MODELING THE EVOLVING STRUCTURE OF SOCIAL TEXT FOR INFORMATION EXTRACTION AND TOPIC DETECTION

Julien Dubuc

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

April 2011

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Julien Dubuc

Entitled: Modeling the Evolving Structure of Social Text for Information Extraction and Topic Detection

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr. Brigitte Jaumard

_____ Examiner
Dr. René Witte

_____ Examiner
Dr. Yuhong Yan

_____ Supervisor
Dr. Sabine Bergler

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____
Dr. Robin A. L. Drew, Dean

Faculty of Engineering and Computer Science

Abstract

Modeling the Evolving Structure of Social Text for Information Extraction and Topic Detection

Julien Dubuc

The advent of "social media" has enabled millions of people to participate in discussions within communities on a global scale. These conversations take place in a myriad of venues, on or off the web, each with its particular approach to implement what we now call "social media" – blogs, bulletin boards, mailing lists. However, while the software powering these communities varies a great deal, and continues to evolve, all of them share a common set of features. When a user initiates a discussion, the message is not addressed to a specific person, but broadcast to any interested reader; such a message can generate replies from other users, and these replies can then generate their own, forming a network of connections between messages. There is a need for a system that can make connections between related pieces of social text, to group information into coherent units. Making use of the structure of the social text helps to determine which elements of the text to consider for a given topic. To do this, a system needs to consider the different *contexts* in which it can be understood. A post, text transmitted by a single author at the same point in time, may have a different topic than the whole thread, which is comprised of all the posts in the discussion following an initial post. Different passages in a post could also have separate topics. Therefore, it is useful to annotate the text with information about its social structure explicitly for use in automatic search and text mining.

Acknowledgments

The author would like to thank his supervisor, Dr. Sabine Bergler. Without her constant encouragement, her unflappable commitment to excellence in research, and her steadfast, enduring patience, this thesis would not have been completed successfully.

Thanks, boss!

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Social Media as a Source of Information

The advent of "social media" has enabled millions of people to participate in discussions within communities on a global scale. These conversations take place in a myriad of venues, on or off the web, each with its particular approach to implement what we now call "social media" - blogs, bulletin boards, mailing lists. However, while the software powering these communities varies a great deal, and continues to evolve, all of them share a common set of features. When a user initiates a discussion, the message is not addressed to a specific person, but broadcast to any interested reader; such a message can generate replies from other users, and these replies can then generate their own, forming a network of connections between messages.

However, when a new user first finds a discussion, it is likely to come from search engines, which only see the information in the text as it is represented for the web. While a web document contains structure in the form of HTML tags, this structure is meant for human readers to interpret, not for machine processing of the information. The rapid evolution of the internet, the web in particular, means there is no consistent, standard way to express a "message" or "discussion thread". This structure implied by the formatting is not readily available for automatic processing. Thus, a search

engine generally cannot discern the discussion's structure in the web formatting. It can only relate a user's query to the social text content at the level which is represented on a given web page. It is not uncommon for a search engine's usefulness with social text to be hampered by this problem.

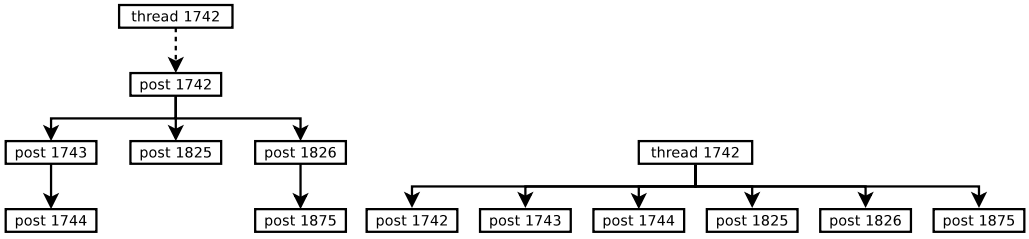
For example, the popular bulletin-board software `phpBB`[1] has a search function which only returns *threads* related to a user's query. A search query featuring multiple keywords can return threads where no single post bears every keyword. If the thread has generated hundreds of replies - which is common for popular discussions - the user will need to scan through dozens of pages of posts to find occurrences of the search terms. If a user has a specific information need - for example, the correct way to enable video output to a projector for an important presentation - tracking the different threads with promising results could lead to learning more unrelated information instead, such as how many users prefer a specific brand of computers, or one user's story of how technical problems caused a difficult time with family.

It is important to clearly define what is meant by "social text". The expression "social media" has now entered common parlance, referring to the multitude of services enabling on-line communities. This encompasses everything from traditional bulletin-boards and mailing lists, weblogs with comment pages, and dedicated picture and video-sharing sites where users discuss each other's work. This thesis is only concerned with the text parts of the content, the *social text*. As such, it is also more focused on the kind of communities where interaction is centered around text, as opposed to dedicated photo or video-sharing services.

1.2 Example of Difficulties in Finding Information

As an example, consider the discussion thread shown in Figure 2. This thread was taken from a discussion website about Machine Learning called MetaOptimize[2], which is described in further detail in Section 5.1. In order to understand how the discussion took place, the structure of the discussion is shown in Figure 1. Someone reading post 1744 by itself would find no information of

[1] http://www.phpbb.com/
[2] http://metaoptimize.com/qa/



(a) reply structure (b) containment structure

Figure 1: Structure of thread 1742 from metaoptimize.com

Post 1742
Hello all,
I'm looking for a decent implementation of deep belief networks in Java. I've found jaRBM, but I think that it is just a RBM implementation and it seems not being updated anymore (Although I haven't fully checked it). I know there are stuffs written in python, but I need a Java implementation.
Post 1743
I know that the Mahout project is planning to add an implementation of stacked RBMs. Apparently the work has started in this gihub branch but I haven't tried it yet.
Post 1744
hmm great news ogriel. I'm going to review the code and try it if it is usable or not.
Post 1825
If you have RBMs or stacked RBMs, it is trivial to get DBNs from that, as far as sampling from the DBN is concerned. If you want to fine-tune the model for something like supervised prediction or classification, you also need neural net code, but that already exist in Java.
Post 1826
If you have a c/c++ implementation, you can make a wrapper to it and call it from java.
Post 1875
No, unfortunately not, if I had a C/C++ implementation, I'm aware that it is possible to call these libs or APIs with JNI.

Figure 2: Contents of thread 1742 from metaoptimize.com

value. When considering it in the context of Post 1743, however it is more meaningful. Post 1743 itself has some information, but to understand it better, it needs to be taken in the context of Post 1742. This makes sense, since post 1744 was written as a reply to post 1743, which itself was a reply to post 1742. Similarly, post 1875, taken by itself, has some meaning, but would have no reason to appear in this discussion if it were not replying to post 1826, which itself needs to be understood in the context of post 1742. This is to show that in a discussion, a piece of the conversation can only be fully understood when taken in the *context* of the rest of the text in the discussion.

This thread also shows an example of how quickly an on-line discussion diverges into different

topics. The initial question in post 1742 was asking about whether a specific piece of software existed that implemented "Deep Belief Networks" in the Java language which was still actively maintained. The sub-discussion found in posts 1826 and 1875 has diverged into the relationship between software written in the C++ language and software written in Java. These were ostensibly contributed to the discussion in good faith, intended to help the author of the original question. That diversion into a different subject did not, in the end, provide useful information, as the original author points out in post 1875. A user interested in the original question will most likely not find useful information in these two posts.

1.3 The Need for fine-grained information

This problem of diverging topics ties into the area of Topic Detection and Tracking (TDT). Here, a "topic" means the common subject, or related content, in a group of related pieces of text. While the problem was originally conceived because of the need to automatically extract information from newswires, it becomes even more challenging with the democratization of mass media brought about because of social media. In online discussion, it is not safe to assume that the topic of a discussion is fixed when it starts; the discussion often branches out into related subjects, unrelated asides, personal opinions or even personal attacks. Thus, the *initial* subject of a discussion is not a guarantee that subsequent posts will follow the same subjects. In fact, as the discussion evolves, individual posts often contain passages dealing with the many subjects that emerge in the conversation. A reader interested in only one of these topics will need to sift through the unrelated parts of the discussion to find those that are of interest.

There is a need for a system that can make connections between related pieces of social text, to group information into coherent units. Making use of the structure of the social text helps to determine which elements of the text to consider for a given topic; for example, knowing a given author is often very vocal about certain topics, or discounting part of a thread spurred by personal attacks. To do this, a system needs to consider the text at different levels of granularity, to show

the different contexts in which it can be understood. A *post*, text transmitted by a single author at the same point in time, may have a different topic than the whole *thread*, which is comprised of all the posts in the discussion following an initial post. Different *passages* in a post could also have separate topics. Therefore, it is useful to annotate the text with information about its social structure explicitly for use in automatic search and text mining.

This thesis proposes an approach to information retrieval in social text which makes extensive use of its hierarchical structure. In particular, Section 3.2 shows how an existing vector-space indexing strategy can be applied recursively to the hierarchical structure, yielding an index that represents the information in the text at multiple levels of granularity. This approach allows an information-retrieval system to use the context provided by the social structure to retrieve a piece of text at the appropriate level of scope, be it a whole thread, a single post or a particular passage. While this approach to indexing produces a more complex index, it is conversely less complex to update this index with new content, making this approach better-suited to the constantly-evolving contents of social media. The intent of this approach is to provide the user of a search engine not only with the text containing information of interest, but also with the appropriate scope in which to consider it. For example, a user may be interested in information found in a single passage of a long and otherwise unrelated post. A system retrieving post would be likely to overlook this information, as it is not the main object of this post. By considering the text at different levels, the hierarchical indexing approach allows a system to retrieve both small, focused passages, and long, varied discussions, for a single query. This can be implemented as part of the software which powers social media on the web, highlighting to the user where a discussion's topic changes, to help the reader focus on the topics of interest.

Section 3.4 introduces a clustering algorithm which also uses the hierarchical structure. The goal of this clustering is to group pieces of text together when they discuss the same topics. By attempting to relate pieces of text which are related by a shared context, it uses the social structure to make informed decisions about which pieces of text are likely to be related. This significantly

reduces the amount of work needed to cluster social text when compared to approaches that are not informed by structure. These "topic clusters" are intended to assist in implementing recommender systems, which can direct readers to pieces of text that are related to what they are currently reading. They can also be used to highlight, within a discussion thread, places where the topic of discussion changes, helping readers to avoid getting distracted by unrelated, tangential discussion. Clustering this way is done in linear time, which is a great improvement over the baseline, bottom-up agglomerative approach to clustering, both in terms of the time to cluster and the memory needed.

These algorithms have been implemented in a proof-of-concept system, which has been dubbed TopicAl. This system is able to parse social text from a variety of sources into a single, unified model of threaded discussion. This data can then be indexed hierarchically, and clustered to group pieces of text together when they share the same topic. The element in the topic cluster represents the text at the appropriate level of score where the topic is encountered. In order to get a better understanding of how well this approach performs on an entire discussion site, it needs to be evaluated in a more objective manner. This requires comparing clusters generated by the system to a classification which is known to be accurate. Section 5.1 shows how social websites which feature community-ranking of posts and threads can be used to implement such an evaluation. Section 5.4 shows some case studies that highlight how well the system is able to separate the different topics in a discussion. The results of the quantitative measurements of the accuracy of the system are shown in Section 5.5.

Chapter 2

Previous Work

2.1 Topic Detection and Topic Modeling

The TDT initiative from DARPA [All02] aimed to detect the topics being discussed in news stories. While newswire articles have a somewhat predictable structure, they do not possess the rich nested structure of replies and quoting found in social text. In fact, one of the TDT challenge tasks was to detect the boundaries between stories presented in a continuous stream of text, based solely on content. This focus on content is in contrast to the focus on structure described here.

[ZZW07] undertake the TDT task of New Event Detection, that is finding the earliest mention of a new topic in a sequence of documents as they arrive. A modified, incremental version of TF-IDF indexing (see Section 3.2.1) is used, which can be updated dynamically to reflect currently mentioned topics by including the time of mention as a factor of the indexing. The index is stored not as a matrix, but as a hierarchy of clusters, where related documents are grouped by content as well as the time they appeared. These clusters contain documents as well as sub-clusters, which contain increasingly smaller, more homogeneous groups of documents related to finer-grained topics. While this hierarchy is not based on structure, its architecture based on incremental indexing is well-suited to sources of text that are constantly updated.

[CM09] attempt to determine the topics mentioned in a document by relating the document's content to Wikipedia. They do this by finding the Wikipedia pages related to terms mentioned in the text, and highlight which of these Wikipedia pages are most important. This results in many pages, of which only a fraction actually represent high-level concepts that could be called "topics". The pages are treated as nodes in a graph, where links between pages become edges between nodes. The most relevant concepts are determined by a graph centrality measure, which is an adaptation of the PageRank algorithm of [BP98] that is biased towards pages linking to concepts represented in the document. This is only suitable to identifying topics related to high-level concepts which possess a Wikipedia entry. The breadth of subject matter in online discussion cannot be expected to be completely covered in an online encyclopedia, even one as extensive as Wikipedia.

[SMG+07] focus on the TDT task of Topic Segmentation, meaning to determine the boundaries in text where the topic changes. Most prior work in this area considered the text as a single continuous stream, analogous to a news ticker, as in [Hea93]. In contrast, this paper looks at small segments of many documents at once, using a measure of Mutual Information to not only compare consecutive segments, but to align similar segments of different documents. This means the language model of a topic is shared between documents, making it more robust.

[GZZ+10] present an approach to topic modeling which is based on the idea that documents are not independent from each other, but are interlinked. This is done using a what they refer to as a Bernoulli Process Topic (BPT), which is an adaptation of Latent Dirichlet Analysis where two topic models are combined: one based on the document contents, and another based on the contents of the linked documents. By integrating the context provided by the structure linking documents, this approaches yields an improvement over baseline probabilistic approaches.

2.2 Social Media Mining

The Blog Track of the Text Retrieval Conference (TREC)[MOS09] has evolved to try to identify topics in social media. The topics here are predetermined categories, discussing noteworthy events.

There is little interest in evolving topics. The blog posts making up the collection are taken at one point in time, and do not show the further evolution of their comment threads.

The community-detection system described in [BdR07] aims to find a set of weblogs related to a central, source weblog. The approach is based both on the topology of the link graph between these blogs, and the text content of the blogs. The system focuses on the relationships between authors, without trying to link individual pieces of text.

The event-detection task outlined in [ZM07] means to detect significant events by finding clusters of related texts in an e-mail archive. In addition to the message contents, the system makes use of the proximity between messages in two dimensions: the time between messages, and the distance between messages in the graph linking messages to their replies. The results show that using this supplemental information helps group related messages together.

In [NDW07], the authors try to determine not only the topic of a web page, but how authoritative the page is on a topic. This is done by finding the number of links into the page from outside sources. Since a web page may deal with more than a single topic, the approach is to find *where* in the page the links point to.

The Semantically-Interlinked Online Communities Project[1] (SIOC) has produced a semantic web ontology for representing both the content and structure of social text [BBFD08]. It defines entities such as `Forum`, `Post`, and `UserAccount`, providing a consistent vocabulary for the metadata associated with social text. It also supports specifying the topic(s) of a given Post as a combination of Category and Tag instances. Here, the content container with the finest level of granularity is the Post. This means that topics are specified for an entire post; there is currently no way to specify which parts of the text deal with specific topics. In this model, the Thread concept is merely a container for Post instances, it cannot bear topic information itself. This makes it difficult to distinguish the topics associated with the entire discussion thread, as opposed to those mentioned in the original post that do not represent how the rest of the discussion subsequently evolved. Recently, the paragraph level has emerged as useful for indexing and information retrieval, as demonstrated

[1] http://sioc-project.org/

by the good performance of paragraph level IR, for instance in the "other" questions in the TREC Question Answering track [DLK06].

[HCL07] eschews traditional IR methods, opting instead to use an approach based on signal-processing. Each topic is treated as a frequency band, and the combination of these topic signals makes up a model for the text contents. This approach allows transforms to be applied to the "signal", to model it either from the time-domain or the topic-frequency-domain.

[CBBK09] discuss a way to distinguish multiple topics in a single document. Observing that related topics are brought up in a similar order, with small variations, this expected ordering can be modeled. After training on example documents, a model can use the expected order to make an informed choice in determining the topic of a new passage. This expectation helps to focus on more likely topic choices, reducing ambiguity.

[WO09] examine the problem of interleaved conversations, where more than a single discussion is taking place over a shared channel. They cite the example of IRC chatrooms, though this can take place in most forms of social media. Three kinds of context are used to discern the interleaved discussion: the authorship of posts, their distribution in time, and explicit mentions of names of participants. These contexts are mapped to dimensions in a vector-space, and the cosine-similarity between these vectors is used to cluster messages into more coherent conversations.

[YCS09] aim to model political blogs to be able to predict the response they generate in comment threads. This is done by making a language model of the language in the post, as well as the language of the comments, as related but separate models. Two variations on Latent Dirichlet Analysis are used: LinkLDA predicts which users are likely to respond, while CommentLDA uses these predictions to predict the content of replies. The topic model defines a topic as a combination of the distribution of words in the blog post, the distribution of users participating in comments, and the distribution of words in commments. This separation aims to model how political blogs often elicit comments from users with different opinions than the original author.

[GLMF09] attempt to make a predictive model of the behavior of bloggers. This model is informed

by two different factors: the distribution of blog postings through time, and the relationships linking blogs together in the blogosphere, providing paths for the propagation of information. The temporal and topological factors are represented in a number of patterns, such as the size of the conversation stemming from a posting, or the frequency in which a post gets cited by other bloggers, which have been observed to follow power-law distributions. A random walk through models of these factors is used to make predictions on when a blogger will write, what the blogger will write about, and which posts in other blogs are likely to be cited.

[WBC+10] focus on the problem of modeling online discussions which involve many topics at once. This is done by observing and modeling user behaviour. Two factors are considered: the topic of discussions where the user participates, and the topics where a user's friends participate. In this case, friends are determined by measuring the flow of information between pairs of users, by tracking the content and frequency of replies. This model is shown to accurately predict user participation in discussions based on their topic.

[WCB10] attempt to model the interactions inside social networks as they evolve through time. Unlike most social network models, which assume the configuration of the network is fixed, this approach, based on Hidden-Markov Random Fields, models social networks where the weights of edges evolve through time.

From these prior papers, it becomes apparent that there is growing interest in modeling the structure of social text, but no definitive model to do so has emerged. There is also interest in how meaning is shared between related pieces of text, though many approaches to deal with this presuppose some prior knowledge of the kind of topics being discussed.

Chapter 3

A Model of Varying Granularity for Text Indexing and Clustering

This chapter presents the main contributions of this thesis:

1. A model of the structure of social text as a hierarchy of containers. This simple model is general enough that it can be extracted from any kind of threaded discussion.

2. A method for building an index of the contents of social text which uses its hierarchical structure. This method uses a well-known indexing formula, but redefines the notions of "collection" and "document", by applying the formula recursively to the container hierarchy to index the text at different levels of scope.

3. A method for clustering the content of social text which uses the hierarchical index to make informed decisions about which pieces of text are likely to be related.

These contributions are built on the initial work detailed in [DB10].

3.1 Modeling Social Text

As noted earlier, a distinguishing feature of social text is its structure. While the terminology can vary between the different services and the different software used to implement these services, the basic mechanisms are similar enough that they give rise to a structure expressed with a consistent set of metaphors. An author initiates a discussion thread by posting a message with a new subject line, which is by default a solicitation for replies. Others can then send a response to this message, and eventually to other replies. This nested reply structure forms a tree rooted at the initial post, and the thread is the set of posts taken together. Having this structure can help determine which pieces of the thread fit the thread's subject line, and which parts discuss a different topic.

In the model proposed here, the structure of social text takes the form of a tree, where nodes at different levels represent different levels of scope when considering the text. The nodes at higher levels act as containers for the content of their descendants. A `Source` node acts as the root of the tree for all text originating from a given blog, forum or mailing list. One level below, its descendants are the `Thread` nodes. The level below threads contains the `Post` nodes. Finally, the level under posts contains `Passage` nodes, which are the leaf nodes of the tree. The source acts as a container for its descendant thread nodes, which themselves are containers for the post nodes which are their descendants. We contend that this "containment" relationship is useful in determining the scope of topics. Assigning a topic to a post node means only the content in the post node is part of this topic, while assigning a topic to a thread node means the content of the thread as a whole is part of that topic. A thread can in fact contain posts within different topic groups; the thread's own topic group is a reflection of its *overall* topic content.[1]

This hierarchy is useful for topic detection, in modeling the way that topics change and evolve in a discussion. A thread often grows to encompass many topics, so it makes sense to try to model both the thread and its posts as having a topic. However, in a complex discussion, even a post may be long enough to deal with more than a single topic. It would be useful to look at a more fine-grained

[1] This hierarchy is only one of many possible combinations of structural elements. Other possibilities not exemplified in this model include the nested reply and quote hierarchies, the author of a message or passage, and the position along the timeline.

level than the post, to determine *where* in the post text the topic is mentioned. We assign topics to paragraphs, posts, and threads, to be able to capture when more than one topic is mentioned in a post, and also assign an overall topic to posts and threads.

In this model, a *passage* is a region of text with a single topic. Since the model is based on the idea of using document structure to delimit text, it makes sense to look for an element of structure inside the post that separates its text into regions that bear possibly different topics. Fortunately, this is precisely how writers use paragraph boundaries [Joh02]. A paragraph in the text of the post becomes a passage node contained by the post node.

All of this information can be obtained from a web-based representation of the social text. While it is easier to deal with more direct access to data, such as monthly archives of a mailing list, or direct access to a blog's database, the structure can be recuperated by anyone from the publicly-available web version. A parser needs to be written which understands a particular format's conventions, such as quoted lines being preceded by > characters in mailing-list archives, while web boards represent quoting as embedding a nested frame in the message frame.

3.2 Indexing Social Text along its Hierarchical Structure

Retrieving information from text can be a costly operation. While text data is dense, and file-based representations are small compared to other types of information, collections of text documents can easily hold millions or even billions of documents. It would be very inefficient, if not completely unrealistic, to store each document in a file and evaluate its text directly each time a user inputs a query to a search engine. This is why search engines, and other information-retrieval systems, build an *index* of the collection, which only stores the information used to retrieve documents from the collection.

One naive way to index documents would be to simply count the number of times each term is repeated in a document. The number of occurrences of a term in a document could be taken as a measure of the importance that term has to the content of the document. This has the undesirable

side-effect that it biases the retrieval towards longer documents, which are more likely to have a larger number of occurrences of any term. One simple way to deal with this is to scale the number of repetitions of a term to the total number of terms in a document, and use this *term frequency* as a measure of the importance of a term for a document.

However, this is not the only challenge in making a useful index. Another challenge is that some terms are more important than others. For example, consider a search query for "Schrödinger's cat". When considering the bulk of text written in English, one can safely assume that the term "cat" is relatively frequent – at least compared to the term "Schrödinger". This can be taken to mean that the presence of term "Schrödinger" is more distinctive than a mention of the word "cat". Thus, from an Information Retrieval perspective, the presence of a rare term is more indicative of the text's content than the presence of a common term. Most common indexing strategies benefit from taking this factor into consideration.

3.2.1 TF-IDF

There is a well-known approach to implementing an index that addresses all of the concerns previously described, which is called "TF-IDF" [SJ72]. The "TF" part of the name stands for "term frequency", while "IDF" refers to "inverse document frequency". The document frequency of a term refers to the proportion of documents in a collection which bear that specific term. By using the inverse of the document frequency, the IDF score is high for rare terms, and low for common terms. The TF-IDF score for a term in a document combines these two factors by taking their product.

For every term t_i in every document $d_j \in D$,

$$\mathrm{tc}_{i,j} = |\{t_i \in d_j\}| \tag{1}$$

$$\mathrm{tf}_{i,j} = \frac{\mathrm{tc}_{i,j}}{\sum_k \mathrm{tc}_{k,j}} \tag{2}$$

$$\mathrm{idf}_i = \log \frac{|D|}{|\{d \in D : t_i \in d\}|} \tag{3}$$

$$\text{tf-idf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i \tag{4}$$

Here, $\mathrm{tc}_{i,j}$ is the number of times term i appears in document j, while $\mathrm{tf}_{i,j}$ is the term *frequency*,

the proportion of the terms in document j that are instances of term i. The *document frequency* df_i is the proportion of documents in the collection that contain at least one instance of term i, while its inverse is called idf_i. The score $\text{tf-idf}_{i,j}$ is the combination of these two factors, which represents the "weight" of term i, how much of the information found in document j is specified by this term.

When a term is very common in a very large collection, the inverse document frequency becomes very low, so much so that the number becomes difficult for computers to represent accurately. When multiplying the IDF score with the TF score, which is also very small, the result is so small that floating-point representations for such small decimal numbers can introduce significant rounding errors. To alleviate this, the IDF factor is actually the logarithm of the inverse of the document frequency. Using the logarithm maintains the relative ordering of IDF scores for every term, so it does not affect the quality of the scoring.

3.2.2 Hierarchical Indexing

A typical indexing algorithm only treats data as a *collection* of *documents*, where every document is treated equally. This means the algorithm makes no use of the structure that ties different pieces of text together, only the text's contents. While this is suitable for a variety of applications, it becomes problematic when trying to index a collection where it is not clear what constitutes a "document". The question arises when dealing with social media: is each post a document? Or each thread? Or even each webpage where some of the social text is archived? Any of these answers could be the appropriate choice, depending on the evolution of the discussion and the information needed by the reader. Section 3.2.3 will detail a way that the TF-IDF indexing algorithm can be adapted for the hierarchical representation of social text presented in Section 3.1.

3.2.3 Hierarchical Indexing Procedure

The first step in making the index is to obtain the term counts for each node in the graph; the text itself is indexed at the passage level. This is done by traversing the graph bottom-up, starting from

the top-level Source node. When the traversal reaches a leaf node, i.e. a Passage node, it determines the terms in the passage text. This requires some pre-processing of the text:

1. Tokenization: The text is split into a sequence of *tokens*, which represent the words of the text, without whitespace or punctuation.

2. Stemming: The remaining word tokens are reduced to base-forms, or *stems*, using the PorterStemmer [Por80] algorithm. This means a verb's different tenses become instances of a single term. Likewise, the singular and plural forms of nouns are reduced to a single term.

3. Stop-word removal: Closed-class words, such as conjunctions, prepositions and articles, are removed from the token sequence, along with very common nouns and verbs which do not carry information specific to the text[2].

These stems become the terms for this passage node, and are stored in a vector together with the count of unique instances of each term in the text.

Once passage nodes have term count vectors, term counts of container nodes at a higher level are defined as the sum of their immediate children's term count vectors. Since the non-leaf nodes are containers for nodes further down in the hierarchy, their term count vectors are defined recursively as the sum of their immediate children's term count vectors. The procedure for computing these recursive term counts is detailed in Algorithm 1.

After the first step of indexing, each node has a vector storing the count of terms in the text it contains. This is all the data from the text that is needed to create every index in the hierarchy. Each node's term count vector is then scaled to the number of terms, to generate a term frequency vector. Each non-leaf node uses its children term counts to generate a document frequency vector, which stores, for each term, the proportion of children which bear that term. Finally, a node's term frequency vector and its parent's document frequency vector are combined to form the node's TF-IDF term vector, using the standard formula [MS03]. This index vector does not consider the

[2]The stop-word list that was used here was taken from http://www.lextek.com/manuals/onix/stopwords1.html

Algorithm 1 HierarchicalTermCount

```
Require: hierarchical representation of social text
  node.termcounts ← ∅
  nodeterms ← 0
  for all child ∈ node.children do
    childterms ← 0
    for all term ∈ child.termcounts do
      if node.termcounts[term] = 0 then
        node.termcounts[term] ← child.termcounts[term]
      else
        node.termcounts[term] ← node.termcounts[term] + child.termcounts[term]
      end if
      childterms ← childterms + child.termcounts[term]
    end for
    for all term ∈ child.termcounts do
      child.termfreqs[term] ← child.termcounts[term] ÷ childterms
    end for
    nodeterms ← nodeterms + childterms
  end for
```

node's text compared to all other nodes, but only to its siblings, which share the same parent and thus the same context.

Algorithm 2 HierarchicalTFIDF

```
Require: HierarchicalTermCount has already been executed
  node.doccount ← ∅
  numchildren ← |node.children|
  for all child ∈ node.children do
    for all terms ∈ child.termcounts do
      if term ∈ node.doccount then
        node.doccount[term] ← node.doccount[term] +1
      else
        node.doccount[term] ← 1
      end if
    end for
  end for
  for all term ∈ node.doccount do
    idf[term] ← log(numchildren ÷ node.doccount[term])
    tf-idf[term] ← tf[term] × idf[term]
  end for
```

3.2.4 Other Approaches to Indexing

TF-IDF is mature, robust, well-understood, and simple to implement. However, there are many other ways to build an index, which have some features that TF-IDF does not. One such method is

Latent Semantic Analysis, or LSA[3] [DDF+90], which can be viewed as an extension to TF-IDF. The goal of LSA is to give weight to terms in a document's term vector, even when the term is not present in that document, as a term with similar "meaning" *does* occur. This is because LSA exploits the co-occurrence of related keywords over the entire collection. For instance, a query on an LSA index for the word "car" may return documents which never present this term, but mention the word "automobile", as these two terms appear in similar contexts, surrounded by a similar distribution of terms. This statistically-derived relationship is what is meant by "Latent Semantics".

While LSA has obvious advantages, there are some reasons why LSA is not well-suited to the hierarchical indexing presented here. In order to exploit the latent relationships between terms, the algorithm needs a large collection of documents, with many instances of co-occuring terms in similar term distributions. This redundancy is what makes the latent relationships robust. However, in the indexing method presented here, the collections associated to each node in the hierarchy are relatively small. Flat collections often have millions of documents, while threads typically get dozens of posts. Even on the most frenetic bulletin boards and mailing lists, discussions rarely reach thousands of posts. With an amount of redundancy that is smaller by orders of magnitude the relationships exploited by LSA are proportionately less robust.

Another concern is the size of the index itself, and the associated performance costs. Most documents have very few terms when compared to the entire set of terms used in all text. The small number of terms that occur in most documents are stop-words; articles, prepositions, conjunctions, and other "closed-class" words[4] that are excluded from the indexing process, as their occurrences carry no information specific to the content of the text. If considering every index vector in every index of every node of the hierarchy as a *single* term space, each vector is extremely sparse. On typical flat indexes, these vectors are not stored as simple arrays of decimal numbers, but in a sparse fashion. Doing otherwise would consume wast amounts of memory to store millions of occurrences of the number "0.0"[5]. Since LSA gives weight to keywords that do not occur in text, the indexes

[3] this is sometimes referred to as Latent Semantic Indexing, or LSI
[4] as opposed to "open-class" words, such as nouns, verbs, adjectives and adverbs
[5] details on this sparse implementation are found in Section 4.3.

become much less sparse. The increased cost in memory made the difference between an index that can fit in the memory of an average laptop computer, and one which cannot fit in the vast memory of a specialized computation server.

3.3 Querying

The indexing procedure described in Section 3.2 can be used to implement an information retrieval system. A user can input a query, which is transformed into a weighed term vector the same way that a passage text is used to create a passage term vector. The passage term vector can then be compared to the term vectors of all the nodes in the index, using a well-known term-space distance measure, such as the cosine-similarity [MS03] measure. The system can then show a list of nodes, ordered by highest similarity first. Unlike a traditional information retrieval system, which uses a "flat" index, the system can present as result not only a piece of text, but the most appropriate scope in which to consider it. This follows the assertion made in Chapter 1 that the information contained in a piece of text is dependent on the context in which it is considered. By indexing a piece of text at varying levels of scope, the system is able to relate a query to the most appropriate level of context for it. In fact, once the hierarchical index is built, querying and retrieval function in the same way as in a typical search engine.

3.4 Clustering Social Text with the Hierarchical Index

Besides simple retrieval, another application that is enabled by the hierarchical index is clustering. Instead of matching pieces of text to a user query, the goal of clustering is to group pieces of text together when their contents are related. This can be accomplished by comparing the term vectors of every pair of pieces of text, using the same kind of vector-space distance measures that are used for querying. A simple way to achieve this is to create a cluster for every piece of text which bears a term vector, compute some measure of the distance between these clusters, and merge the two

clusters that are most similar. When two clusters are merged, the term vector for the resulting cluster is a combination of the term vectors for the two merged clusters. This means the distance between the newly-merged cluster and all the other remaining clusters needs to be computed before further merging can be done, as which pair of clusters is most related changes every time a new cluster is created with a new term vector.

Repeating this procedure gradually reduces the number of clusters; if left unchecked, the process would eventually produce a single cluster containing everything. This is why the process is typically stopped once the distance between the two most-related clusters is below a fixed threshold value. When using a distance measure such as cosine-similarity (see Section 3.4.1), the distance values are decimal numbers from 0.0, meaning "completely unrelated" to 1.0, meaning "completely related". The threshold would then be a decimal value situated somewhere between those two extremes. This procedure is called "bottom-up agglomerative clustering" [MS03]; "bottom-up" because a large number of small clusters progressively become a small number of large clusters, and "agglomerative" because every element in the cluster has equal status, as opposed to "hierarchical" clustering, where the order in which clusters are merged produces different structures, with different meanings.

It should be noted that while the method described previously is conceptually simple, it is also quite expensive. Also, it makes no use of the hierarchical structure of social text data. By treating every node equally, the algorithm assumes that any post has an equal chance of being related to any other post, or in fact any other thread. It seems obvious that there is actually a higher chance that a post is related to the thread where it appears, or to posts which also appear in the same thread. More generally speaking, there is a higher chance that two nodes related by structure are also related by content. The structural information would then be useful for clustering, but the clustering procedure would need to be adapted to use this information This section presents a clustering procedure that makes extensive use of this structural information.

3.4.1 Measuring Similarity

Items to be clustered can be considered as points on a hyper-plane. Computing the similarity between two vectors can be done by simply using the Euclidean distance. This is difficult to apply to vectors in term space. Depending on the indexing algorithm used, the magnitude of a vector may not be as relevant as the "direction" in space. Another way to measure the similarity between vectors in n-dimensional space is to use the angle between two vectors. The angle does not depend on magnitude, only on the relative value of each weight dimension in relation to the other dimensions. The cosine of the angle between any two vectors will yield 1 when the angle is 0. This means that if two vectors have the same direction, the cosine will be 1, regardless of the magnitude of the vectors. Since in term space, all non-zero dimensions will have a positive weight, two vectors with no term in common will be orthogonal, and the cosine of the angle between them will be 0. Any partially-related vectors will have a cosine of their angle between 0.0 and 1.0, which is convenient. This measure, called the "cosine similarity" measure, is often used to compare term vectors [MS03]; it is detailed in Equation 7.

For term vectors $\vec{a}, \vec{b}$:

$$|\vec{a}| = \sqrt{\sum_{i=1}^{n} a_i^2} \tag{5}$$

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i \tag{6}$$

$$\textsc{CosineSimilarity}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} \tag{7}$$

3.4.2 Hierarchical Clustering in Social Text

The hierarchical structure can also be used in clustering pieces of text, to determine which elements of the discussion share the same topics. One difficulty of clustering is that since any element cluster can be merged with any other, there is a very large number of pairs of elements that can potentially be merged. With a wide variety of topics, there will be a large number of smaller clusters, so most comparisons will not be successful. However, the social structure can help make informed decisions

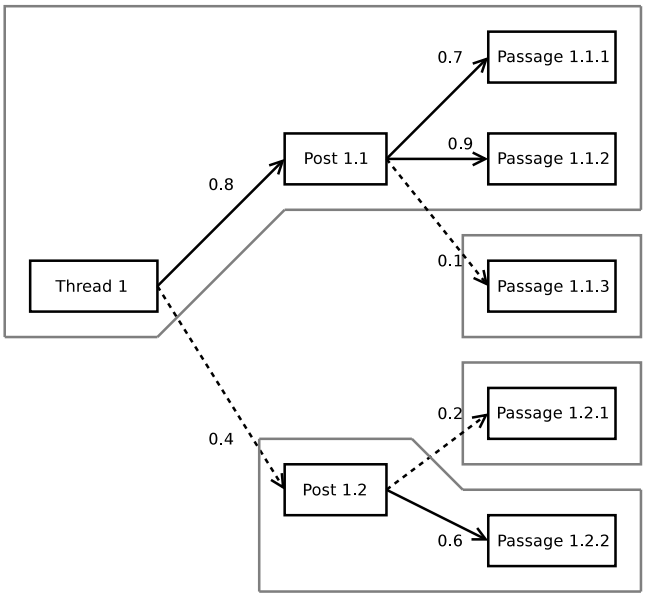


Figure 3: Clustering social text with threshold 0.5

about which elements to try to merge. For instance, since the topic of a thread can change as posts get added, it makes sense to compare a post with its thread. Similarly, since a post can contain multiple subjects, it is helpful to compare a post with its passages. These comparisons are more meaningful than simply comparing any two elements at random, as these nodes are related by the text's social structure.

The first step in clustering is to create one cluster for each node in the hierarchy, with the same TF-IDF vector as its node. Subsequently, the tree is traversed depth-first, and each node's cluster is compared to its parent's using the cosine distance between their TF-IDF vectors. If the similarity is above a fixed threshold, the clusters are merged, as is shown in Figure 3. The resulting cluster's term vector is the sum of the merged clusters' term vectors.

When clustering along structural links between text nodes, it is not possible to group together two related pieces of text that do not have structural links, such as two posts on the same subject that appear in unrelated threads. In order to make these clusters more useful and coherent, a second pass of clustering is necessary. This second pass performs a more traditional bottom-up agglomerative clustering on the clusters obtained from hierarchical clustering. While this method

is still $O(n^2)$ for n clusters, it takes place after the first pass, so a great number of clusters have already been merged, reducing the complexity significantly. Since the clusters being compared are then not related by the text's structure, it is not safe to assume that they are related by content, as in the first pass. To account for this, a higher threshold value can be used, to make the criteria for merging more strict than in the first pass.

Algorithm 3 HierarchicalCluster

```
Require: 0.0 ≤ threshold ≤ 1.0
Require: hierarchical index built
  node.cluster ← {node}
  for all child ∈ node.children do
    child.cluster ← HierarchicaCluster(child, threshold) {first, cluster children recursively}
    similarity ← CosineSimilarity(node.cluster, child.cluster)
    if similarity ≥ threshold then
      node.cluster.termvector ← node.cluster.termvector + child.cluster.termvector
      node.cluster ← node.cluster ⋃ child.cluster
      child.cluster ← ∅
    end if
  end for
  return node.cluster
```

3.4.3 Consequences of Hierarchical Clustering

While it uses a standard, well-understood formula, the hierarchical indexing and clustering has interesting consequences.

In a more traditional, flat index, the similar term frequencies would both be combined with the same document frequency vector to generate similar TF-IDF vectors. Consider, for example, a post which has very similar term frequencies to the thread which contains it. This means that when clustering, the post and its thread would be combined into the same cluster. This is not necessarily the case with this hierarchical indexing.

The high-frequency terms for the thread likely occur in most posts in the thread. This means the document frequency for these terms will also be high for the thread as a whole. Since the thread acts as a collection for the posts, this high document-frequency will weigh down the TF-IDF scores for these terms in the posts bearing them. This lower score is a useful feature, as it indicates the

presence of the term in the post is not a *differentiating* factor for the post contents when considered in the context of the whole thread. The high score for a term in the thread compared to the low score for the same term in the post mean the term is topical for the entire thread, not just the single post. In effect, the thread's topic will subsume the post's content. The second phase of clustering will then make use of this information to link the thread, not the post, with related text content from other threads, posts, and passages.

Consider the case where a thread only contains a single post, having not yet generated replies. The post and thread both have the same term counts and term frequencies. However, these nodes are indexed as part of two collections. The post is indexed together with other posts in the thread, but there are no other posts; the document frequency will be 1 for every term in the post. The thread is indexed compared to every other thread from the same source. This is probably a high number of threads, implying that the document frequency of the thread's terms will be comparatively low. The low document frequencies will boost the TF-IDF scores for the thread's terms. If the change in document frequencies is not uniform for every term – a likely assumption with many threads bearing different content – the angle between the post's vector and the thread's will be changed. Depending on the threshold value used in clustering, the post can end up in a different topic cluster than the thread.

Conversely, posts with very different term frequencies from their containing threads will also form a separate cluster from their thread. If a few posts in a thread discuss a different subject, they will cluster together in a topic separately from the thread's overall topic. The goal is to link a topic to the level of scope where it is found in the text, be it a single passage, a whole post, or the entire thread.

The hierarchical index is considerably more complex than a flat index, as it contains a large number of indexes – one for each internal node in the hierarchy. These multiple indexes can refer to the same text content, but at different levels of scope. While this means the initial construction of the index is costly, incrementally updating the index with new content is not. Updating the index

with new content is done by adding nodes for new passages, posts, and possibly threads. The index is built for the text in the new passages, and rebuilt for every ancestor up to the root, the Source node. This means that most existing nodes are not affected by an update. Similarly, the new clusters generated for the added content can be compared to the existing, merged clusters for the existing content, without affecting the clustering results.

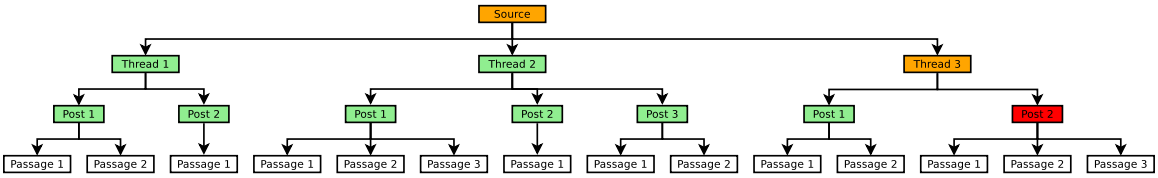


Figure 4: Adding a new post results in an incremental index update

This incrementality was one of the important goals of this approach. Both the hierarchical indexing and clustering procedures presented here were designed to deal with the way social text content is updated continuously. Any tool designed to retrieve information from social text necessitates incremental indexing. In a simple implementation of TF-IDF indexing, as soon as a new document is added to the collection, the document-frequency vector changes for the *entire collection*. This means that, depending on the terms found in the newly-added document, *any* already-indexed document could receive modifications to its term weights. In contrast, when a new post is added to the hierarchical index, most existing term vectors are not affected. Figure 4 shows an example of what happens when the index is updated: a new post, in red, is added to an ongoing discussion thread. The post index needs to be built, while the thread where the post appears, shown in orange, has its index updated, like the source node, which indexes the thread. The green nodes are nodes that contain an index of their descendants, which is not affected by this update, while the white passage nodes do not bear an index, as they have no descendents to index. This structure guarantees that when a new post is added, only 3 nodes need to be indexed anew: the post, its thread, and the source node.

Another area where the hierarchical approach is interesting is in the clustering phase. When

considering a traditional, flat collection, every item is compared to every other item. Each comparison is relatively expensive, as it entails calculating the cross-product of two possibly large term vectors. Even (merely) quadratic complexity can make clustering a large collection impractical. Most elements are not related, so much of this hard work does not bring about successful merging.

The hierarchical method helps alleviate this in some ways. By comparing a parent node to its children, and children of a same parent node, the recursive clustering compares nodes that share a context. Since they occur in a related context, it is reasonable to assume a higher likelihood that clusters will be merged. The second pass, which ultimately compares every cluster left, is still a quadratic operation, but it deals with a smaller number of clusters, since many clusters sharing context have already been merged. The clusters compared at this later stage do not necessarily have the shared context shared by clusters compared in the first pass. Hierarchical indexing and clustering is thus an effective way to address vast, constantly growing collections of social text that need to be dynamically indexed.

Chapter 4

Implementation

The model described in Chapter 3 was implemented as a system called TopicAl, shortened from "Topic Analyzer". This chapter provides a description of this implementation. It would be beyond the scope of this document to describe the software architecture of the implementation in exhaustive detail. However, there are a number of issues that came to light while developing this system that warrant further examination.

4.1 Platform

The TopicAl system was developed in the Python[1] programming language. Python is a modern, dynamic, interpreted, strongly-typed, multi-paradigm language, in that it supports object-oriented development, as well as functional programming and imperative style. Besides the author's familiarity with this language, there are a number of reasons that made Python particularly suitable for this task.

The first is the clarity and conciseness of Python code, which makes development significantly faster than for many other languages. Another was the extensive built-in library, which provides any Python installation with a number of features that are necessary in a system like TopicAl:

[1] http;//www.python.org/

- the `mailbox` package supports parsing archived UNIX mailboxes, which is the format used by many mailing-list software packages.
- the `urllib` package supports retrieving arbitrary files through HTTP.
- the `re` package provides comprehensive support for regular expressions.

In addition to those built-in packages, there is a large ecosystem of third-party packages which provide a number of features crucial to the TopicAl system:

- the `BeautifulSoup`[2] package provides a parser for HTML documents. This parser is robust enough to deal with incomplete or missing tags, which are a common occurrence in web pages, and render stricter XML parsers unsuitable. It is used to extract the text contents and social structure from websites.
- the `NetworkX`[3] package, developed at Los Alamos National Laboratory, provides support for very large graphs, and efficient algorithms for processing them. The hierarchical structure linking different nodes is implemented using NetworkX.
- the `NLTK`[4] package, which stands for "Natural Language Tool Kit", provides a complete suite of tools for doing Natural Language Processing. The algorithms for tokenizing text, and reducing words to base-form terms using the PorterStemmer algorithm, were taken from this package.
- the `NumPy`[5] package, also known as "Numerical Python", provided the numerical matrix implementation used in the bottom-up agglomerative second pass of clustering. It is also used as a baseline for comparison. This is described in Section 4.4.

[2] http://www.crummy.com/software/BeautifulSoup/
[3] http://networkx.lanl.gov
[4] http://www.nltk.org
[5] http://numpy.scipy.org

4.2 Parsing Data Sources

Chapter 3 claimed that the hierarchical model can represent social text data from any kind of community. However, depending on the software used to implement the community, the social text is represented in a variety of ways. This means that parsers need to be written for each format in order for data represented in a given format to be processed by the TopicAl system. Given the variety of formats, parsers for only a few formats have already been implemented:

- the UNIX mailbox format
- the blogger weblog platform
- the StackExchange question-answering platform
- the OSQA question-answering platform

The mailbox format is used by many mailing-list services as the format for storing archives on the web. Supporting it made it possible to treat mailing lists like social websites. The "blogger" platform is a popular software package for weblogs, developed by Google. The StackExchange and OSQA platforms are used to build discussion websites where users can vote on the quality of questions and answers. Since they are curated by the community the text is often of higher quality than unmoderated forums. The scores given by the community establish a metric against which the output of TopicAl can be compared, as detailed in Section 5.1. These parsers show that the model presented in Chapter 3 can indeed cope with the variety of ways social text can be represented.

Most blogs, and many other social platforms, can syndicate their content into feeds using the RSS or ATOM standards, which are XML formats. Supporting these could enable access to a huge variety of text online. The problem with this idea is that feeds syndicate only the original blog post, and not the comments posted in reply to this post. Some blogs and discussion sites also have "comments" feeds, but there is one such feed for each post. This is useful for users wanting to follow a particular discussion, but does not preserve the reply structure or the links between a post in the

post feed and its particular comments feed. This makes syndicated feeds unsuitable for providing discussion context.

An attempt was made to support the WordPress blogging platform, which was ultimately abandoned. WordPress appears to be the most-popular blogging platform currently in existence; supporting it would give access to a huge portion of the social text currently being written. However, the flexibility that has contributed to its success as a platform also makes it difficult to process automatically. The HTML code for a page generated by WordPress depends on the theme file used. Depending on the theme, different names are used for tags, and the same information is found at different levels of nesting. The large variety of WordPress themes means that there would need to be a correspondingly large number of parsers.

4.3 Indexing Implementation

One important challenge in developing the TopicAl system was the representation of the hierarchical index. This is actually composed of a large set of indexes, one for each internal node in the hierarchy. Aside from the root index, which indexes all threads and contains all terms in the collection, these indexes are relatively small. They refer to small numbers of "documents" – the number of posts in a thread, or even the number of passages in a post.

The simplest way to implement this index would be to use arrays, where each array represents the contents of a node as indexed by its parent node. Each position in the array refers to a term in the hierarchy's term space, and the value at this position is the term's weight for this array. If implemented this way, most arrays would be extremely sparse; a given passage can only contain a tiny fraction of the entire hierarchy's term space. Using most of the memory to store repeated instances of the number "0.0" makes such large demands on the memory as to make this approach unworkable.

There are a variety of sparse-vector implementations available for Python, as part of the NumPy package. Unfortunately, using these implementations was not workable. Depending on the approach,

they still consumed too much memory, or were thousands of times slower.

The solution was to use associative maps, which Python refers to as "dictionaries", to store each term and its weight as key-value pairs. Dictionaries are implemented as hash-tables, which are very efficient in the time it takes to store and retrieve values. They also resize themselves automatically, so that they only take as much memory as is required to store the data. The basic Python `dict` class was extended into a class called `VectorSpace`, which provides a number of additional features:

- arithmetic operators, that implement typical vector operations; vector addition and multiplication, scalar multiplication and division, vector cross product and dot product.
- instead of raising an exception, accessing the value for a non-existent key gives the default value of 0.0, so that operations can be done on vectors with different term sets completely transparently.

This makes objects of class `VectorSpace` behave like typical mathematical vectors, where dimensions are named by strings instead of being numbered by integers.

There is a feature of the Python platform that makes this solution more suitable than it would be on other platforms. In many languages, such as Java and C++, using strings as keys would mean that in every vector, each key would be a different string object, even when the contents of the key are the same. In Python, string objects are immutable, and immutable objects are stored only once. Each key is actually a pointer, a reference to an object that exists only once for any given value.

When assigning a string to a variable, if this string already exists somewhere else in the program, the variable will point to this instance instead. Only when using a string for the first time will a new object be created. This means that, for example, a program can have a number of variables with the string value "discombobulation", but all of these variables actually point to a single instance of this string, which is unique for the entire program. Given the size of strings, and the space overhead inherent to an object instance, by directly using strings as keys, python is much more memory-efficient than many other languages.

4.4 Clustering Implementation

In order to have a good understanding of how well the hierarchical clustering performs, it should be compared to a more mundane, baseline clustering method. The method chosen was bottom-up agglomerative clustering, which computes the similarity between every pair of clusters, merges the two most similar clusters, and repeats this process until the highest similarity between clusters reaches a value below the merge threshold. This requires computing $O(n^2)$ cosine similarities for n nodes in the hierarchy, as well as storing these similarities in a matrix of size $n \times n$. This was realized using the matrix implementation from the `NumPy` package.

It should be noted that this complexity is much higher than that of the hierarchical clustering. The hierarchical algorithm only computes one cosine distance measure for each edge in the graph of the hierarchy - one linking each non-root node to its parent. This means that for a hierarchy of n nodes, the algorithm computes $n-1$ cosine distances, which is $O(n)$. In practical terms, this means to process the test datasets described in Section 5.1, the hierarchical algorithm needs less than 1 GiB of RAM and takes less than 2 minutes. In contrast, to store the large matrix of similarity measures, the bottom-up algorithm required over 8 GiB of RAM. As the most capable computer that was available had 8 GiB of RAM, the algorithm's performance was further degraded by its use of virtual memory. Also, the bottom-up algorithm's performance is dependent on the threshold value used; the lower the merge threshold, the higher the number of clusters that are merged. Since merging a cluster means recomputing the similarities between the merged cluster and every other remaining cluster, the time to run the system can vary drastically. For threshold values over 0.6, the system took less than 10 hours to run, while for a value of 0.1, the run-time was over 27 hours.

Chapter 5

Analysis

Most typical computer algorithms solve a problem for which there exists an exact solution. For basic computing tasks, this is taken for granted. For example, a sorting algorithm is expected to put every element in a sequence in order. Different algorithms have different characteristics, in terms of run-time complexity, memory-space complexity, and the stability of previous orderings, but any sorting algorithm is expected to produce an ordering that puts every element of the sequence in order. Failure to do so means that the algorithm is incorrect, and cannot be used.

Some problems are harder to solve. In some cases, it may be simple to produce an algorithm that would find an exact solution, but the number of possibilities to consider is so large that it is impossible for an actual computer to solve. In certain cases, it is even impossible to *verify* that a solution is indeed the optimal one, since that would involve verifying exhaustively that no better solution exists. Algorithms that address these issues use some kind of approximation, which results in a "good" solution that may not be optimal. NP-Complete problems, such as the Travelling Salesman Problem, are examples of such particularly-hard problems.

When dealing with Natural Language Processing tasks, the problem is further compounded, as there is no optimal solution to be found. This difficulty stems from the very nature of natural language. Humans use language creatively, with different assumptions, which makes it easy for two

people to have different understanding of the same text. Computing languages have been developed specifically to allow people to specify a complex problem in a way that is unambiguous, to guarantee that both the computing machine and its human operator comprehend the problem the same way. There are no such guarantees in human languages; people often misunderstand each other. This makes it difficult to objectively assess the performance of a system that retrieves information from natural language data.

In order to foster the development of Information Retrieval as a field, organizations have been set up that manage standardized retrieval tasks. One such organization is TREC, the "TExt Retrieval Conference", which currently manages challenge tasks in 8 different research tracks, for text in various domains. Each track consists of a collection of documents along with a number of queries for information held in this collection. Researchers are given the collection and queries and are challenged to create systems to produce results for those queries within a fixed time-frame. After the results are handed in, they are compared to the "correct" results, as determined by groups of people who read the texts in question. Comparing the results of a system with this "gold standard" yields quantitative measures, which can be used to compare different approaches on an evaluation scheme in an objective fashion. These measures, such as Precision and Recall, are described in more detail in Section 5.3.

5.1 Evaluation Data

While there are some social media datasets available, they are not very well suited to evaluating this kind of clustering. The ICWSM blog datasets for 2006 and 2009 [MOS09] do not contain evaluation classes. The TREC Blog06 collection was used for a variety of tasks related to topic classification in the TREC Blog Track; this could be applied to the topic clustering described here. However, the TREC topic classes only cover those subjects deemed relevant by the track organizers. This collection also does not encode the social structure of its contents. Postings are represented in two ways: the contents of the Blog's RSS feed, which does not include comments, and the actual HTML

page, which can encode the social structure of comments in a huge variety of formats. Given the size and variety of this collection, it is not practical to use it to evaluate this system.

However, there are some social media websites which feature relevance judgements. Such sites provide a community not only with a platform for discussion, but also for rating the quality of the texts posted. Users are attributed a number of points, which they can use to "vote up" questions and answers to these questions to attest for the quality and relevance of a given posting. Users can also "vote down" postings which they consider irrelevant or of poor quality. When reading a question, the user will see answers presented not in order of posting time, as is typically done on social sites, but in order of score, with the response deemed most pertinent presented first. Writing content that garners positive votes earns points for the user, ensuring that those whose merit has been recognized by the community have more voting power.

While a number of sites, such as ExpertsExchange[1], have used community ratings to help readers find relevant texts, this type of platform was further refined into the model described previously by the website StackOverflow[2], a site for programming questions. Its underlying platform, called StackExchange[3], has been adapted to provide hosting for sites on a wide variety of subjects, from systems administration to cooking. An alternative, open-source website platform called OSQA[4] has emerged, enabling anyone to make a social website featuring community-based relevance feedback on any topic. Aside from the underlying software, both platforms provide users the same set of tools to ask and answer questions and rate the quality of postings. The term "stack sites" will be used here to refer to such discussion websites, regardless of the underlying software.

Two different stack sites have been used to test the TopicAl system described in Chapter 4. The first site, MetaOptimize[5], concerns Machine Learning in the context of Natural Language Processing. The site is relatively small, so it can be processed in its entirety, yet large enough to provide a variety of different topics. The participants also share a basic level of technical expertise, which means there

[1] http://www.experts-exchange.com
[2] http://stackoverflow.com
[3] http://stackexchange.com
[4] http://www.osqa.net
[5] http://metaoptimize.com/qa/

	MetaOptimize	Moms4Mom
Threads	711	1165
Posts	4492	10956
Terms	9881	14289

Table 1: Basic statistics on the evaluation datasets

is less noise[6] than on a more general site. This makes it easier to find meaningful clusters.

The second site, Moms4Mom[7], focuses on parents giving advice to other parents on how to raise their children. It was selected because, unlike MetaOptimize, it does not cover a technical subject, and does not contain much specialized language. However, given its serious subject matter and active voting community, the level of noise in the discussion is still quite low. Its size is roughly thrice that of MetaOptimize, which is still small enough to process the site in its entirety. Some figures on the size of these datasets are shown in Table 1.

Using the post ratings found in stack sites provides a standard against which to measure the clusters output by TopicAl. Using stack sites also has the added benefit that there are few formatting artifacts, such as the signature blocks and quoted text typical of e-mail discussions. Coupled with the consistent format found in HTML pages generated by these sites, this means that the system's output is not dependent on the quality of parser heuristics, and gives a fair picture of how the model itself fares. Having these two datasets will help highlight where the differences in content influence the system's output, and how consistent the system is in the face of varied input.

5.2 Evaluation Issues

While social websites featuring community ratings are not built with the rigorous guidelines that would be used in a collection intended for automatic evaluation, they do provide relevance judgements from knowledgeable users, against which a system can be compared. The approach that was used here was to use these user-attributed scores to initiate a new clustering process. The clusters

[6]Here noise means rude, antisocial users, personal communications in a shared channel, users going deliberately off-topic, advertisers hijacking the discussion, and other ways nefarious users actively undermine the conversation. Depending on participation, the user moderation of stack sites is somewhat effective at combating this behavior.

[7]http://moms4mom.com

generated this way can be compared with the TopicAl clusters output.

Since this website was not intended for automated evaluation, there are a number of issues to be dealt with before it can be used to evaluate the TopicAl system. The most obvious difference to deal with is the different kind of score used. TopicAl will merge clusters based on the cosine similarity between their TF-IDF vectors, and this similarity is bound to the range [0.0, 1.0]. In contrast, the scores given by users of stack sites are integers which are (theoretically) unbounded, and sometimes negative.

In order to use the integer scores for clustering, one could simply use a different threshold value when merging clusters. This is problematic because of the unbounded nature of the integer scores. When scores are limited to a finite range, this range expresses the complete spectrum between "relevant" and "irrelevant". When scores vary, it is not only because of the relevance of a particular post, but the interest that it generates. A perfect answer to a more particular subject may be scored lower than an unpopular answer for a popular question. Therefore, it is difficult to relate scores posts from different discussion threads.

One way to deal with this issue is to make the observation that for every question, the highest scoring answer is the most related available answer. Based on this observation, the highest-scoring answer can be mapped to a cosine similarity of 1.0. The lower scores can then be scaled to the highest-scoring answer's score, putting them in the [0.0, 1.0] range. One issue to take into account is the possibility of negative scores. As was noted earlier, users make use of their points to vote postings up *or down*; a particularly ill-received posting may ultimately reach a negative score. This raises the issue of which integer score is mapped to 0.0. If the very lowest negative score is mapped to 0.0, postings with an integer score of 0 will have their score mapped to a positive score. In fact, any other ill-received post with a higher, though still negative, integer score, will be mapped to a positive score. This is not desirable. Because of this, the evaluation procedure was implemented to map the integer score of 0, as well as anything with a lower integer score, to the decimal score 0.0. The integer score of 0 is considered to mean "completely irrelevant", and this scheme does not

Algorithm 4 StackSiteRescore

```
Require: thread node
  maxscore ← 0
  scaledscores ← ∅
  for all post ∈ thread.children do
    if dataset.scores[post] > maxscore then
      maxscore ← dataset.scores[post]
    end if
  end for
  if maxscore ≤ 0 then
    maxscore ← 1 {prevent division by zero}
  end if
  for all post ∈ thread.children do
    if dataset.scores[post] ≤ 0 then
      scaledscores[post] ← 0.0
    else
      scaledscores[post] ← dataset.scores[post] ÷ maxscore {floating point division, decimal result}
    end if
  end for
  return scaledscores
```

consider that anything can be more irrelevant than "completely". The procedure for mapping these scores is shown in Algorithm 4.

Another issue is that the scoring relationship model used for TopicAl does not exactly match the model used by stack sites. On stack sites, questions receive scores, as do answers. Comments, which are what stack sites call answers to answers, also receive a score. This means that everything that the hierarchical model calls a "post" is scored by users for interest and relevance. Individual passages, however, do not receive independent scores from users. This means that while the clustering of threads and posts can be compared to human judgements, passage clustering cannot.

Another fact to keep in mind is that the scores change over time. The scores, much like the text, are generated by the users, and will evolve as the discussion takes place. Running and evaluating the system at two different points in time will undoubtedly produce different results. Therefore, any evaluation using such a dataset is directly tied to the date and time at which it was produced. The results presented here are based on the state of both sites on February 16, 2011.

5.3 Evaluation Methodology

Once the scores taken from a stack site have been re-scaled, using the procedure outlined in Algorithm 4, they can be used to assess the quality of the clusters output by the TopicAl system. In order to do this, a new clustering process is initiated. The clustering method is the same as the one used by TopicAl to build its clusters; the important difference is that instead of using the cosine similarity between the term vectors of nodes, this clustering is done with the re-scaled scores taken from the stack site. These clusters are assumed to be accurate for the purposes of evaluation; they constitute the "gold standard". The threshold value used to generate these evaluation clusters is the same as used to run the clustering on the hierarchical index.

This clustering is treated as a binary classification problem: whether or not a post node belongs in the same topic cluster as its parent thread node. By treating the clusters generated from the users' scores as accurate, the clustering decisions obtained from the hierarchical index can be classified as true positives, true negatives, false positives and false negatives. Here, "true positive" means both the TopicAl system and the evaluation procedure agree that two nodes belong in the same cluster, while "true negative" means they agree the nodes should be in separate clusters. A "false positive" is a case where the system has clustered two nodes together, while the evaluation procedure has determined they belong in separate clusters. Conversely, a "false negative" means the TopicAl clustering put two nodes in separate clusters when the evaluation procedure states they belong in the same cluster. The procedure for classifying the clustering results this way is shown in Algorithm 5.

These classes can then be used to calculate measures of precision, recall, and accuracy, using the cluster evaluation methods described in [MS03]. While these are well-known measures used to evaluate a system that retrieves documents from a collection, they take a slightly different form in the context of evaluating clustering. Here, *Precision* means the proportion of instances where the system put a post in the same cluster as the thread while the evaluation clusters also put post and thread in the same cluster. *Recall* means the proportion of instances where the evaluation dataset has the post and its thread in the same cluster while the TopicAl system also clustered them

Algorithm 5 StackSiteEvaluation

```
Require: 0.0 ≤ threshold ≤ 1.0
  TP ← 0 {true positives}
  TN ← 0 {true negatives}
  FP ← 0 {false positives}
  FN ← 0 {false negatives}
  for all threadnode ∈ threads do
    evalscores ← evalscores(threadnode)
    for all postnode ∈ threadnode.children do
      if evalscores[postnode] ≥ threshold {The nodes should be clustered together} then
        if threadnode.cluster = postnode.cluster {The nodes are clustered together} then
          TP ← TP +1
        else
          FN ← FN +1
        end if
      else
        if threadnode.cluster = postnode.cluster then
          FP ← FP +1
        else
          TN ← TN +1
        end if
      end if
    end for
  end for
```

together. To put it another way, the precision score is the proportion of cluster results that are accurate, while the recall score is the proportion of accurate clusterings which were found.

These two measures are complementary; raising precision typically means lowering the recall. For example, a system that retrieves every document in a collection, or outputs a single cluster which contains every input element, would show 100% recall, but a precision close to 0%. Neither of these measures paints a complete picture by itself. This is the motivation behind the *F1-Score*, which is simply the mean of the precision and recall. The *Rand Index* is a different measure, which is more specific to the evaluation of clustering methods. It is defined as the proportion of clustering decisions where the generated clusters and evaluation dataset's clusters are in agreement - whether that decision resulted in the post being clustered with its thread or separately.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{8}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9}$$

$$\text{F1-Score} = \frac{\text{Precision} + \text{Recall}}{2} \tag{10}$$

$$\text{Rand Index} = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

Both the F1-Score and the Rand Index are intended to give an overall assessment of the quality of clustering in a single measurement. The way they approach this goal is different. Unlike the F1-Score, which is simply the average of the Precision and Recall scores, the Rand Index aims to report on the accuracy of all merging decisions, including those which did *not* result in a merge. Of all the evaluation measures presented here, the Rand Index is the only measure that incorporated true negatives into its calculation, as shown in Equation 11. This means that in cases where the data is heterogeneous, and should yield more numerous, smaller clusters, the Rand Index shows a more accurate picture.

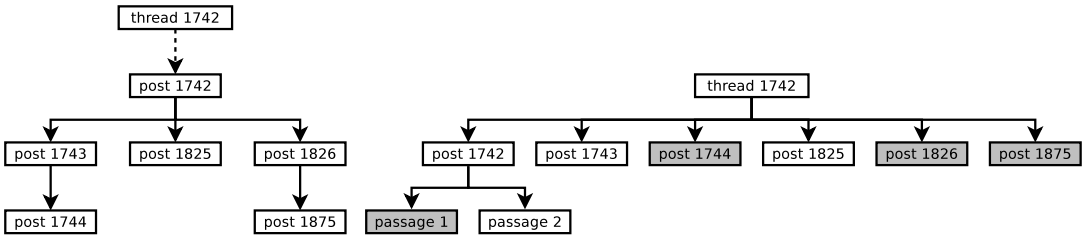
All of this means that, while the system's clusters can be compared to clusters derived from user feedback, and give familiar-sounding measures like "precision" and "recall", these measures should not be taken blindly, and should especially not be taken to have the same meaning they do in the field of Information Retrieval. In fact, closer inspection reveals a number of interesting consequences of hierarchical clustering, and yields some insight as to the accuracy of the evaluation strategy used here.

5.4 Case Studies

This section presents the result of the hierarchical clustering procedure on two sample threads. Closer observation of these threads highlights cases where the clustering performs well and where it does not. One is from the MetaOptimize dataset, the other from the Moms4Mom dataset; both are shown after hierarchical clustering with a threshold of 0.3. They were selected for different reasons. The first shows a case where the clustering performed exactly in line with the evaluation procedure, and closer observation confirms the accuracy of the results. The second shows a case where the clustering lamentably fails to produce results in line with the expectations of the evaluation

procedure; however, the causes for these discrepancies are varied, and not all of them indicate poor performance on the part of the clustering procedure.

5.4.1 MetaOptimize Thread 1742



(a) reply structure (b) light nodes are in the thread's cluster, shaded ones are not

Figure 5: MetaOptimize thread 1742

post 1742	QA score: 1	class: true positive
Hello all,		
I'm **looking** for a **decent implementation** of **deep belief networks** in **Java**. I've **found jaRBM**, but I think that it is just a **RBM implementation** and it seems not being **updated anymore** (Although I **haven**'t **fully checked** it). I know there are **stuffs written** in **python**, but I need a **Java implementation**.		
post 1743	QA score: 2	class: true positive
I know that the **Mahout project** is **planning** to **add** an **implementation** of **stacked RBMs** . **Apparently** the work **has started** in **this gihub branch** but I **haven**'t **tried** it yet.		
post 1744	QA score: 0	class: true negative
hmm great **news ogriel**. I'm going to **review** the **code** and **try** it if it is **usable** or not.		
post 1825	QA score: 3	class: true positive
If you have **RBMs** or **stacked RBMs**, it is **trivial** to get **DBNs** from that, as far as **sampling** from the **DBN** is **concerned**. If you want to **fine-tune** the **model** for **something** like **supervised prediction** or **classification**, you also need **neural net code**, but that **already exist** in **Java**.		
Post 1826	QA score: 0	class: true negative
If you have a c/c++ **implementation**, you can make a **wrapper** to it and **call** it from **java**.		
Post 1875	QA score: 0	class: true negative
No, **unfortunately** not, if I had a C/C++ **implementation**, I'm **aware** that it is **possible** to **call** these **libs** or **APIs** with **JNI**.		

Figure 6: Scored contents of thread 1742 in the MetaOptimize dataset

Figure 5a shows a discussion thread taken from the MetaOptimize dataset. It contains 6 posts; the first is composed of 2 passages, while the others have a single passage. Posts 1742, 1743 and 1825 have obtained some points from the community, as shown in Figure 6, where the text of each post is reproduced in full, with the index terms highlighted in **bold**. Table 2 shows the terms with

term/node	thread	1742	1743	1744	1825	1826	1875
rbm	0.18	0.03	0.05		0.07		
java	0.11	0.06			0.03		
dbn	0.11				0.17		
stack	0.10		0.07		0.05		
implement	0.09	0.05	0.03			0.10	0.05
jarbm	0.07	0.08				0.17	
...	...	...	...	...	...	...	...

Table 2: Index term weights for thread 1742, and posts therein, in the MetaOptimize dataset

the highest weights in the thread, along with the weights for each of these terms in each post in the thread.

The term with the highest weight for the thread as a whole is "rbm", which is the acronym for "Restricted Boltzmann Machine". This term is relatively frequent in this thread, appearing four separate times, which ensures it gets a high term frequency for the thread. It is also a relatively rare term among other threads, giving it a low document frequency for the thread. The low document frequency, along with the high term frequency, account for its high TF-IDF score.

However, as its 4 appearances occur in 3 separate posts, its document frequency for posts in the context of this thread is also relatively high. This explains why the individual posts bearing this term have a lower weight for it than the whole thread. In particular, post 1825 bears 2 of the 4 instances of the term "rbm", which accounts for a larger proportion of term instances in the post than in the thread - 2 out of 21 term instances, as opposed to 4 out of 78 - yet its weight for the post is less than half of what it is for the thread. This means that the term "rbm" has more to do with the topic of the thread as a whole than the topic of that particular post.

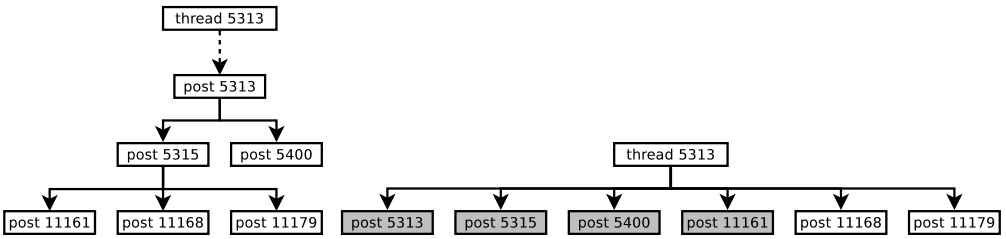
Compare this to the term "dbn" - an acronym meaning "Deep Belief Networks" - which is tied for second place in weight for the thread, as seen in Table 2. Similarly to "rbm", the term "dbn" is relatively frequent in the thread and relatively infrequent outside the thread. However, unlike "rbm", all 2 instances of the term "dbn" appear in post 1825. This means that the document frequency for the post remains low, while the term frequency is even higher than for the thread. That is the reason why the weight of "dbn" for post 1825 is even higher than the entire thread. In terms of topicality, "dbn" has more to do with the topic of that particular post than the thread as

a whole.

Figure 5b shows which nodes in the thread's hierarchy belong to the same cluster as the thread. When clustering with a threshold value of 0.3, the system's clustering of posts matches that of the evaluation clusters. It is also worth noting that, while the second passage of post 1742 lies in the same cluster as the post and the thread, the first passage does not. Looking at the text for this passage, it is clear why that is: the passage has only one index term, which is not repeated anywhere else in the post, or in the thread for that matter. This means the TopicAl system has determined, correctly, that while the content of post 1742 has the same topic as the thread overall, only its second passage bears this topical content.

5.4.2 Moms4Mom Thread 5313

While the hierarchical clustering performed well for MetaOptimize thread 1742, it did not perform as well in some cases. In the Moms4Mom dataset, thread 5313 performed particularly poorly, for a variety of reasons. This thread starts with a parent asking what to expect when a child consults with a doctor about a lazy eye. The contents of the thread are shown in Figure 8. Figure 7a indicates the thread's structure by showing which posts are replies to previous posts, while Figure 7b shows which posts were clustered together with their containing thread.



(a) reply structure (b) light nodes are in the thread's cluster, shaded ones are not

Figure 7: Moms4Mom thread 5313

For this thread, the clusters output by TopicAl do not match the clustering expected from the user scores. None of the posts clustered together with their thread should have been, while only one of the posts was correctly excluded from the thread's cluster.

post 5313	QA score: 3	class: false negative
My 4 year old **daughter has** been **recently diagnosed** with a **lazy eye**. Next **week** we have an **appointment** to see a **pediatric opthamologist**. **Has anyone else gone** through **this before**? What can I **expect regarding** the **appointment** and **treatment**?		
post 5315	QA score: 2	class: false negative
Wikipedia has some **information** on **common treatments**. **Looks** like your **daughter** will have to have a **patched**-up **eye** for some **time**. Best **wishes**!		
post 11161	QA score: 0	class: false positive
We'**ve already talked** to her about an **eye patch** a **little** and **told** her that she might have to **look** like a **pirate** for a while. She thought that **sounded fun** - we'**ll** see how long that **attitude** lasts.		
post 11168	QA score: 0	class: false positive
That's **actually** a **nice idea**... I **presume** the **eye patch** will be a **white adhesive patch** - get a "**classical**" **black patch**-on-a-**string**, **maybe paint** a **white skull**-and-**bones logo** on it...		
post 11179	QA score: 0	class: true negative
@**mkcoehoorn** From my **experience** working with **children** who have had **patched eyes** the thing that often **frustrates** them is that they will **patch** the good **eye** in order to **exercise** the **lazy eye**. That is **frustrating because** the **child has** to work **harder** to see. In my **experience** they often get **tired** of not seeing **clearly** and want to **remove** the **patch**.		
post 5400	QA score: 1	class: false negative
My **daughters eyes cross**, so we are **trying glasses** first to see if they **help** the **eyes** to **straighten** out. I **hope** so **because** we **don**'t want to have to put her through a **surgery**. She is a year and a **half**, and she just **screamed** so much while the **doctor looked** at her **eyes**. My **husband** and I had to **hold** her down, so she could **look** at her **eyes**. I **was really disappointed** with **this pediatric doctor**...you would think if she works with young **children** she would be a **bit** more **friendly** or know how to work with her better. Though, we knew she would most likely **cry**, but I would have thought she would be a **bit** more **understanding** with our **daughter**.		

Figure 8: Thread 5313 from the Moms4Mom dataset

Part of this is because of how users behave when scoring posts. As was noted earlier in Section 5.1, stack sites feature three "types" of posts: questions, answers and comments. The question initiates a thread, answers are replies to the question, and comments are replies to answers. Here, post 5313 is the question, posts 5315 and 5400 are answers, and posts 11161, 11168 and 11179 are comments, as can be seen in Figure 7a. The point of these sites it to provide answers to questions; since comments are not directly replying to the question, they often do not receive scores. They are intended for meta-discussion, so they are not an *expected* source of text directly pertinent to the question - users are expected to write such text in the form of an answer. As such, they might share a high similarity with the thread's overall contents, and could indeed be related to its topic, but the way users interact with them does not provide an easy way to verify this by comparing against their score.

The way clustering on cosine-similarity works actually has an opposite effect. Since comments

are typically shorter than full answers, they have a smaller term-space, with fewer dimensions that carry weight. This means they bear fewer factors that could differentiate them from the thread as a whole, making them more likely to be merged into the thread cluster. This could be problematic, but in this case, the two comments that TopicAl clustered with the thread, comments 11161 and 11168, do seem to provide useful insight on how to encourage a child through a difficult situation. Both mention trying to turn a possibly traumatic experience into an opportunity for play, with the goal of making the experience more positive. It could be argued that while these comments are tangential, they are more useful to the parent who asked the question than the actual answers.

Post 5313 is the question that originated the thread. It uses some specific terms that are found nowhere else in the thread, and are rare in the dataset as a whole: "ophtamologist", "diagnosed", "appointment". Since "ophthalmologist" was misspelled as "ophtamologist", this helps separate this thread from other threads that could mention the same medical specialization. These serve to differentiate the original post from the rest of the thread.

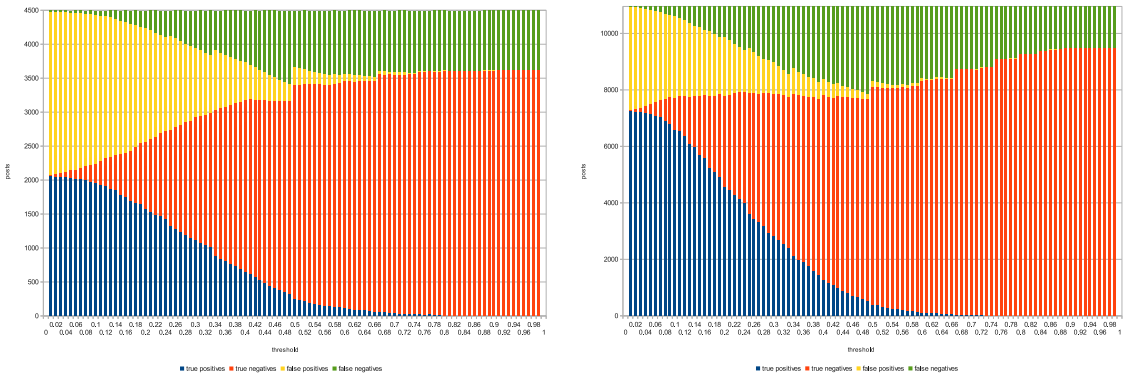
Post 5315 is the highest-scoring answer in this thread, which means users judged that it carries the information most pertinent to the question. However, this information comes in the form of a link to a Wikipedia article. TopicAl does not consider links to other webpages, as they are removed from the text before processing, along with other HTML formatting. As such it is indeed a pertinent post, but the information it contributes is not found in the text of the post itself.

Post 5400 comes from a parent who dealt with a similar situation. This answer to the question is actually a personal story of the disappointment that parent felt when the pediatric doctor could not contain the child's crying. This is related to the question, but arguably provides little actionable knowledge that the author of the question could use. Being the longest post in the thread, it displays a larger variety of vocabulary than other posts. When indexing, this larger vocabulary becomes a wider term-space where most terms have little in common with the rest of the thread: "screamed", "surgery", "husband", "disappointed". It could then be argued that TopicAl was correct in excluding this answer from the cluster of posts related to the thread.

The only post that is "accurately" clustered is comment 11179, which is excluded from the thread's cluster. This agreement between the TopicAl clustering and the expected relatedness inferred from the score should provide some comfort, but reading its text tells a different story. The comment is apparently written by someone who has dealt many times with the situation described in the original question. It seems to provide the most straight-forward answer to one part of the question, namely what to expect from the treatment. In doing so, it is arguable more on-topic than any other answer or comment.

In short, thread 5313 not only shows some difficulties in hierarchically merging clusters based on the similarity of their term vectors; it highlights that even the standard it is held up against is far from perfect. Both the cosine-similarity and the user-attributed scores fail to accurately represent how related or interesting a post is to the discussion. It is one of the most spectacular cases of failure; as Section 5.4.1 showed, both the TopicAl clusters and the evaluation derived from user-scores is typically more accurate. However, it it important to keep in mind that the results of quantitative evaluation should not be taken blindly.

5.5 Examining Hierarchical Clustering Results



(a) MetaOptimize (b) Moms4Mom

Figure 9: Classification of results of hierarchical clustering at different thresholds

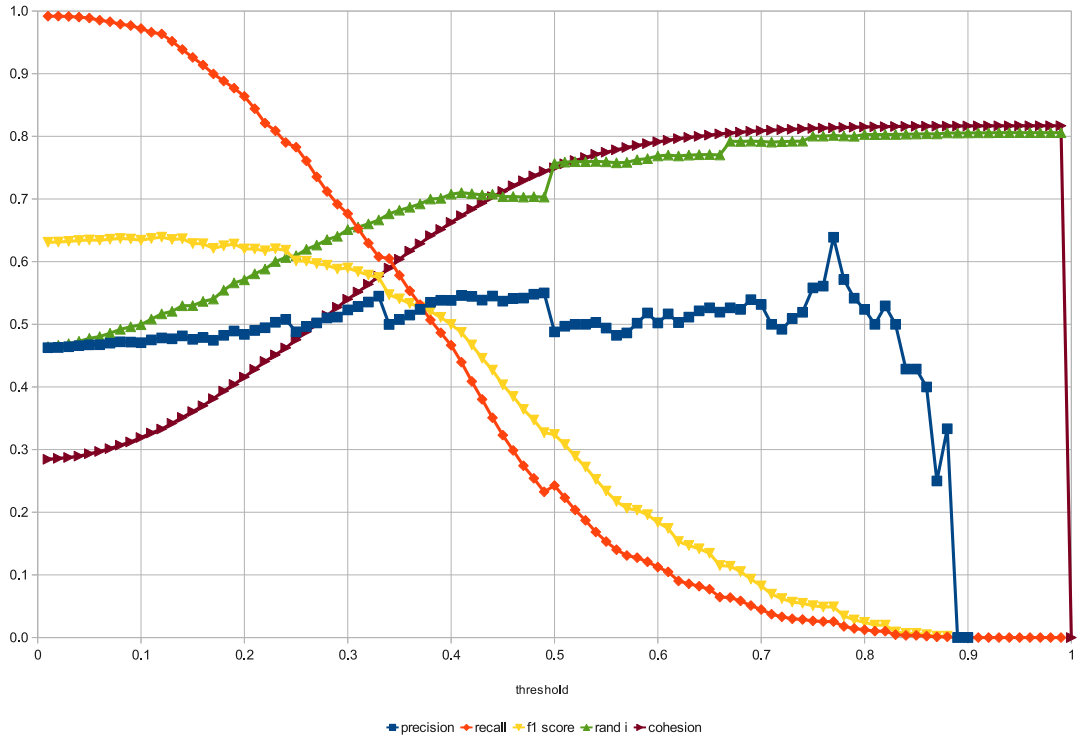


Figure 10: Evaluation of hierarchical clustering for the MetaOptimize dataset

With the procedure detailed in Algorithm 5, all clustering decisions can be put in one of 4 classes: true positives in blue, true negatives in orange, false positives in yellow and false negatives in green. The results of these classifications is dependent on the threshold value used to merge clusters and evaluate them. These results are shown for both datasets in Figure 9, for all threshold values of two significant decimals in the range [0.01, 0.99].

These classifications can be used to compute measures of the accuracy of clustering that were described in Section 5.3. Figure 10 shows the results of these measures on the hierarchical clustering of the MetaOptimize dataset, for every threshold value between 0.01 and 0.99. Figure 11 shows the same measures for the Moms4Mom dataset. Both figures are reproduced in detail in Appendix A.

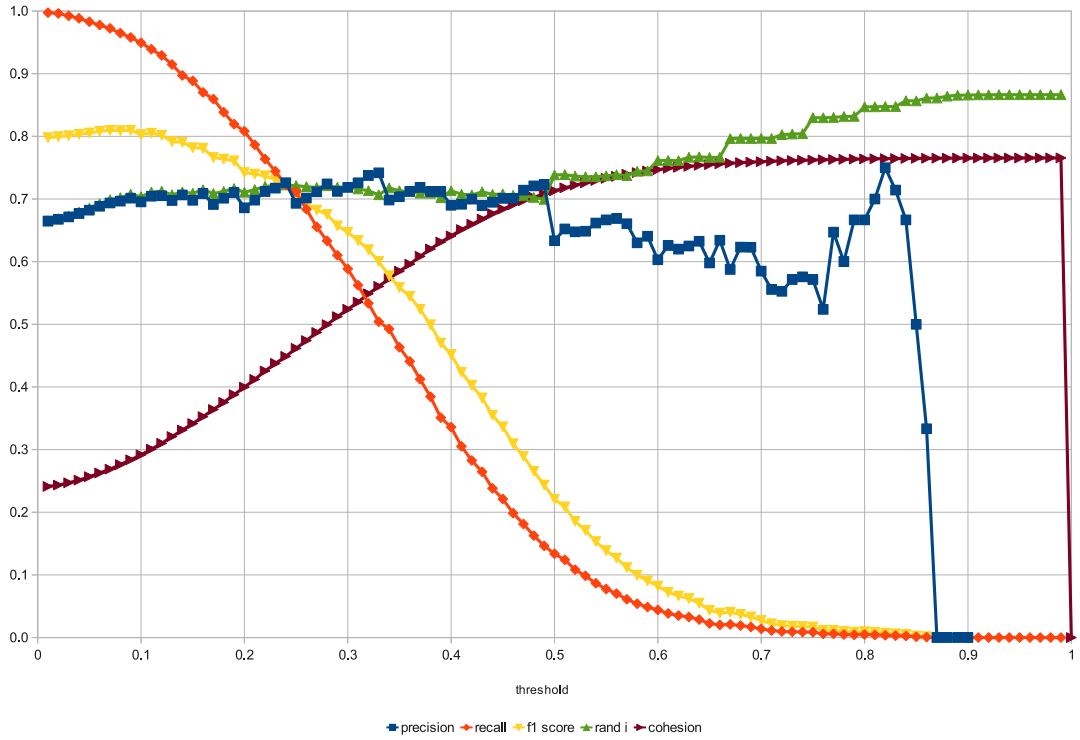


Figure 11: Evaluation of hierarchical clustering for the Moms4Mom dataset

5.5.1 Precision

One startling characteristic of the precision measurements is the abrupt changes in value at certain threshold values. Notably, there are sharp breaks in the precision measures at threshold 0.25, 0.33 and 0.5. This is a by-product of the evaluation method used. The QA websites used for this work let their users assign points to rate the quality of postings, and these point totals are integers. When these scores are scaled to the [0.0, 1.0] range, they are divided by the highest score found in their containing thread, as described in Section 5.3. In threads where the maximum score is small, the scaled scores become fractions with a small denominator. For example, in a thread where the highest-scoring post has a score of 4, a post with score 2 will see its score scaled to 0.5.

This scaled score is the evaluation target, as it determines whether or not, for a given threshold value, the post *should* or should *not* be clustered together with its thread. As cosine similarities depend on the presence of many terms, there is a large variety of similarity scores, which are more

evenly distributed. The result of the cosine similarity measure is rarely such a crisp fraction. This means that there are many posts which change classification around threshold values for such crisp fractions. Using the above example of a post with a QA score of 2, in a thread where the highest-scoring post has a score of 4, when the post is clustered with its thread, it is considered as a true positive with a threshold below 0.5, while it is classified as a false positive for threshold values of 0.5 and over. This issue becomes less relevant as more users cast more votes on the importance of questions.

For threshold values over 0.5, the precision score varies wildly, until a point where it reaches 0. This is because the precision calculation considers only results classified as true positives and false positives. Both of these classes represents cases where a post *was* clustered together with its thread, whether rightly or wrongly. With higher threshold values, there are very few cases where clusters are merged; At threshold 0.5, only 20% of nodes are merged. This means that when computing the fraction in Equation 8, both the numerator and the denominator are so small that slight variations in either have enormous effects on the Precision measurement. Threshold values over 0.5 yield so few clusters that they are not useful for practical purposes.

It should also be noted that except for the aforementioned spikes, the precision scores are more or less constant for the useful range of threshold values. In the MetaOptimize dataset, the precision score remains in the range [0.45, 0.55] for all threshold values below 0.7, while in the Moms4Mom dataset, the precision score remains in the range [0.65, 0.75] for all threshold values below 0.5. This seems to indicate either an upper boundary on the accuracy of the clustering, or that term occurrence outside the context is almost irrelevant

To summarize, the precision score changes very little with regards to the threshold value used, but is very susceptible to side-effects of the evaluation procedure. In light of this, there are serious doubts that the precision score computed here is a useful assessment of whether the successful clusterings are “accurate”.

5.5.2 Recall

The Recall scores for both datasets vary between almost 1.0, when the threshold approaches 0, to nearly 0.0, when the threshold approaches 1. The distribution of recall scores for various threshold values follows a sigmoid curve, with some small discontinuities visible at threshold 0.33 and 0.5. As with the precision scores, these discontinuities are the result of the way the integer scores are mapped to decimal numbers, leading to threshold values where many nodes change evaluation class.

These discontinuities are more visible in the MetaOptimize results, but can also be found in the Moms4Mom results. This is because instead of considering the false positives, the recall fraction has the false negatives as part of its denominator. When approaching high threshold values, the number of false positives drops to 0, while the number of false negatives stays almost constant. This means the denominator of the fraction does not reach very small values, which explains why there are no abrupt changes in the recall scores for high threshold values. The number of false negatives *does* approach 0 when the threshold values approach 0.0; for those threshold values, however, the number of true positives is so large that the denominator remains large, which ensures the fraction does not change value sharply at certain threshold values.

As a consequence, unlike the Precision scores, the curve for Recall scores is very smooth. This stability of the measurements indicates that the Recall score is less susceptible to side-effects of the evaluation procedure. As such, it could be taken with a higher confidence in its usefulness than the precision score.

The recall curves also confirm that higher threshold values are not useful for practical clustering. At a threshold value of 0.5, the recall score for the MetaOptimize dataset is 0.2426; in the Moms4Mom dataset, it reaches 0.1337. In both cases, this shows that only a small fraction of the useful clusterings were accomplished. This makes a strong case that the lower threshold values provide a more useful set of clusters.

5.5.3 F1-Score and Rand Index

The F1-Score also shows an advantage for the hierarchical clustering for the lower threshold values. However, this difference is mostly representative of the Recall score, as the Precision measurements are relatively stable for this range of threshold values. This means the F1-Score tends to give higher scores to classifications which produce the most clusters.

The Rand Index gives a different picture of the overall performance of clustering. It shows a low score for low threshold values, and a higher score for the higher threshold values. This is because unlike in other measures, true negatives count towards the Rand Index. For threshold values over 0.5, true negatives represent the majority of nodes classified in both datasets, as can be seen in Figure 9. The majority of other nodes are classified as false negatives, as the clustering at such high threshold values merges very few clusters, accurately or not. As such, the Rand Index tends to give higher scores to classifications which produce very few useful clusters.

5.6 Comparing Clustering methods

The figures presented in Section 5.5 give a good idea of how well the hierarchical clustering deals with different types of data. However, they paint an incomplete picture. In order to better understand just how well the TopicAl system performs, it has to be compared with other approaches. The measures of precision and recall were conceived for this specific purpose of comparing different approaches to a given task. The difficulty with this is that TopicAl attempts to solve a different problem than what existing systems were designed to do. This is why the evaluation procedure detailed in Section 5.3 had to be developed. There are no state-of-the-art results from last year's conference against which to measure this hierarchical clustering.

In order to paint a more complete picture, a baseline approach was necessary. The baseline used here is bottom-up agglomerative clustering. In this algorithm, one initial cluster is created for every element to be clustered. The similarity is computed between every pair of clusters, and the two most-similar clusters are merged. The similarities are then recomputed between the newly-merged

cluster and every other remaining cluster. This process is repeated until the highest similarity falls below the threshold value. This was chosen because it is one of the simplest clustering algorithms; it makes no assumptions on the data to cluster, and its performance is easy to understand.

The bottom-up clustering also serves another, more practical purpose. Since the hierarchical algorithm only considers merging clusters based on the structural links between them, there are some valid merges that it cannot accomplish. For example, it cannot merge two related posts which appear in two unrelated threads. This deficiency is an obstacle to using TopicAl for purpose of querying the entire collection for text on a given topic. As was detailed in Section 3.4.1, a practical implementation needs to supplement the hierarchical clustering with a bottom-up second pass. This is why this section presents results for three methods of clustering: hierarchical-only, bottom-up only, and a combined method with a hierarchical first pass and a bottom-up second pass.

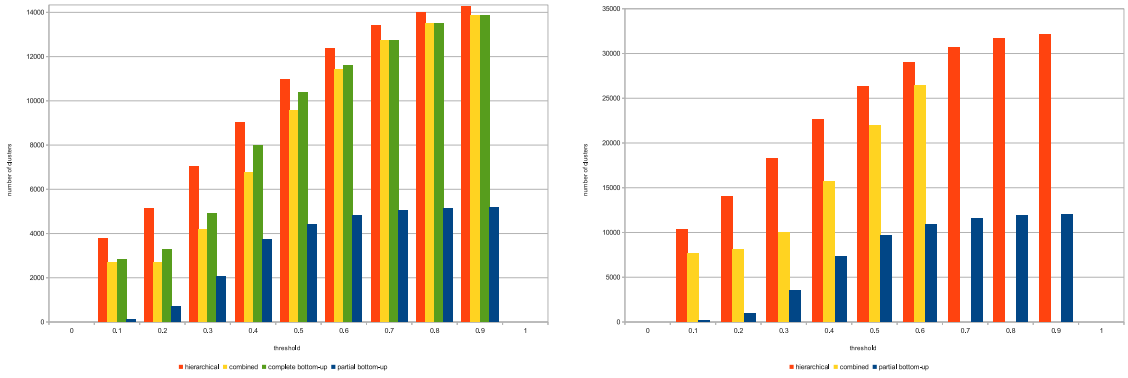
There is a challenge in implementing bottom-up clustering, however. Since it needs to merge the pair of clusters that has the highest similarity, it needs to compute and maintain the similarity measure between all pairs of clusters. This requires keeping a very large matrix of similarity values in memory. With the larger Moms4Mom dataset, this required more memory than the 8 GiB that the most capable available computer possessed. This means that there are no results for the bottom-up algorithm on the Moms4Mom dataset.

In order to still get some understanding of how well the hierarchical algorithm performed against a well-known approach, a substitute was needed. The substitute proposed here is a bottom-up clustering of one portion of the dataset: the thread nodes and the post nodes, but not the passage nodes. The passages were omitted because there was no evaluation metric available to determine whether or not a passage is related to its containing post. Because of this, it was possible that removing the passages would not alter the results significantly, and this partial clustering would still provide useful information about how well the hierarchical algorithm performed.

This assumption was not guaranteed to be true, and needed to be verified by itself. This is why for the smaller MetaOptimize Dataset, both the complete and partial clustering were performed.

This helped highlight where the complete bottom-up clustering performed similarly to the partial clustering, and where its performance differed.

It should be noted that some data points are missing for the assessment of the combined clustering for high threshold values in the Moms4Mom. This is also because of memory limitations. The hierarchical first pass reduces the number of clusters that the bottom-up second pass needs to consider. This reduces the size of the matrix of distances between cluster pairs to a size that can fit in the 8 GiB of memory of the most capable computer available, if the threshold value is low enough. For threshold values of 0.6 and under, the hierarchical clustering reduced the number of clusters enough to perform the bottom-up second pass. Since the more practically useful results are obtained with threshold values under 0.5, this does provide enough information to ascertain the usefulness of combined clustering.

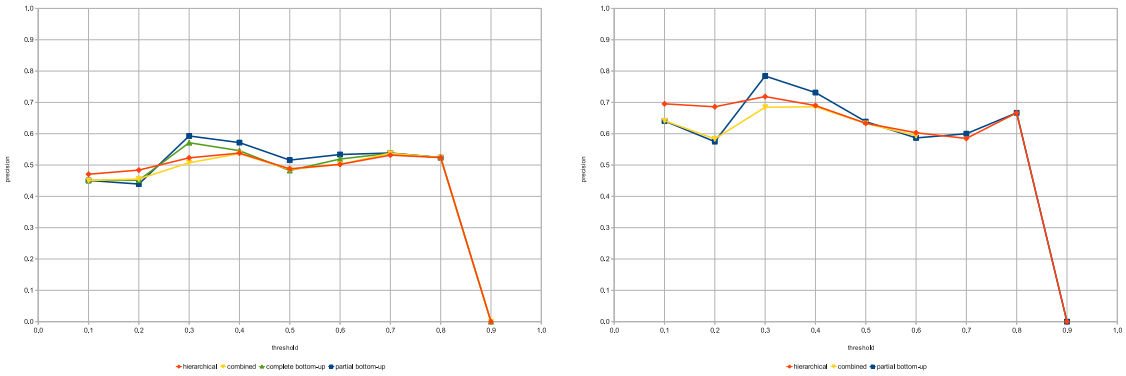


(a) MetaOptimize (b) Moms4Mom

Figure 12: Numbers of clusters produced by different clustering methods

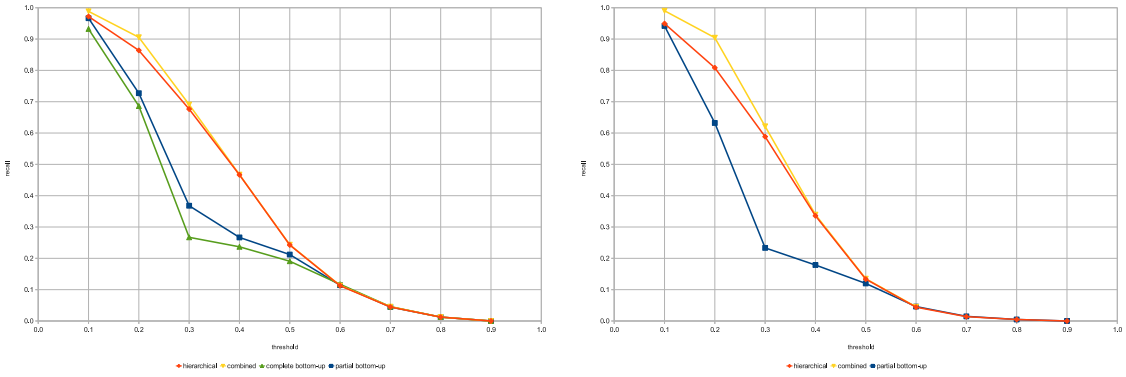
It should also be noted that since the partial bottom-up clustering worked on fewer elements, it produced fewer clusters. Figure 12 shows the numbers of clusters produced by each clustering method. The orange bar shows the number of clusters produced by hierarchical clustering, the yellow bar shows combined clustering, the green bar shows bottom-up clustering of the complete collection, while the blue bar shows the bottom-up clustering of only posts and threads. It can be readily observed that the partial bottom-up clustering produces a much smaller number of clusters.

The bottom-up agglomerative clustering performed here is much more expensive then the hierarchical clustering. In the larger Moms4Mom dataset, performing the bottom-up clustering at threshold values over 0.5 took on the order of 10 hours; at threshold 0.1, it took over 27 hours. This made it difficult to provide as many data points as in the assessment of the hierarchical clustering. The results shown in this section are only computed for threshold values of 1 decimal place in the range [0.1, 0.9], unlike the previous section.



(a) MetaOptimize (b) Moms4Mom

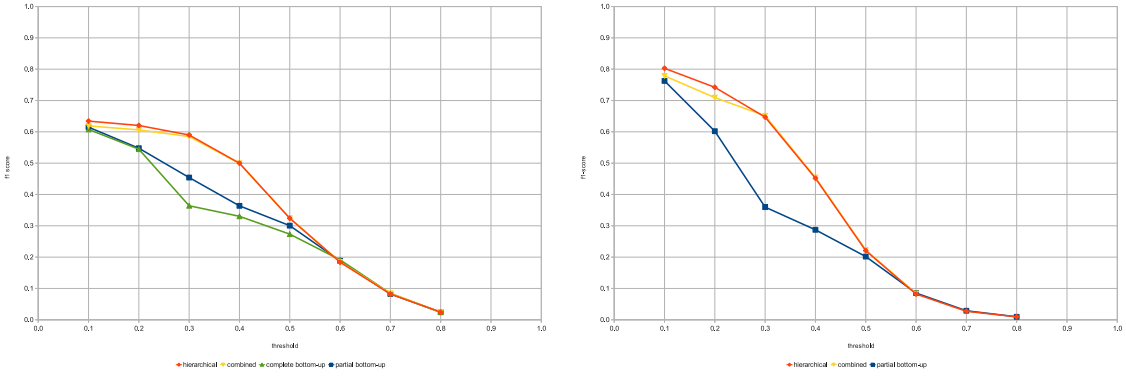
Figure 13: precision score for different clustering methods



(a) MetaOptimize (b) Moms4Mom

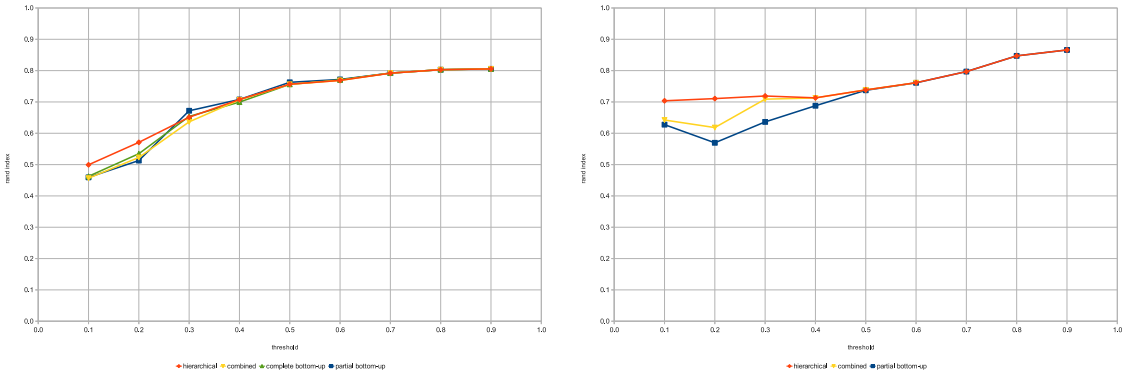
Figure 14: recall score for different clustering methods

It is reasonable to ask whether this approach to evaluation is objective, or whether it has an unfair bias favoring the hierarchical model over more traditional approaches. The scores from stack



(a) MetaOptimize (b) Moms4Mom

Figure 15: F1-Score for different clustering methods



(a) MetaOptimize (b) Moms4Mom

Figure 16: Rand Index for different clustering methods

sites, as they are used in evaluation, only relate a post to the thread it appears in. The hierarchical clustering has access to the link between post and thread, which standard, bottom-up clustering does not.

This does not appear to be the case. The evaluation scores for both datasets show that, for threshold values of 0.5 and over, the performance of both algorithms is very close. The only measure where there is a marked difference is precision, where the bottom-up algorithm has an advantage over the hierarchical algorithm. This is more noticeable for the MetaOptimize dataset, but it can also be observed in the results of the Moms4Mom dataset. The bottom-up algorithm, which merges clusters

based only on the most-similar pairings, regardless of structural links, can still find the most-related groupings, as established by the evaluation scores. This makes a strong case that the bottom-up algorithm has the same ability to find the related pairings as the algorithm that is informed about the structure.

The challenge then becomes to explain the large discrepancy in favor of the hierarchical algorithm for the lower threshold values. All measures for both datasets show that for threshold values close to 0.2, the hierarchical algorithm performs much better than the bottom-up algorithm. This gap is especially pronounced for the recall score. This can be explained because of the way bottom-up agglomerative clustering works. Two clusters are only merged when, of all the existing clusters, they are the pair with the highest similarity. In this situation, this means that their term spaces have the most in common of any pair of nodes in the text hierarchy - regardless of *where* in the hierarchy they are located. For high threshold values, this works well, as clusters are merged when their contents are nearly identical.

However, the situation changes over time. As clusters get merged, the maximum similarity between any two clusters becomes quickly much lower. When approximately 20% of clusters have been merged - or, more precisely, when the merging of clusters has reduced the number of clusters by approximately 20%, the maximum similarity between any two clusters reaches 0.5. This means that there are very few cases where the relatedness of two pieces of text is expressed by a very high similarity of their term spaces. Most related pairs of text nodes in the hierarchy have much lower similarity; this is where the bottom-up algorithm's tendency to favor only similarity begins to hamper its performance.

The bottom-up method will merge all nodes with a very similar term space together, regardless of their positions in the hierarchy. By greedily merging posts solely based on term-space similarity, the bottom-up algorithm fails to consider whether a post is related to the discussion in which it occurs. The resulting clusters are more homogeneous for high threshold values, but as can be clearly seen in the results, for low threshold values, which represent the similarity between most related

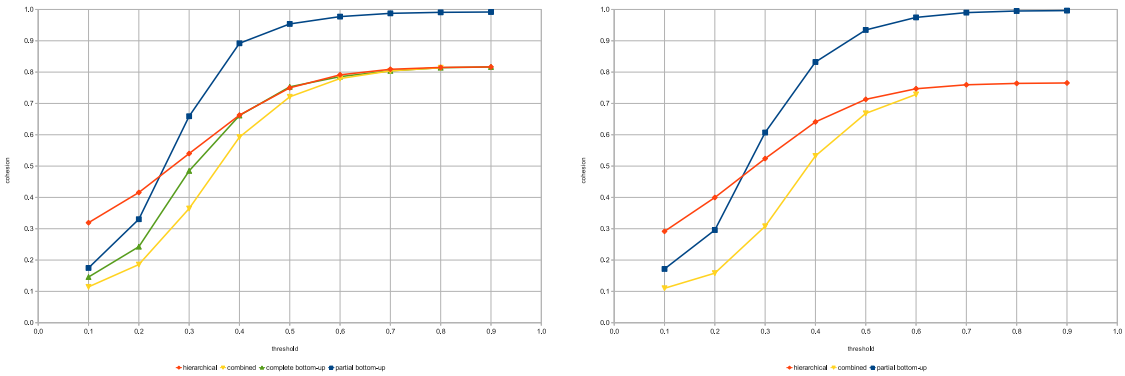
posts, the cohesion becomes *lower* than with the hierarchical algorithm.

By contrast, the hierarchical algorithm will only merge nodes that are linked in the hierarchy. For high threshold values, the clusters are notably less cohesive, but conversely, the cohesion is much less susceptible to change when the threshold varies. This is because the structure linking threads, posts and passages is the only factor determining which clusters can be merged. The bottom-up algorithm, in order to merge two pieces of related text with a similarity of 0.3, needs to merge every piece of text with a higher term-space similarity beforehand. In doing so, the clusters form around a core of common terms, but also accumulate every term with a low weight in the same term vectors. Individually, these low-weight terms make no difference to the results, but after thousands of clustering steps accumulate significant weight in a cluster's term space. Eventually, two pieces of related text end up in clusters that are not similar enough to be merged. By only considering pieces of text related by the social structure found in the discussion, the hierarchical clustering procedure can cut through this noise and, for the more useful, lower threshold values, produces superior results.

One obvious negative consequence of this is that the hierarchical algorithm has no way of linking related posts found in different threads. This is the reason why a practical system needs to complement the hierarchical clustering with a more basic, all-pairs type of clustering. Because it takes place after the first step, the bottom-up algorithm can avoid many of the pitfalls that undermined its performance when it was run alone. The clustering is not initiated on singleton clusters representing one piece of text, but on larger, more heterogeneous clusters that already account for some of the diversity of terms that can be found in related pieces of text. As can be seen from the results of combined clustering, the second pass accounts for a large reduction of the number of clusters, which does result in more heterogeneous clusters. While this results in a reduction of the precision score, it also leads to an improvement in the recall score.

5.6.1 Cluster Cohesion

There are some other quantitative measures that can be used to assess the hierarchical clustering. One possible measure of the effectiveness of clustering is Cluster Cohesion, which measures how much elements in a cluster are related to each other. This can be implemented as the mean cosine distance between the cluster's term vector and the term vectors of each clustered element, as shown in Equation 13. The average Cohesion for all clusters in the dataset is an indication of the quality of the clustering results. As with other measures, this is dependent on the dataset, as well as the threshold used to cluster the dataset.



(a) MetaOptimize (b) Moms4Mom

Figure 17: Cluster Cohesion for different clustering methods

$$\vec{C} = \sum_{i=1}^{|C|} \vec{v_i} \tag{12}$$

$$\textsc{ClusterCohesion}(C) = \frac{\sum_{i=1}^{|C|} \textsc{CosineSimilarity}(\vec{C}, \vec{v_i})}{|C|} \tag{13}$$

The cohesion results for both datasets are shown in Figure 17. They show that, unlike other measurements, there is a wide difference between the cohesion of partial bottom-up clusters and complete bottom-up clusters. When clustering without passages, the maximum cohesion at high threshold approaches 1.0, while bottom-up clusters built with passages have a maximum cohesion closer to 0.8.

One thing that becomes apparent is that the hierarchical clusters have roughly the same maximum cohesion as the bottom-up clusters, and the combined clusters. This could be taken to mean that the two clustering approaches are comparable, even though the hierarchical clusters are done by doing a small fraction of the work. However, while the clustering methodology varies, both these scores are computed on the hierarchical index. Since flat clusters can also be made using a more traditional, flat index, this warrants further investigation.

Another point of interest is that the clusters obtained by combining the two clustering approaches actually yields clusters that are *less* coherent than using either approach independently. This is contrary to initial expectations; the motivation of combining both approaches was to make *better*, more coherent clusters. However, this lower coherence does not necessarily mean the clusters are less useful. The combined clusters are the result of doing bottom-up clustering on the hierarchically-built clusters, which are already heterogeneous, as they only allow clustering together elements related by context. It would be difficult for a system clustering already-heterogeneous elements to combine them in ways which make them less disparate.

5.7 Concluding Remarks on the Evaluation Procedure

Assessing the quality of these results proved to be as much of a challenge as developing the system itself. This is, in part, because the TopicAl attempted to solve a different problem than what previously-available collections were intended to evaluate. The hierarchical indexing and clustering attempt to address the full breadth of topics covered in threaded discussion, including topics mentioned only in passing, in tangential, "off-topic" postings, without making any assumptions about what the reader's interests may be. Relying on user-attributed relevance scores provided a useful point of comparison, which can be obtained from social websites covering a wide range of topics. Performance concerns limited the choice of stack sites used for evaluation to relatively small communities, where user participation is not high enough to guarantee enough redundancy in scoring for the evaluation procedure to provide perfectly reliable results.

Nevertheless, the evaluation results shown here have provided useful insights into the performance of hierarchical clustering on social text. In particular, they have shown that hierarchical clustering, by focusing on social structure, will avoid being waylaid by spurious similarity between pieces of text that do not appear in a shared context. In a sense, this follows how human readers process threaded discussion; a person reading a post on a given topic will read other posts in the same thread before trying to find related posts in different threads. Hierarchical clustering avoids being "distracted" by following the same principle of looking within the context provided by the discussion structure. When implementing a reading aid, or a recommender system, which is intended to help humans parse the reams of social text, there are clear benefits in working with the same assumptions that human readers have.

Chapter 6

Conclusion

The hierarchical indexing and clustering presented here produced an improvement in the quality of clustering results over the baseline bottom-up approach, while significantly reducing the complexity of the clustering task. This suggests the hierarchical approach is better-suited to incrementally indexing the ever-increasing volumes of structured text currently being produced. This is because it integrates the notion of *context*, enabling it to consider a piece of text at different levels of granularity. Context is emerging as an important notion which researchers are using to narrow down focus on large amounts of text for different uses.

Social media poses challenges for Information Retrieval which are still unresolved. By opening the web to discussion by anyone, about any subject, this has produced enormous quantities of text which cannot be simply considered as "web-pages" without losing much of the information which they contain. This has spurred much new research into social text, where many different approaches attempt to find information from on-line discussion in ways that are informed by the structure of social text.

The approach proposed here was in line with the basic Computer Science principle of divide-and-conquer. Regardless of the specifics of the formatting used to represent discussion for the web, what we call social text can be understood using a common set of metaphors. A site can be considered as

a set of threads, a thread as a set of posts, and a post as a set of passages. This relationship between pieces of text forms a hierarchy, where each level provides the *context* within which to understand elements as the lower level.

The hierarchical indexing method proposed in this thesis uses this hierarchy of contexts to apply well-known text indexing methods in a recursive manner. This is unlike web search engines, which are not informed by social structure and treat social text as documents in whichever form the software represents them for the web. The hierarchical indexing treats a post both as a document in the collection that is its thread, and as a collection of its own passages. Likewise, the thread acts both as a collection of posts and as a document in the collection that is the entire social discussion platform. By considering the text at multiple levels of granularity, this approach allows the system to identify information mentioned only in passing, in an otherwise-unrelated context, while still giving a useful picture of the discussion as a whole. This method can be used when implementing the interface to a social-media website, to help the reader focus on parts of the discussion that are pertinent to the reader's interests by highlighting points where the discussion diverges to different topics. This would help readers focus on the information they require from the text, without the distractions that reliably arise in public discussions.

The hierarchical structure of social text can also be used to implement clustering, to group together parts of the discussion that share the same topic. By focusing on the text content within its different contexts, hierarchical indexing allows to make more accurate representation of which information is specific to a passage, which is related to a single complete post and which is related to an entire discussion thread. This way, a post that changes subject from its discussion thread can be clustered with text on this specific topic, while the other posts in the thread, as well as the thread as-a-whole, will belong in another cluster.

Using the structure to inform text clustering has been shown to provide results that can be comparable to a traditional approach. When considering the lower threshold values, which give the most useful clusters, the hierarchical clustering appears to perform better than traditional approaches, by

focusing on nodes related by the discussion's structure. This can be accomplished using a fraction of the resources necessary for the baseline algorithm, while taking orders of magnitude less time. This hierarchical approach does not completely alleviate the need for the traditional, bottom-up approach, which can find related pieces of text without relying on structure. However, combining the two approaches yields better results than either method taken separately, while requiring significantly fewer resources than the bottom-up approach alone.

The evaluation process to validate the clustering was based on discussion sites where the user community can rate the quality of postings. This user-attributed score constitutes a standard against which the generated clusters can be compared. Relating the similarity measure used in clustering to the user scores proved to be a challenge. There are still unresolved issues concerning the evaluation method, as the usefulness of some of the measurements was undermined by the side-effects of the evaluation procedure. Nonetheless, by relying on a measure of relevance provided by the same community that produced the discussion, this method can be used to evaluate the accuracy of information retrieval in social text at multiple levels of granularity, regardless of the subject matter, without employing human judges to produce an evaluation collection. There is hope that with a sufficiently-large community, the side-effects of this method can be overcome.

6.1 Future Work

The TopicAl system described in this thesis has demonstrated its usefulness in finding related postings in an online discussion. However, the capabilities of the current implementation are limited by performance concerns. Keeping the index for the entire hierarchy in memory limits the size of discussion sites that can be processed. Aside from simply using a computer with more RAM, there is an obvious strategy to mitigate this problem: move the index out of the main memory to a secondary storage, such as a hard drive. This effectively augments the maximum size of the discussion that the system can process by several orders of magnitude. Of course, using a hard drive for storage is orders of magnitude slower than using RAM; some care must be put into the efficiency of input/output

operations. The recent advent of flash-based solid-state drives also provides a way to use secondary storage that is significantly faster than traditional hard disk drives.

The more advanced file-systems in modern operating systems can not only span multiple disks, but also multiple hosts, in a transparent fashion - even over the internet. This would conceivably enable discussions of any size to be indexed. It would also provide the possibility to distribute the work of indexing the pieces of text between different computers. Some work would certainly be needed in order to ensure the integrity of the data in such a distributed system, but the hierarchical structure of discussion already divides the data into separate areas which can be processed independently.

Once the factors limiting performance are dealt with, many more advanced language modeling techniques can be used. As mentioned in Section 3.2.4, the large amounts of memory used by LSA's term-space expansion made it unsuitable. While the social sites used for evaluation were possibly too small to provide the redundancy needed to make LSA's term-space expansion effective, improvements in use of memory would enable larger datasets to be used.

It should also be noted that while the indexing and clustering procedures presented here are premised on the idea of exploiting the hierarchical structure found in social text, there is nothing in the algorithms themselves that is specific to social text. Any collection of text that is organized in a hierarchical fashion could be processed this way. One obvious case is the family of XML formats used for documentation, such as DocBook. Another such case, perhaps more related in spirit to social text, is the growing number of "folksonomies" - community-built knowledge repositories. The best-known of these sites is indubitably Wikipedia, but any sufficiently-large wiki where pages are classified in a coherent hierarchy of categories could be parsed and input into the current TopicAl system.

Another area that warrants further investigations is the different hierarchies that relate pieces of text. The TopicAl system used the containment hierarchy, as it provides context to a piece of text in a way that is easy to understand, while making a hierarchy that has a fixed depth. There are other hierarchies to be found in social text, which may yet be used to provide context for Information

Retrieval. One which has been mentioned previously is the reply structure, where a post is not a descendent of the thread containing it, but of the previous post to which it is replying. This can be extended to the passage level; when a reply quotes portions of a previous message, the passages that come after a quoted portion in the reply can be considered as descendants of the quoted passage in the original message. In many animated online discussions, a single contentious passage may be quoted in dozens of replies. Considering the reply-passages together may be more interesting than simply considering the reply-passage inside the post where it occurs.

Another structure that can be investigated is authorship, where the parent of a given post-node is a node not representing its thread, but its author. This can be extended down to the passage-level, to contrast the passages that are original to a post with those that were quoted from other posts, written by other authors. This could be used to automatically identify sub-communities of authors interested in the same topics, and measure each author's influence over others. There is a rich variety of links in the structure of social text that provide context, which can be used to find related content.

One difficulty of using these hierarchies is that they are not actually hierarchies. A message can quote different previous messages, from different branches in the reply tree. When this happens, the link structure is no longer a tree, but a directed acyclic graph. When dealing with blog posts that can be edited after-the-fact to quote their own comment thread, the directed graph can actually gain cycles. Considering the authorship context, the link structure of a discussion between two authors who repeatedly reply to each other will also yield cycles. The algorithms presented in Chapter 3 assume that a node is present in only one context. It is not clear that the hierarchical clustering makes sense if a node has more than one parent, or more than one term-vector. The depth-first traversal in indexing will certainly not work for graphs with cycles; in order to use these structures as context, a different indexing algorithm will be needed.

If the indexing *were* adapted to deal with generalized directed graphs, then TopicAl could be used to process normal web pages, treating it as the context in which to index the other pages to

which it links. It would be challenging to do this over the entire web, as web pages contain not only text, but navigation elements, advertisements and non-text contents. Focusing on a specific site, such as WikiPedia or another large-scale collaborative resource with many internal links and a strong focus, would provide interesting avenues.

Appendix A

Detailed Results

Figure 18 shows the graph for the results on the MetaOptimize dataset. Figure 19 shows the graph for the results on the Moms4Mom dataset.

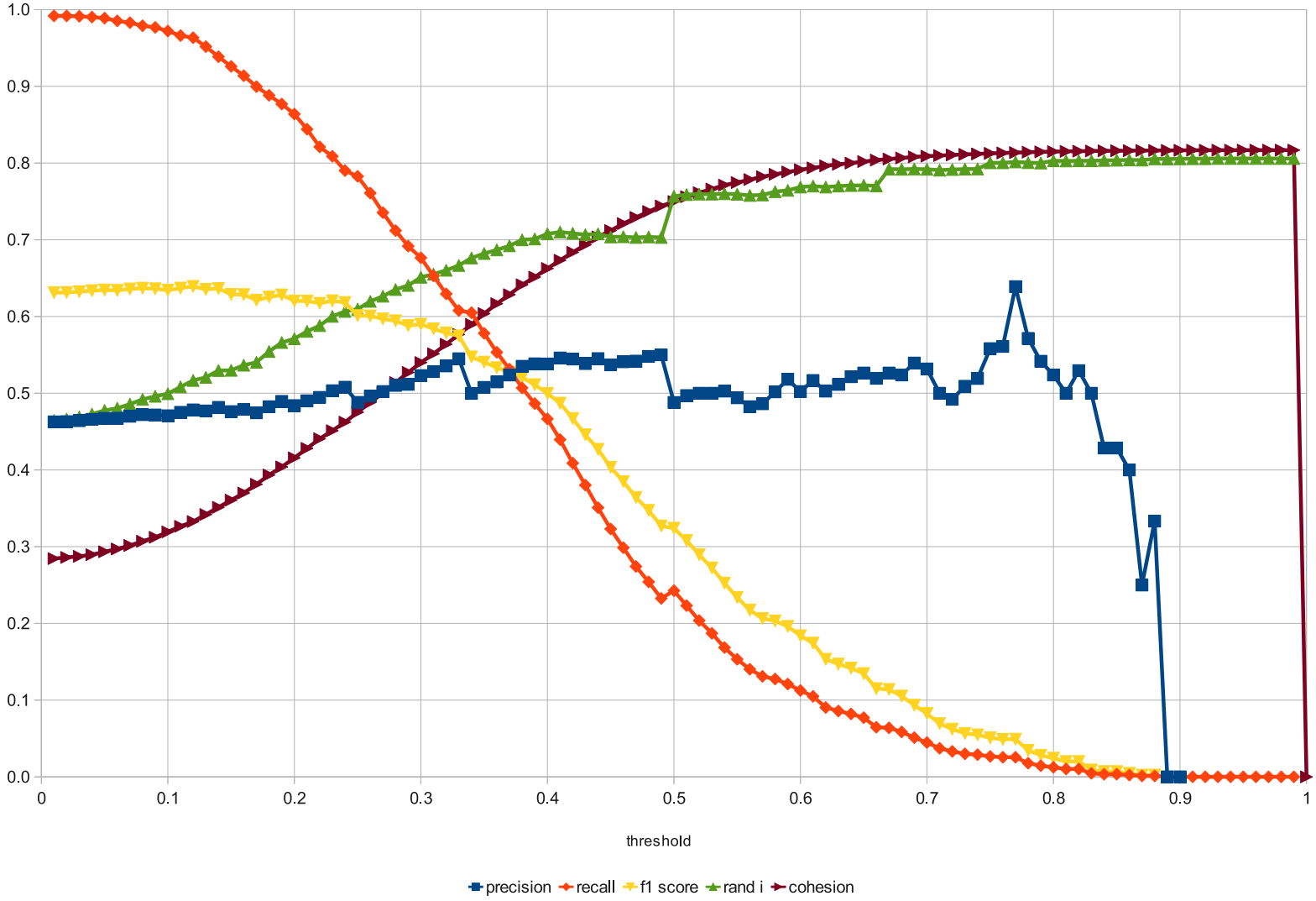


Figure 18: Evaluation of edge clustering for the MetaOptimize dataset: Large Format

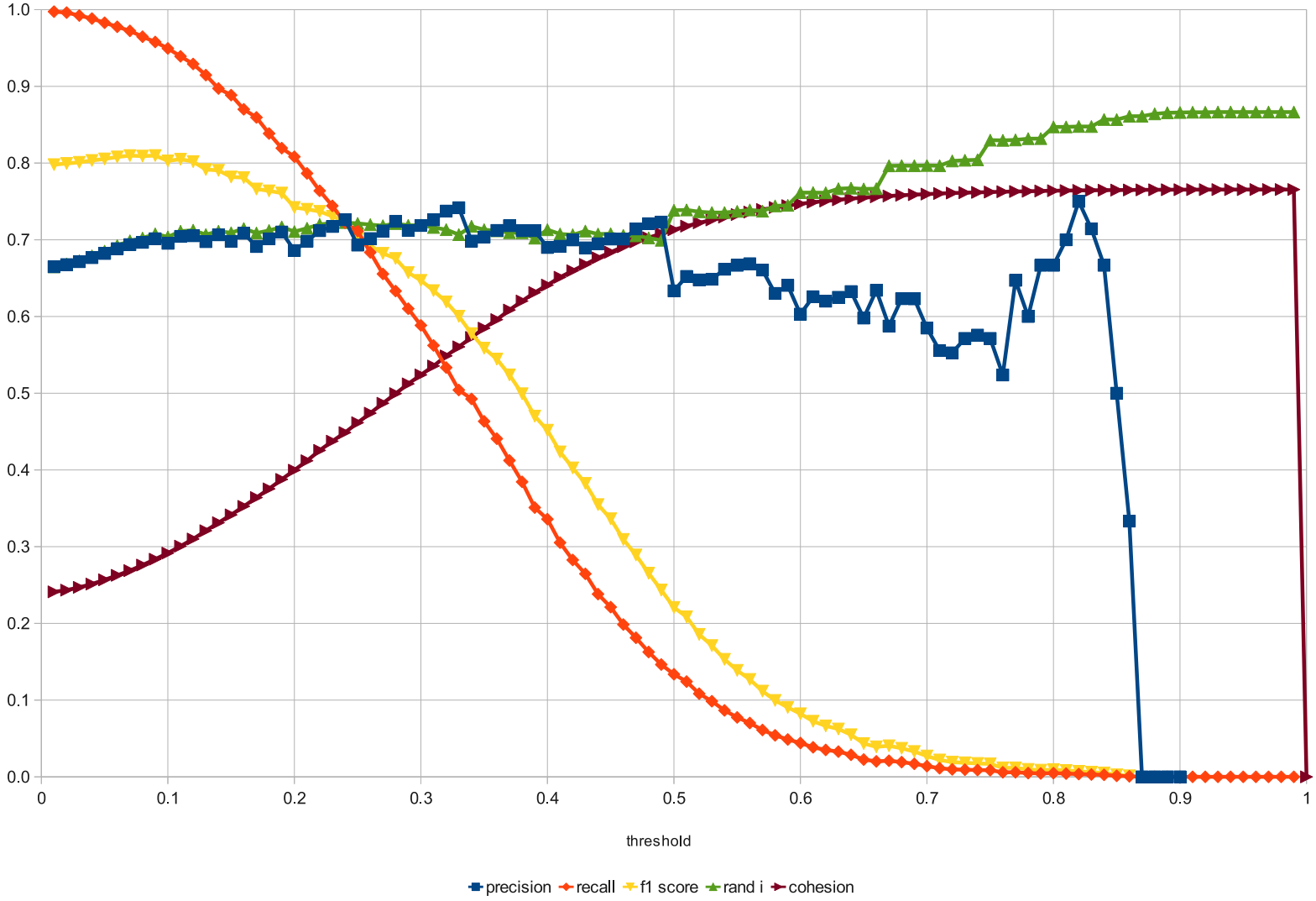


Figure 19: Evaluation of edge clustering for the Moms4Mom dataset: Large Format

Bibliography

[All02] James Allan. *Topic Detection and Tracking: Event-based Information Organization*, chapter 1: Introduction to Topic Detection and Tracking, pages 1–16. Kluwer Academic Publishers, 2002.

[BBFD08] U. Bojrs, J. G. Breslin, A. Finn, and S. Decker. Using the semantic web for linking and reusing data across web 2.0 communities. *Web Semant.*, 6(1):21–28, February 2008.

[BdR07] Jeroen Bulters and Maartin de Rijke. Discovering Weblog Communities. In *International Conference on Weblogs and Social Media*, 2007.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.

[CBBK09] Harr Chen, S.R.K. Branavan, Regina Barzilay, and David R. Karger. Global models of document structure using latent permutations. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 371–379, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[CM09] Kino Coursey and Rada Mihalcea. Topic Identification Using Wikipedia Graph Centrality. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of*

the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers, pages 117–120, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[DB10] Julien Dubuc and Sabine Bergler. Structure-Aware Topic Clustering in Social Media. In *DocEng '10: Proceedings of the 10th ACM Symposium on Document Engineering*, 2010.

[DDF+90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[DLK06] Hoa Trang Dang, Jimmy Lin, and Diane Kelly. Overview of the TREC 2006 Question Answering Track. In *15th Text REtrieval Conference*, 2006.

[GLMF09] Michaela Götz, Jure Leskovec, Mary McGlohon, and Christos Faloutsos. Modeling Blog Dynamics. In *Proceedings of the Third International ICWSM Conference*, 2009.

[GZZ+10] Zhen Guo, Shenghuo Zhu, Zhongfei (Mark) Zhang, Yun Chi, and Yihong Gong. A Topic Model for Linked Documents and Update Rules for Its Estimation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

[HCL07] Qi He, Kuiyu Chang, and Ee-Peng Lim. Analyzing Feature Trajectories for Event Detection. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 207–214, New York, NY, USA, 2007. ACM.

[Hea93] Hearst, Marti A. TextTiling: A Quantitative Approach to Discourse. Technical report, University of California at Berkeley, 1993.

[Joh02] Barbara Johnstone. *Discourse Analysis*, chapter Discourse Structure: Parts and Sequences, page 76. Blackwell Publishing, 2002.

[MOS09] Craig Macdonald, Iadh Ounis, and Ian Soboroff. Overview of the TREC-2009 Blog Track. In *TREC 2009*, 2009.

[MS03] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2003.

[NDW07] Lan Nie, Brian D. Davison, and Baoning Wu. From Whence Does Your Authority Come? Utilizing Community Relevance in Ranking. In *AAAI'07: Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 1421–1426. AAAI Press, 2007.

[Por80] M. F. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[SJ72] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[SMG+07] Bingjun Sun, Prasenjit Mitra, C. Lee Giles, John Yen, and Hongyuan Zha. Topic Segmentation with Shared Topic Detection and Alignment of Multiple Documents. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 199–206, New York, NY, USA, 2007. ACM.

[WBC+10] Hao Wu, Jiajun Bu, Chun Chen, Can Wang, Guang Qiu, Lijun Zhang, and Jianfeng Shen. Modeling Dynamic Multi-Topic Discussions in Online Forums. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

[WCB10] Danny Wyatt, Tanzeem Choudhury, and Jeff Bilmes. Discovering Long Range Properties of Social Networks with Multi-Valued Time-Inhomogeneous Models. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

[WO09] Lidan Wang and Douglas W. Oard. Context-based message expansion for disentanglement of interleaved text conversations. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 200–208, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[YCS09] Tae Yano, William W. Cohen, and Noah A. Smith. Predicting Response to Political Blog Posts with Topic Models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 477–485, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[ZM07] Quiankun Zhao and Prasenjit Mitra. Event Detection and Visualization for Social Text Streams. In *International Conference on Weblogs and Social Media*, 2007.

[ZZW07] Kuo Zhang, Juan Zi, and Li Gang Wu. New Event Detection Based on Indexing-tree and Named Entity. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and development in Information Retrieval*, pages 215–222, New York, NY, USA, 2007. ACM.