A miss is a "yes" instance which has been classified as a "no" instance, thus, a false negative. A false alarm, on the other hand, is a false positive, a "no" instance, which has been classified as "yes". The probability of a miss is computed by dividing the number of false negatives by the overall number of "yes" instances. Respectively, the probability for a false alarm is the number of false positives divided by the overall number of "no" instances. The detection cost value is then computed as

$$C_{det} = C_{miss} \cdot p_{miss} \cdot P_{target} + C_{FA} \cdot p_{FA} \cdot P_{nontarget}$$

where

- C_{miss} and C_{FA} are costs of misses and false alarms,
- p_{miss} and p_{FA} are the probabilities of misses and false alarms and
- P_{target} and $P_{nontarget}$ are the prior probabilities depending on the corpus and the evaluated task.

While p_{miss} and p_{FA} are determined by testing, P_{target} is a constant, which denotes the prior probability for a data item being a "yes" instance. $P_{nontarget}$ simply is defined as $1 - P_{target}$. For the test set used in this study, P_{target} equals 0.455 for the NTD task and 0.00585 for the TT task. The evaluation plan of NIST [3] suggests, that misses should be more dramatic than false alarms. Hence, the cost constant C_{miss} is defined as 1.0 and C_{FA} as 0.2. That means, that unnecessarily creating a new topic is assumed to be not

as bad as missing the creation of a required new topic in the NTD task. Also, assigning a thread to a cluster that it does not belong to is considered less a problem than missing its assignment to a really related topic.

Finally, the detection cost C_{det} is normalized, such that a perfect system scores 0 and a trivial system, which always returns the result, that is more likely for the test set, scores 1.

$$C_{det,norm} = \frac{C_{det}}{denom(C_{miss}, P_{target}, C_{FA}, P_{nontarget})} \tag{4}$$

$$denom(C_{miss}, P_{target}, C_{FA}, P_{nontarget}) = \begin{cases} C_{miss} \cdot P_{target}, & \text{if } P_{target} < P_{nontarget}, \\ C_{FA} \cdot P_{nontarget}, & \text{otherwise.} \end{cases}$$

The minimum normalized detection cost is a suitable metric to compare the quality of clusterings and thus, their constructing algorithms.

Cluster labeling Automatic evaluation of generated cluster labels is difficult, because the content and structure of created clusters differs significantly from the manually composed clusters and cannot be mapped properly. Therefore, names that have been assigned to the test clusters cannot be taken into account to evaluate labels of machine clusterings. However, labeling could be tested on the manual clustering itself, but since these clusters are a result of intuition rather than reasonable matching, the labeling algorithm would probably work unexpectedly. Thus, the quality of cluster labeling is evaluated manually as proposed by Surendran et al. in [6].

Each cluster's label is manually rated using one out of three quality levels:

good for topic labels, which are appropriate and descriptive,

mixed for labels, that are partially good, but also contain undescriptive or misleading words and

bad for labels that do not reflect the content of the cluster.

The overall score is then computed as

$$score = \frac{(N_{good} + 0.5 \cdot N_{mixed})}{(N_{good} + N_{mixed} + N_{bad})}$$

where N_{level} for each $level \in \{good, mixed, bad\}$ is the number of cluster labels that have been assigned $level$. This measure will evaluate to 1, if all labels have been tagged with "good", while an overall "bad" labeling will lead to a score of 0.

Cluster retrieval To evaluate cluster retrieval, precision and recall values for about 100 queries are determined. The acquired values are then combined to a single quality measure called *f-score*, a weighted harmonic mean, to compare the performance of different retrieval systems. It is computed as

$$f\text{-}score = \frac{(C^2 + 1) \cdot prec_{avg} \cdot rec_{avg}}{C^2 \cdot prec_{avg} + rec_{avg}} \tag{5}$$

where

$$prec_{avg} = \frac{1}{|Q|} \sum_{q \in Q} prec(q)$$

and

$$rec_{avg} = \frac{1}{|Q|} \sum_{q \in Q} rec(q).$$

The functions *prec* and *rec* compute for every query q in the tested set of queries Q the precision and recall. The constant factor C can be used to emphasize either one of precision or recall. Values of C bigger than 1 will favor recall, while values in between 0 and 1 put more weight on precision. This work uses $C = 3$ to emphasize the recall, because the ability to retrieve the right documents is more important than not having irrelevant threads listed as well. As long as the latter are not too numerous, it is acceptable for a user to review a few results until a suitable answer to the query is found. The defined score will emphasize high recall and evaluate to 1 for perfect retrieval systems, which just return the desired results and to 0 for trivial systems, which always return all or no documents.

4.3 Results

To find out the influence of the different optimization approaches realized in this study, different setups of the clustering scheme have been tested.

base just performs basic preprocessing steps, like stop word removal and normalization. Only text and subject similarity are considered for merge decisions of clusters.

noise removal extends the base system's preprocessing by the removal of citations, signatures and other noise, that can be removed with its whole line.

full adds the author based similarity measure to the noise removal system.

The graphs in figure 7 on page 23 show their error probabilities. One can easily see, that the noise removal system and the full system are almost identical for most of the time. When the similarity threshold for merging decisions rises, noise removal only achieves better results on new topic detection, while the full system performs a bit better in topic tracking. However, the range where they do differ is less interesting to clustering as it is used in this work, because the average cluster size will not become big enough for practical use, if the required similarity for cluster merging is so high. Thus, author

similarity seems to have no relevant effect to the clustering decisions. Nevertheless, the base system is almost continuously outperformed by the enhanced versions. For topic tracking, only the miss probability is shown, because its distribution is the same as for the false alarm probability. If using a hard clustering algorithm, every time a thread does not follow its intended topic, it is assigned to exactly one false topic, thus, yielding one miss and one false alarm respectively.

System	**NTD**	**TT**	**Average**
Base	0.295	0.491	0.393
Noise Removal	0.276	0.434	0.355
Full	0.291	0.434	0.363

Table 3: Normalized detection costs $C_{det,norm}$ for NTD and TT tasks.

Table 3 shows the normalized detection costs for each clustering scheme. They have been obtained by running each respective system with the optimal parameters, acquired from tests on the training set. For new topic detection, the cost of the noise removal system is lowest, while the full and base system are almost identical. In topic tracking, the base system is much worse than the others. In average, the noise removal system performs best, closely followed by the full system. Relative to the base system, the average cost has been reduced by 9.67% by applying specialized noise removal strategies.

Label quality	**Total**	**Fraction**
Good	28	39.44%
Mixed	27	38.03%
Bad	16	22.54%
Score	**58.45%**	

Table 4: Results for label quality evaluation.

Table 4 displays results of the manual tagging of generated cluster titles. Labels of singleton clusters, which have not been generated automatically, have been ignored for this test. The evaluated clustering had been created from the test set using the noise removal system with the optimal parameters from training. The score of 58% shows, that the majority of labels at least partially covers the content of their associated clusters.

Retrieval System	Average Precision	Average Recall	F-Score
Thread based	48.7%	84.61%	78.8%
Cluster based (base)	36.3%	85.8%	75.5%
Cluster based (noise removal)	34.9%	84.31%	73.9%
Cluster based (full)	34.9%	84.3%	73.8%

Table 5: The evaluation results for cluster based information retrieval.

The results of cluster retrieval evaluation can be seen in table 5. The values have been determined using the optimal parameters identified during training with respect to the f-score, defined in equation (5). The optimized thread based retrieval system has been added as a reference. In comparison to thread based search, the recall of cluster based retrieval is equal or slightly better, whereas precision decreases significantly. The base system seems a little superior to the others, most likely because it had smaller clusters, which better matched the queries.

With the experiments and the evaluation results of this section, benefits from and weaknesses of the current implementation and its theoretical foundation can be discussed.

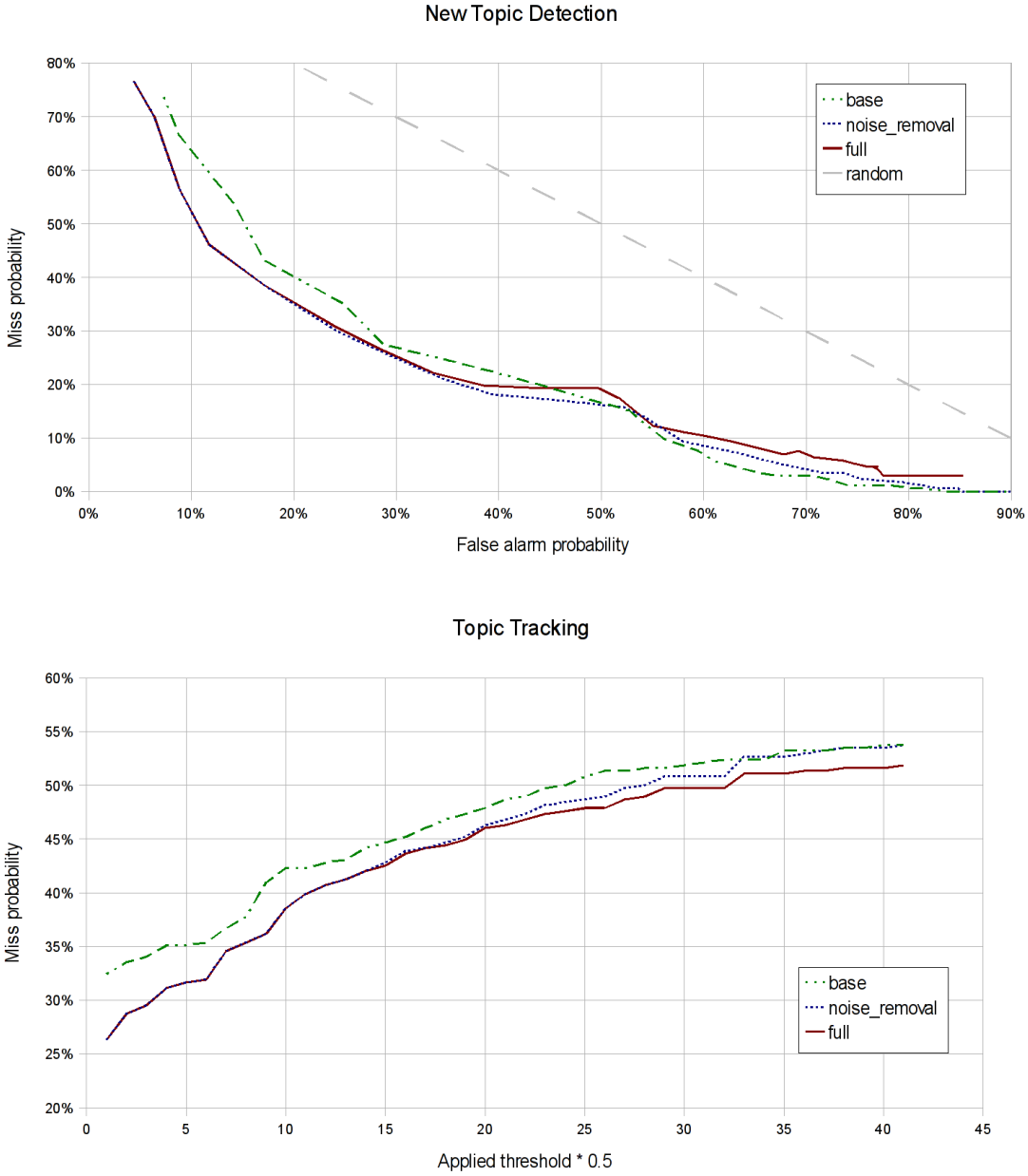

Figure 7: Error probabilities for NTD and TT task.

5 Conclusion

The evaluation results of the clustering algorithm indicate, that broad but appropriate noise removal has the potential to increase the efficiency of topic detection and tracking on a technical mailing list. Due to the structural and textual similarities to many Internet forums or other kinds of threaded discussion communities, these insights of this study should also apply to them. However, before TDT can become a vital part of the user interface in discussion communities, the cluster quality still needs to be improved. The overall cluster size has been quite small, about two threads per cluster. Nevertheless, if the clustering was continued, many false assignments used to happen and these drastically reduced the clustering's quality. If compared to the data used to evaluate the clustering algorithm in [7], the related problem might be the granularity of clusters. While the document collection used in this work is already very focused, namely on a particular operating system, the data processed by Zhu spans totally different topics. Hence, the overall text similarity of threads in between clusters should be significantly higher for the document collection obtained from [18] than for that created from an Internet forum discussing arbitrary topics. Therefore, more and better similarity criteria must be applied to further distinguish the already coherent threads.

One attempt to do so was to incorporate the user similarity of threads, but clustering decisions based on author information did not yield any advantage over pure text comparison with noise removal. Although it worked for tuned parameters on the training set, the test results became worse, if the author similarity was taken into account. However, Zhu and Cselle [7, 5] both found that factor valuable, so why should it be different for this work? One possible explanation could be a structural property especially present in mailing lists. Threads are often very short, for a couple of reasons. Sometimes, nobody answers to an entry, because other community members do not know what to write or do not want to answer at all. Nevertheless, the author of the entry could reply to his own question several times, to keep the thread active and to increase the chance of having it answered. Further, a person, who proposed a question might find a solution by himself and replies to his own thread, which usually closes the discussion. In these cases, the resulting threads are composed by just one author and create very high similarities, if compared to other threads of the same user. A sufficiently low similarity threshold to group threads with multiple authors will result in many undesired combinations of such "monologues". Even the two phased classification heuristic 4 on page 12 does hardly improve the decision quality. Thus, user similarity obviously turns out to be a poor reference for clustering decisions, if grouping threads of a technical mailing list.

Other possibilities, to improve clustering quality, could be the enhancement of text bodies with synonym sets, e.g. using WordNet [13] for selected nouns or verbs. That should increase the similarity of threads, which discuss the same topic with different words. A similar approach would be to expand abbreviations, which are very frequently used in technical mailing lists. However, the latter would require a domain specific data base of such short forms or synonyms, because most abbreviations, that could be found

in the document collection used in this work, had been names of software or protocols. Either way could prove valuable in future work.

Another concern, which needs to be addressed, is the currently hard clustering scheme. As mentioned earlier, threads or even single posts can discuss multiple topics at the same time. Since every thread is assigned to just one single cluster by the algorithm proposed in this work, the distribution over clusters cannot fully reflect the real topic structure. If a thread could belong to several clusters, its inherent topic multiplicity could be accounted for. Thus, implementing a soft clustering algorithm could improve topic tracking for many ambiguous threads, but new training and test data would have to be created, too.

The retrieval of clusters holds several advantages over thread based search. If looking at table 5 on page 22, retrieval of clusters results in a similar recall as thread based retrieval, but the precision suffers from the relatively high amount of returned documents. On the other hand, retrieval of clusters could add some relevant information, which would not have been retrieved directly. This could be observed for the base system, which is slightly better in recall. Therefore, a relevant result, which does not share any vocabulary with the query, could still be found through the connection of the query to other threads in the same cluster. Due to the fact, that clusters are less numerous than threads, the threshold value for ranked retrieval has to be increased to include the centroids of clusters, that contain threads important to a query. By doing so, the chance to classify other nearby clusters relevant increases as well, as shown in figure 8. If an irrelevant cluster is retrieved, it decreases the precision more than an unnecessarily returned thread. Yet, if combined with a user interface depicting the cluster structure, users can easily navigate the results and judge from a cluster title whether it contains relevant documents or not. In case of threads, they have to check at least the subject of each document to know whether it is useful or not. Thus, cluster retrieval can still be advantageous, although the precision is outperformed by thread-based retrieval.

If a higher precision is desired, the document retrieval could be performed in two steps. Since the most likely relevant threads are still those, which are most similar to the query, the results given by clusters could be narrowed with a second retrieval step:

1. Retrieve the clusters closest to the query and create a set of their documents.
2. From the set of documents, retrieve threads, which are close to the query.

For the second step, the similarity threshold to classify a document relevant could be higher than for direct retrieval without suffering from bad precision, because the source documents are already limited. This way, a better trade-off between precision and recall could be achieved as with any of the one step methods. This approach has not been tested, yet. Hence, it remains an option to improve the current methods in future work.

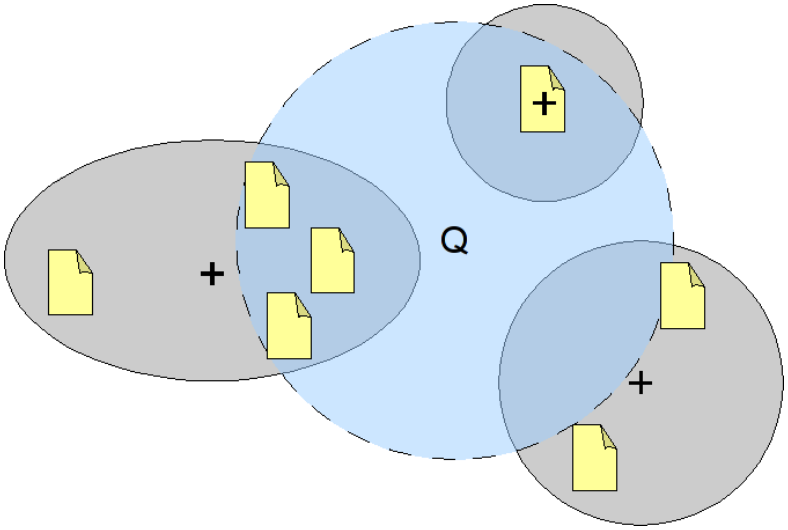

Figure 8: Centroid dilemma in cluster retrieval. The black crosses denote the cluster's centroids and "Q" represents the query's position in vector space. The dashed line shows the similarity threshold, within which results are collected. To reach the desired documents in the left cluster, the query's relevance threshold has to be increased, such that the additional irrelevant cluster to the right is included, too.

The cluster labeling algorithm already performed quite well during experiments, except for few small clusters which mostly have emerged from faulty assignments. However, program names or protocol identifiers are quite rare in candidate lists, although they would make up valuable words, because they often define a topic. On the other hand, many of these proper names are frequent in the whole collection and accordingly low rated. For that reason, it might probably be advantageous to utilize a domain specific name list, to emphasize these terms. This approach could complement the extension of the document preprocessing, mentioned earlier in this section. The data base necessary to expand abbreviations would also suite for this task and lead to a better cluster labeling.

The prototype emerged from this work has the ability to cluster colloquial texts of bad quality based on their inherent topic. Extensive preprocessing increases the quality of its clustering over that of unspecialized topic detection and tracking. Finally, automatic generation of decent cluster labels and an adapted, topic based search algorithm allows users to benefit from the alternate view on the discussion data. The current web-page interface to the cluster structure has been created with the option in mind, that index pages could be created automatically as replies to searches, while the underlying posts remain unchanged. Using PHP and a web-server integration for python, it should be easy to adapt the prototype for practical use. With feedback of human users, further possible improvements could certainly be identified.

References

[1] Big Boards. Statistics for offtopic.com. "`http://www.big-boards.com/board/624/`", 07/2010.

[2] Google Inc. Google. "`http://www.google.com/`".

[3] NIST. The 2004 topic detection and tracking task definition and evaluation plan. Technical report, National Institute of Standards and Technology, 2004.

[4] Juha Makkonen, Helena Ahonen-Myka, and Marko Salmenkivi. Simple semantics in topic detection and tracking. *Inf. Retr.*, 7(3-4):347–368, 2004.

[5] Gabor Cselle, Keno Albrecht, and Roger Wattenhofer. Buzztrack: topic detection and tracking in email. In *IUI '07: Proceedings of the 12th international conference on intelligent user interfaces*, pages 190–197, New York, NY, USA, 2007. ACM.

[6] Arun C. Surendran. Automatic discovery of personal topics to organize email. 2005.

[7] Mingliang Zhu, Weiming Hu, and Ou Wu. Topic detection and tracking for threaded discussion communities. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 77–83. IEEE, 2008.

[8] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM.

[9] J. Beringer and E. Hüllermeier. Online clustering of data streams. Technical report, 2003.

[10] Mario Barcala, Jesus Vilares, Miguel A. Alonso, Jorge Grana, and Manuel Vilares. Tokenization and proper noun recognition for information retrieval. *Database and Expert Systems Applications*, 0:246, 2002.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[12] Martin Porter, Richard Boulton, and Andrew Macfarlane. The english porter2 stemming algorithm. "`http://snowball.tartarus.org/algorithms/english/stemmer.html`", 2002.

[13] Christiane Fellbaum, editor. *WordNet: An electronic lexical database*. MIT Press, Cambridge, MA, 1998.

[14] Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252–259, 2003.

[15] Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, 1994.

[16] Steven Bird, Edward Loper, et al. Natural language toolkit. "`http://www.nltk.org/`", 2001.

[17] Beatrice Santorini. *Part-of-speech tagging guidelines for the Penn Treebank project*, 1991.

[18] Users of lists.ubuntu.com. The ubuntu-users archives. "`https://lists.ubuntu.com/archives/ubuntu-users/`", 2004.