Ontology-assisted Methods for the Detection and Clustering of Hierarchical Topics on the Social Web

THESIS

submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Kristian Slabbekoorn

Tokuda Laboratory
Department of Computer Science
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
http://www.cs.titech.ac.jp/

Ontology-assisted Methods for the Detection and Clustering of Hierarchical Topics on the Social Web

Author: Kristian Slabbekoorn
Email: kt.slabbekoorn@gmail.com

Abstract

Discovery of topics of interest on the Social Web can enhance applications like user recommendation and expert finding. We propose methods for ontology-assisted, hierarchical topic clustering on the Social Web. To overcome sparsity issues of LSA-based topic modeling, we map user content to hierarchical, ontological classes, which we manipulate to express topical similarity at a chosen topic scope. Using community detection techniques, we find topic clusters with machine-readable labels without having to pre-define the number of topics. We evaluate against Twitter users and the 20-newsgroups dataset. Compared to the state-of-the-art in topic modeling and document clustering, LDA and k-means, we can discover an appropriate number of topics for the chosen topic scope, outperform both baselines by up to 26.7% on Twitter content, and perform equivalently on newsgroup posts.

Thesis Committee: Prof. T. Tokuda, Prof. M. Saeki, Prof. T. Tokunaga, Prof. K. Gondow, Prof. S. Nishizaki

Contents

List of Figures

Acknowledgements

Before you lies my Doctoral thesis – the final work I've performed at Tokyo Institute of Technology. Over the past three years, I feel that I have learned many technically, academically and socially valuable skills. I am grateful that I have been able to partake in the opportunity to pursue research in Japan, and experience daily life in a culture that is very different from the one I grew up in.

I would like to express my gratitude to my supervisor, Professor Takehiro Tokuda, for making this possible for me, and for his continuous guidance, support and help with my Ph. D. study and research. His guidance has been of great help throughout my time spent researching and writing this thesis. My gratitude also goes out to Assistant Professor Tomoya Noro for his continuous help, advice and comments regarding my research. I would also like to thank remaining members of Tokuda Laboratory for their support.

Secondly, I would like to thank the Monbukagakusho (MEXT) department of the Japanese government for the opportunity to conduct my research in such a wonderful country, and for funding my endeavors with their generous scholarship.

Last but not least, I would like to thank my friends and family for their continued support.

Kristian Slabbekoorn
Tokyo, Japan
February 18, 2016

Chapter 1

Introduction

With the emergence and rapid popularization of various social media, communication is done less frequently in person, and increasingly more often via online social services such as Facebook and Twitter. Recent analyses show that US Web users spend on average 23 hours per week on email, text and social services; 87% of users log in to Facebook at least once per week, and 32% log in to Twitter at least once per week [18]. Inevitably, communities form on these media, bringing together friends and like-minded individuals. Restricting our scope to the Social Web, we can distinguish between connections between individuals based on (1) interpersonal ties, and (2) mutual interests. There are many applications that can benefit from the discovery of these types of connections, and from the clustering of individuals into social or interest-based groups. For example, discovering the structure of connections between people can provide social scientific insights into human behavior or assist user recommendation based on similarity of interests, and expert finding, where we are looking for individuals with specific skills, mentioning unique types of terminology in his content.

The tie-based clustering of individuals (henceforth *users*) by their connections is a classic, interdisciplinary task within the social, natural and computer sciences, commonly known as *community detection*. There has been a significant amount of research into this notion [48][35]. A population is represented as a graph, where each vertex is a user and each edge ties two users together. Edges can be weighted in order to express strong or weak ties. Such a graph can then be clustered into one or more subgraphs (the communities). There exist many algorithms for clustering in graphs, such as modularity optimization [45] or clique percolation [16], which generally boil down to the principle of the *strength of weak ties* [17], where the goal is to identify weak, inter-community connections and remove or ignore them; what remains are the communities of interest. A distinguishing property of such community detection algorithms is that the number of communities, or clusters, do not need to be known beforehand. A downside to tie-based community detection is that content is not considered; we argue that on the modern Social Web, interpersonal ties matter less when we are purely interested in users with similar topics of interest.

When we are more concerned with content, we can turn to the related but distinct tasks of *topic modeling* and *document clustering* [70]. Here, we try to assign similar documents to the same semantic topic. Links between documents are not assumed

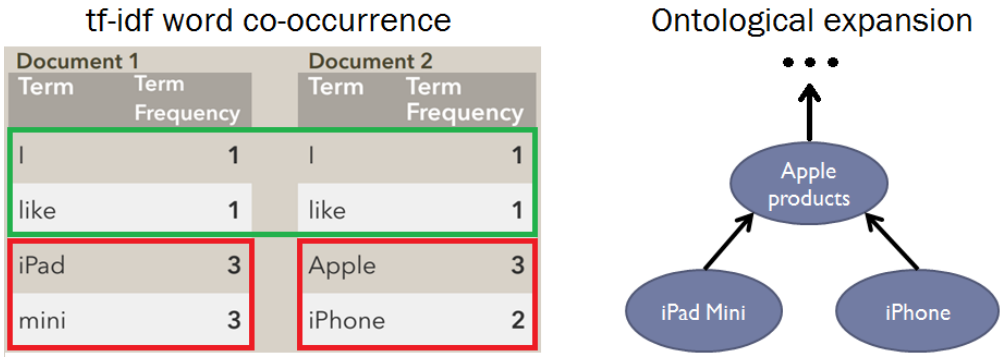

Figure 1.1: A simple comparison of tf-idf word co-occurrence and ontological expansion. tf-idf is unable to capture the similarity between the documents on the left. We can expand both documents to the "Apple products" class when doing ontological expansion.

to be present, and we must therefore analyze the content of the documents in order to determine conceptual similarity. This is most commonly achieved by leveraging word co-occurrence – if words that are relatively unique to the entire collection appear often together in the same subset of documents, we may assume that this subset shares some common topic. Since this topic is not expressed in the documents themselves, it is called *latent*; the most common class of topic modeling algorithms use a technique known as *latent semantic analysis* (LSA) to find these "hidden" topics [14]. Document clustering works in a similar way, where a k-means-based classifier is often used to divide documents into a pre-defined number of topics. We can map the tasks of topic modeling and document clustering onto the Social Web by considering each user's content as a document; the latent topics belonging to groups of users can then be regarded as user "interests". However, a problem with methods that rely on word co-occurrence is that in cases where not much text is available, or text is not necessarily grammatically accurate, these methods tend to perform poorly [24][74]. See figure 1.1 (left) for a simple example of this. Another limitation of most existing document clustering approaches is that the number of topics needs to be known beforehand, which is rarely the case when dealing with an ad hoc collection of users on the Social Web (such as a stream of users resulting from a keyword search on Twitter, for example).

In this thesis, we take a hybrid approach: we use an ontology-assisted topic modeling technique to determine the topical similarity among a population of Social Web users, then use a quasi-clique community detection algorithm to cluster users by shared topics of interest – we assume the number of topics is not known, and infer this from the data.

Initially, we capture topic information by applying ontological expansion: we match named entities discovered in users' text content to an external knowledge base, DBpedia [4], and obtain full class hierarchies for three types of classes (DBpedia [1], YAGO [64], Schema.org [55]) contained in the DBpedia ontology for these entities. See figure 1.1 (right). We use this technique as a foundation for two different methods.

Enhancing user recommendation Our first method focuses on improving a user relation-based approach from previous work [46] for user recommendation on Twitter. Due to the large number of Twitter users, finding users who often provide valuable information related to the topic of interest is difficult. One simple way is to search for tweets by some keywords related to the topic of interest and see how many tweets each user posted. However, using this method we quickly end up with spam bots and other unrelated content; we need to consider user relations as well to derive some minimum level of user influence. Second, it is difficult to choose appropriate keywords to search for related tweets. Suppose that we would like to get tweets related to malware. Some tweets may include names of malware, Internet security companies, technology related to malware, and so on. Since it is almost impossible for us to list all keywords related to the topic of interest, we need to take the deeper semantics of tweets into consideration. To solve these problems, we present a method for user recommendation based on user relations and ontological timeline analysis. The method finds users to follow related to the topic of interest based on provided keywords, then picks up users who continuously post related tweets from the user list. Given some keywords related to the topic of interest, the method first searches for tweets including the keywords, extracts user relations based on tweet behavior among users from the obtained tweets, then ranks users, taking the discovered user relations into account. We then create simple topic ontologies, or taxonomies, of each of the users ranked high after the first phase from tweets collected during different time periods, then determine whether each user continuously post tweets related to the topic of interest. We adopt *ontological timeline analysis* in this phase instead of keyword matching as in the first phase to cope with the second problem mentioned above; that is, we cannot encapsulate the entire topic of interest using a (manually created) list of keywords alone.

We evaluate the approach on four different keywords. The approach generates recommendation rankings of users to follow for each keyword. We manually check the relevance of the tweets of users found to the keywords, then calculate the normalized DCG [29] score for the resulting ranking. We compare our ontological timeline analysis method to rankings based on tweet count, user influence and *TURKEYS* [46], which is a combination of both.

Hierarchical topic clustering Our second method aims to cluster Social Web users by topics of interest. Based on the class hierarchies gathered in the ontological expansion step, we propose a weighting scheme that allows us to determine *trait vectors* (ontological classes weighted with a prominence score) that characterize individual users within a population, and capture users' dominant topics of interest. The scope of topics can be dynamically controlled with a *topic scope* mechanism – it depends on our application whether we want to detect users interested in "sports", or go a step down and distinguish the "American football" interest from the "soccer" interest, for example. This is an important distinction from other topic modeling approaches, which usually require the number of topics to be known in advance, and may not capture hierarchical topics.

In the second part, we construct an undirected *scoped topical similarity* (STS) graph of Social Web users, weighted based on the cosine similarity between users' trait vectors. This graph is clustered into topics of interest shared by groups of users using a

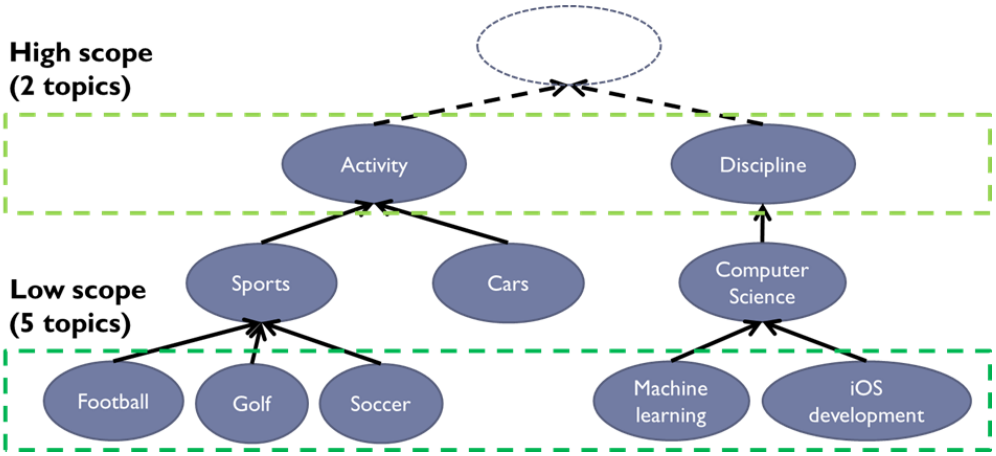

Figure 1.2: Using *scoped topical similarity* clustering, we can manipulate the *topic scope* in order to cluster users at an arbitrary height of the topic hierarchy. In this example, we can cluster by two broad topics (high scope) or five specific topics (low scope).

variation of the clique-based Highly Connected Subgraph (HCS) algorithm [22], which finds communities by recursively splitting a graph along its minimum cut. We modify this algorithm by (1) considering edge weights when determining minimum cuts; (2) loosening constraints on what constitutes "highly-connectedness"; and (3) assigning lowly-connected users that would otherwise be dropped to clusters that match their traits best. Depending on how we set our topic scope, we obtain a high or low slice of the topic hierarchy. See figure 1.2 for an illustration of this. The flexibility of the approach enables two interesting side-effects: using the same algorithms, (1) we can compute cluster-based trait vectors that allow us to label clusters with appropriate tags that are both human-readable and machine-understandable, since they link directly into the Linked Data cloud [7]; and (2) we can cluster a graph divisively into increasingly more specific topics by recursively applying the described method on resulting topic clusters.

We evaluate the approach in three ways. We first investigate the quality of our STS weighting scheme, in terms of nDCG [29] rankings for two different seed users with known interests, and compare the results to baselines such as tf-idf term weighting to show the advantages over word co-occurrence-based methods. We then evaluate scoped topic clustering based on (1) a manually constructed ground truth of 175 Twitter users, distributed over 4 topics and 11 sub-topics; and (2) a dataset commonly used in document clustering research (the 20-newsgroups dataset [53]). The topic clustering results are compared mainly to latent Dirichlet allocation (LDA) [10], the current gold standard for topic modeling; a state-of-the-art Twitter-centric version of LDA, called Twitter-LDA [74]; and k-means clustering [21], a classic document clustering algorithm. Lastly, we evaluate full hierarchical clustering, comparing to hLDA [9] and hierarchical k-means.

1.1 Thesis organization

After a short re-iteration of the main contributions of this work, we begin the main matter of the thesis, which is organized as follows. In chapter 2, we offer some background into the general concepts that we deal with in this thesis, and outline past research into user recommendation, topic modeling, document clustering and community detection, on and off the Social Web. We outline their problems, and how our proposal differs from these works. What follows are the three main components of our approach. First, in chapter 3, we explain how we handle entity recognition and ontological expansion, which forms the basis for the two methods we devised. Chapters 4 and 5 will then explain in detail our proposals for improving user recommendation and hierarchical clustering, respectively. In chapter 6, we discuss our implementation of the hierarchical clustering method and its algorithmic complexity. In chapter 7, we outline our experimental setup and methodology, and evaluate both proposals compared to baselines and the state-of-the-art. Finally, in chapter 8, we conclude the paper with a summary of the results and directions for future work.

1.2 Contributions

We can define three main contributions for this thesis.

1. We propose a method for improving follow relation-based user recommendation on Twitter by including an ontology-based procedure for checking whether user interest into a topic is consistent.

2. We extend contribution (1) into a general, ontology-assisted approach for the hierarchical clustering of Social Web users by topic of interest when the number of topics is not known in advance – we allow the discovery of topic clusters at a chosen *topic scope* that is independent of the underlying data. Using these techniques, we additionally describe:

    a) a method for automatic labeling of clusters with human- and machine-readable topic tags;

    b) a method for divisive clustering of a user graph into a topic hierarchy.

3. Based on contribution (2), we design and implement a system for the real-time clustering, visualization and exploration of streams of Twitter users.

Chapter 2

Background and Related Work

In this chapter, we will introduce the background concepts and previous research that we build upon throughout the thesis. We start by giving brief introductions to some natural language processing notions, such as the term vector space model for representing documents and tf-idf term importance weighting in section 2.1. This is followed by a short review of the classical computer science problems of topic modeling, document clustering and community detection and common implementations, in 2.2. We also cover topics relating to the Semantic Web [6], such as Linked Data [7] and ontologies such as DBpedia [4], in section 2.5. We also introduce the Social Web and its peculiarities, and why we have chosen this environment and Twitter as our main use case, in 2.6.

The background is followed by a short survey of previous work into approaches that relate to our proposed methods, in section 2.7. We touch briefly on their shortcomings, and introduce our proposals to improve them, which we will expand upon in detail in chapters 3, 4 and 5.

2.1 Document processing and representation

Within languages, we can distinguish between *formal* and *natural* languages. Formal languages – such as programming languages – have syntactic and semantic rules that can be formally defined using a formal grammar. This grammar is then used as a reference for a parser to extract concrete meaning from text written in this language, based on which a computer can take deterministic actions. Natural languages – such as English, German, etc. – do not have their semantic rules formally defined, making it far more difficult for a machine to derive meaning from such languages. For example, the simple sentence "John sits behind his computer" is ambiguous. John could be sitting in a chair in front of his computer, which would be the most common semantic meaning of this sentence. However, we cannot rule out the possibility he is not literally sitting *behind* his computer: we need additional context information, perhaps the sentence before or after this sentence, to make a definitive judgement on its meaning.

The processing of natural languages, commonly known as *natural language processing* (NLP), is a challenging task which has been the subject of extensive research [57]. Natural language text can be processed in many ways to make it more consumable for computers, such as part-of-speech tagging, word stemming, stopword re-

moval, and so on. Furthermore, we can distinguish structural NLP from statistical NLP. Structural NLP mainly aims to analyze and understand text based on grammar rules or known structural properties of the language. However, given the ever increasing computational processing capabilities of computers, statistical NLP is a more promising area that has become dominant in recent years. In statistical approaches to NLP, we interpret the semantics of text based on stochastic or probabilistic models derived from the analysis of large existing corpora. Statistical NLP has a wide array of applications, such as machine translation, speech recognition, word disambiguation, and topic segmentation. In this thesis, we will focus only on leveraging statistical NLP methods for topic segmentation purposes. For a more comprehensive overview of statistical NLP, see [37].

2.1.1 Vector space model for documents

If we want to divide documents by topic, we need some way of representing and comparing documents to each other to determine their similarity. A common model for representing documents is the *vector space model*, in which documents are represented as term vectors:

$$d_j = (w_1, w_2, \ldots, w_n) \tag{2.1}$$

See figure 2.1 for a simple representation of n sentences in n-dimensional term space. For practical purposes, only three dimensions are shown – under typical conditions, we have many thousands of dimensions (one for each word in a document) which we cannot represent visually. Each component of the vector corresponds to a word in the document, with the value being a weight that expresses some property about this word in terms of the rest of the document collection. Words are assumed to be independent of each other: ordering of words within a sentence is not considered. This is also called the *bag-of-words* model, since we literally have an unsorted collection of words. This assumption is not necessarily realistic, since word ordering is obviously an important part of any natural language. For some applications, such as topic modeling, this abstraction does not pose problems; for others, such as sentiment analysis, it greatly hurts results. Commonly, *term frequency–inverse document frequency*, or tf-idf, is used for weighting document vectors. tf-idf expresses the importance of a word within a document compared to other documents within the collection:

$$\text{tf-idf}_{t,d,D} = \text{tf}_{t,d} \times \text{idf}_{t,D}, \text{ where } t \in d,\ d \in D \tag{2.2}$$

Here, term frequency is how often a term occurs within each document, while document frequency records the number of documents in which this term occurs, of which we take the inverse, so that terms that occur in many documents get a low weight. By weighting documents and their words using tf-idf, we can express which terms define the documents best. The vector space model for documents is useful for applications within the space of *document clustering*. We introduce one important document clustering algorithm, k-means clustering, in section 2.3.

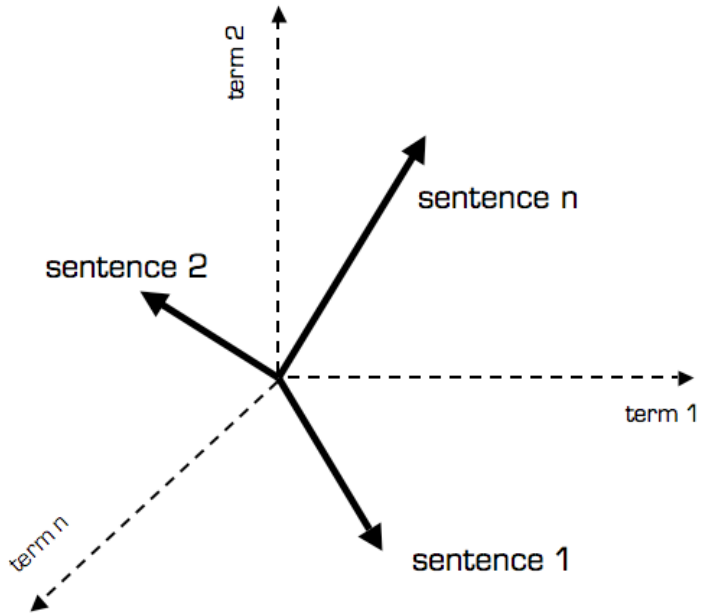

Figure 2.1: A simple example of the vector space model for documents[1]. Three documents with three dimensions each are shown.

2.2 Topic modeling

Within the field of topic modeling, we aim to discover the topics of a document within a collection by analyzing the natural language text in these documents. Topics are generally considered *latent*; that is, topic information does not exist within any of the documents, and needs to be "invented" just by looking at the words. One particular approach that has gotten popular in recent years is *latent semantic analysis* (LSA) [14], particularly probabilistic forms of LSA [25]. Probabilistic LSA aims to model the probability of co-occurrences between words and documents as mixtures of multinomial distributions. Put simply, the documents are analyzed in order to derive a generative probabilistic model that is most likely to have created the underlying collection of documents. *Latent Dirichlet allocation* (LDA) is currently the state-of-the-art implementation of PLSA.

2.2.1 Latent Dirichlet allocation

LDA is a generative model that allows sets of observations to be explained by unobserved groups, or topics, that explain why some parts of the data are similar [10]. A "topic" in LDA is a probability distribution over a fixed vocabulary (for example, a topic *genetics* would likely contain words "gene" and "dna" with high probability). LDA generates document words in the following way. For each document in a collection, (1) a distribution over topics is chosen randomly. Then for each word position in the document, (2) a topic is randomly chosen from the topic distribution, and (3) a word is randomly chosen from the distribution over the topics' vocabulary. Each

[1] Image source: http://blog.christianperone.com/?p=2497

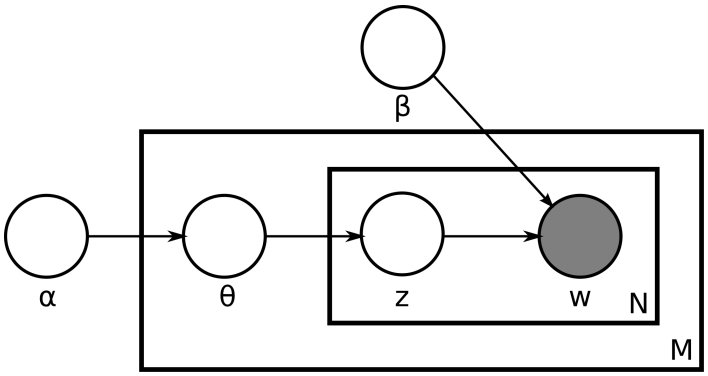

Figure 2.2: Plate notation representing the LDA model and variables.

document gets assigned topics in different proportions, and each topic gets assigned words in different proportions: these are called the *Dirichlet distributions*.

See figure 2.2 for a representation of the LDA model and its variables in the so-called *plate notation*. Here M is the document collection; N is the vocabulary; α is the set of documents $\alpha_{1:M}$, where M is the number of documents and each α_m is a distribution over topics; β is the set of topics $\beta_{1:K}$, where K is the number of topics and each β_k is a distribution over the words; θ denotes the topic proportions for each topic in each document; z contains the topic assignments for each word in each document; and w is each actual word in each document. w is grayed out, to denote that it is *observed*; the rest of the variables are white, to denote they are hidden, or *latent*. This model forms a joint distribution of hidden and observed variables, from which we can generate documents.

This generation is done by sampling from this distribution. There are several ways to sample, but most commonly it is done using *Gibbs sampling* [11]. With Gibbs sampling, we construct a *Markov chain* to derive the posterior of the joint distribution (the topic structure given the observed documents and words). By running the chain for a long time, we can collect samples and approximate the distribution using the collected samples. For a more in depth explanation of LDA and Gibbs sampling, see [8]. We employ LDA with Gibbs sampling as a baseline in the evaluation, using the implementation that is included in the MALLET package [39].

2.3 Document clustering

The task of document clustering is to group related documents together, usually by some shared topic. This can be determined though several means, but most commonly, distance between documents is calculated by some kind of similarity measure, such as the cosine similarity of tf-idf-weighted term vectors, with each document projected into multi-dimensional space. Subsequently, documents that are close together are grouped into clusters. One of the most common document clustering algorithms is k-means, which we will explain briefly.

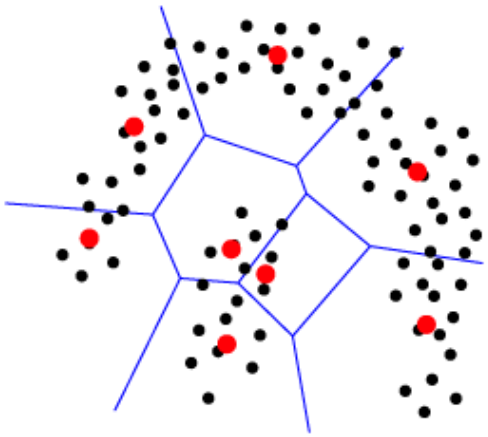

Figure 2.3: A simple example of k-means clustering with $k = 8$. The red dots are the centroids for each cluster.

2.3.1 k-means document clustering

k-means document clustering aims to partition a number of documents into k clusters, in which each document belongs to the cluster with the nearest mean; that is, we calculate k centroid vectors that are composed of the mean of one of k disjoint subsets of document term vectors within the multi-dimensional space; each of these subsets therefore comprises a cluster. For example, figure 2.3 shows the 8-means clustering of points in a two-dimensional space. Again, since document vectors can have many thousands of dimensions, this is not possible to render visually for anything larger than three dimensions.

The most common algorithm to implement k-means document clustering is *Lloyd's algorithm*. We use this algorithm in our work to implement k-means as a baseline to which we evaluate our approach. Refer to [47] for implementation details.

2.4 Community detection

Community detection generally refers to the discovery of community structure in networks or graphs. There are many types of algorithms to find communities. One such type are minimum-cut methods, in which we divide a network in a pre-determined number of parts such that the weight or length of edges between parts are minimized. This type of method has the same weakness as topic modeling and document clustering methods, namely that the number of communities needs to be pre-defined. For our work, however, we are mainly interested in community detection methods that can derive the right number of communities from the structure of the graph.

One popular algorithm that can achieve this is *modularity maximization*, in which the quality of particular divisions of a network into communities is measured [45]. We search through possible divisions of the graph for those that give a particularly high modularity score: it is intractable to do this exhaustively, hence approximation algorithms are usually employed. For our work, we apply a lesser-known algorithm, called the Highly-Connected Subgraph (HCS) algorithm [22]. Using this algorithm,

we find minimum cuts in a graph recursively until all clusters are *highly-connected*; that is, fulfill some minimum edge degree constraint. Unlike regular minimum-cut community detection, we do not have to pre-define the number of parts to cut in. And unlike modularity optimization, it is fast and deterministic, making it more attractive for our purposes.

2.5 The Semantic Web and Ontologies

The Web has been evolving continuously since its inception at CERN in 1991 by Tim Berners-Lee [5]. What started out as a simple means to share academic data by electronically linking together documents has grown to become a major part of our lives. With the advent of Web 2.0, the web gradually evolved into a fully interactive environment where users could not only access information from Web pages, but also publish their own content such as blogs and videos, stay in touch with their friends, chat/talk with people in real-time, do online banking, and so on. What remained missing from this picture, however, was the development of standard ways of representing data, and to give it universally understood semantics to allow even computers to understand what these pieces of data actually mean. Berners-Lee proposed his vision of a fully Semantic Web [6]. The Semantic Web is not intended to be a replacement for the current Web, but rather a new layer superimposed on top of it, injecting a new stack of technologies into the existing Web stack that are based on established Web standards and protocols, such as HTTP and XML.

Since its initial proposition, various initiatives have been set up in attempt to realize the goal of creating a fully Semantic Web. One of the most successful initiatives has been "Linked Open Data" (LOD), which aims to interconnect vast amounts of data from many disciplines using standardized syntactic and semantic conventions, and have this data open for everyone to use and add to. In fact, the term "Semantic Web" is now often used interchangeably with "Linked Open Data." The main goal behind LOD, or simply "Linked Data," is to create a "Web of Data" next to the current "Web of Documents" – one that is interpretable by machines. In Linked Data, related data that was not previously linked is connected to form a new, large-scale, integrated knowledge base. It is this interconnection between datasets that is the most important aspect of Linked Data; by providing links to other datasets, applications may exploit this extra and possibly more precise knowledge. With standardized semantics, this knowledge base can be queried as one would a regular database.

While details of the Semantic Web stack are out of the scope for this thesis (see e.g. [60] for a more extensive introduction), one component that bears explaining are *ontologies* and *taxonomies*, meant to give semantics, i.e. concrete meaning, to data on the Web. We explain them in a little more detail in the next section.

2.5.1 Ontologies and taxonomies

An *ontology*, in the philosophical sense of the word, concerns "what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences[2]." In the context of informa-

[2] `http://en.wikipedia.org/wiki/Ontology`

Table 2.1: Distinguishing characteristics between the World Wide Web and the Social Web.

World Wide Web	**Social Web**
Documents	Users
Formal content	Informal content
Connected through hyperlinks	Connected through interpersonal ties
Static (pages not updated regularly)	Dynamic (timelines updated daily)

tion science, an ontology is the representation of this concept in the form of a structural framework for organizing information. Ontologies are employed in a multitude of disciplines, such as artificial intelligence (AI), knowledge representation and enterprise architecture. A *taxonomy* is very similar to an ontology – in fact, in the context of the Semantic Web, there is no real clear distinction between the two. That said, the word "ontology" tends to be associated with more complex and formal collections of terms, whereas the word "taxonomy" is usually used when strict formalism is not employed.

2.6 The Social Web and Twitter

In this work, we make the explicit distinction between the the regular, World Wide Web (Web of Documents) and the *Social Web* (Web of Users). A short summary of distinguishing properties is shown in table 2.1.

The experiments for this work are performed mainly on Twitter. The reason for this is that data on Twitter is open, with publicly accessible APIs, and in widespread use. It is also an excellent example of an environment where messy, sparse text data comprises the majority of content: Twitter is a so-called *micro-blogging* service, meaning there are strict limitations to the size of individual posts (or "tweets"). Each post can comprise at most 140 characters. It is therefore quite different from a traditional blogging service, mostly consisting of short, informal blurbs rather than well-formed essays or stories. These properties make it challenging for traditional topic modeling and document clustering methods to obtain enough data to make meaningful generalizations about the content of tweets.

2.7 Related work

Due to its many applications, Twitter user recommendation is a commonly studied problem [32]. More generally, topic modeling, (ontology-based) document clustering and classification, and community detection are all classic, multi-disciplinary problems that have been the subject of a significant amount of research, leading to the development of many different methods and algorithms [8][35]. This section will present an overview of past research that relates most to our work, outlining differences compared to our approach, as well as their drawbacks and our proposed improvements.

2.7.1 Twitter user recommendation approaches

Twitter provides its own user recommendation service that recommends users who have mutual followers/friends, but it requires us to already follow some users related to the topic of interest in advance to have the service give an appropriate recommendation result. We want a service that can recommend the right users based on a keyword search.

TwitterRank [69] finds influential users by taking topical similarity among users and link structure (follow relation) into account. Although it returns influential users for each topic cluster, we cannot control how topics are clustered. As a result, we cannot always find an appropriate cluster corresponding to the topic of interest. Twittomender [20] uses lists of followers, friends, and terms in their tweets to find users related to a particular user or query. It calculates similarity between users by representing each user as a weighted term vector. However, it does not take the overall semantics of a user's tweets into consideration, which reduces chance of finding relevant users since each tweet is limited to 140 characters.

Syed et al. [65] proposed a scheme for using Wikipedia as an ontology for describing documents. They demonstrate several algorithms for finding named entities in a document and to leverage the Wikipedia category hierarchy in order to find a set of topics for one or more documents. We exploit the Wikipedia category hierarchy in a similar way, but we use the cleaner YAGO hierarchy and focus on finding topics of interest of a user from tweets rather than domain-specific articles, which makes our problem quite different in nature and significantly more difficult. Nakatsuji et al. [44] used taxonomies to model user interests and how they change over time. Focusing on specific domains of interest, they apply hand-crafted taxonomies to classify entities. Item recommendations are made for an active user based on the similarity of (super) topics with target users. Our approach is not aimed at item recommendation but user recommendation. We model a user's entire recent tweet history to recommend users who consistently post tweets about a certain topic.

2.7.2 Topic modeling approaches

In 2003, Blei et. al. [10] revolutionized the topic modeling field with the introduction of a pLSA-based generative probabilistic topic modeling technique called latent Dirichlet allocation (LDA). Since then, a great number of variations of and additions to LDA have been developed that address specific use cases. We briefly discuss two Twitter-centric approaches and a hierarchical approach.

Hoang et. al. [24] conducted an empirical study on different ways of performing topic modeling on Twitter tweets using the original LDA model and the author-topic model [56]. They find that topics learned from documents formed by aggregating tweets posted by the same users may help to significantly improve some user profiling tasks. The work by Zhao et. al. [74] proposes the TwitterLDA topic model for microblogging data. TwitterLDA is another variant of LDA, *hLDA*, which assumes there is only one common topic for all words in each tweet, and that each word in a tweet is generated from either a background topic or the user's perceived topic. The authors demonstrate a significant improvement over standard LDA on Twitter data. In [9], Blei et. al. introduce a hierarchical version of LDA, using a nested Chinese restaurant pro-

cess to generate infinitely deep trees in order to cluster documents at multiple levels of topic abstraction. We include these latter two works in our evaluation of a Twitter-based dataset. The main issue with all approaches that incorporate a variation of LDA is the inherent reliance on word co-occurrence between users or documents, and the need to pre-define the number of topics (number of hierarchical levels for hLDA) that exist in the data. This is especially problematic on the Social Web, where text content is sparse and noisy, leading to insufficient term overlap to make accurate predictions about their latent semantics, and where topics cannot be concretely pre-defined. We address the first concern by mapping text content to a subset of the roughly 300,000 hierarchical classes of an external ontology; all subsequent processing is done on these classes. This dimensionality reduction helps minimize sparsity and noise, while leaving enough classes to be able to model unique user interests. Secondly, we do not need to pre-define any number of topics: given a topic scope, we infer this from the data using clustering techniques.

2.7.3 Document clustering and classification approaches

Approaches that leverage some form of document classification involve the a priori definition of a set of possible topics or categories, with the subsequent application of NLP and machine learning techniques to allocate each document or user to one of these categories. For example, Zhao et. al. [75] propose a topic-oriented community detection approach which combines both social objects clustering and link analysis, in order to identify more meaningful topical communities. They classify documents according to a number of pre-defined topics, taking into account interpersonal connections as well as content similarity. Tsur et. al. [67] present an algorithm for classifying Twitter tweets into pre-defined topics based on hashtags. While we agree with these works that topic-oriented communities are more meaningful than tie-oriented ones, these types of techniques are classification methods that pre-define a set of around eight to ten flat topics to assign users or documents to, which limits usefulness in scenarios where we are interested in specialized topics or hierarchical topics. Our approach is not one of classification, but of clustering: we construct topics dynamically in terms of classes that characterize groups of topically similar users.

An advanced streaming, machine learning-based method for matching individual tweets to a pre-defined ontology of 300 classes with high precision was introduced by Twitter's user modeling team [72], and is currently in active use as a part of Twitter Analytics[3]. Although this work focuses on individual tweet topic modeling, it offers some insights into extending this to user interest modeling based on tf-idf-style weighting of the discovered tweet topics and additional user behavior (retweet, reply, favorite, etc): this is similar to how we model user interests, however we aim to discover much more fine-grained interests by leveraging and defining them as a subset of hundreds of thousands of classes instead. This allows us explicit control over the scope of interests and to cluster a larger collection of users by their shared interests. Furthermore, Twitter Analytics only offers a general overview of one's own follower interests (see figure 2.4); we instead develop an application that can visualize *any* (public) individual user and their interests, clustered by users who share their interests.

[3] `https://analytics.twitter.com`

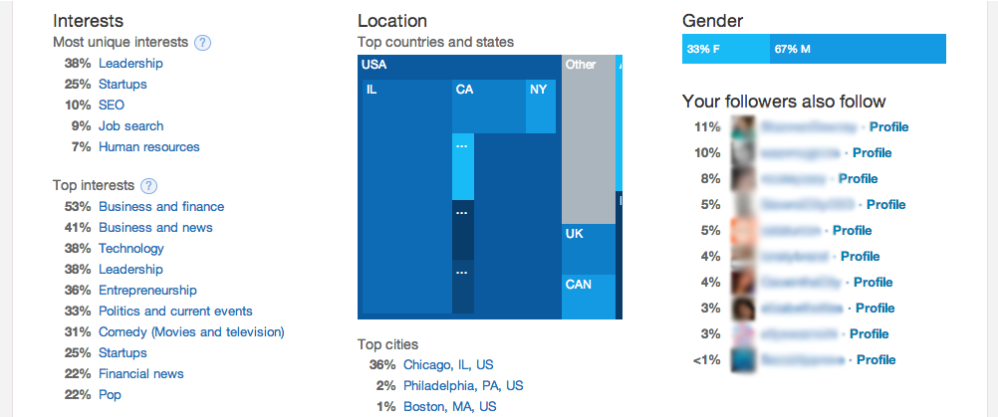

Figure 2.4: A (partial) screenshot of the Twitter Analytics dashboard, showing an aggregate view of follower interests.

2.7.4 (Ontology-assisted) Social Web approaches

The emergence of the Social Web has given rise to a substantial body of research exploring differences between Social Web content and standard content, and developing specialized methods and variations on common techniques to deal with these differences. In [54], for example, the authors rebuild the NLP pipeline from start to end with microblogs in mind, reporting double the F-measure compared to the state-of-the-art. In [36], a graph-based method is introduced that conducts named entity normalization and recognition simultaneously, looking for pairs of words that share the same lemma and judging whether these words mention the same entity in a stochastic way. A general exploration into the difficulties of standard NLP on the Social Web is given in [15]. We will limit ourselves to discussing works that specifically combine the Social Web and semantic expansion to external ontologies (most commonly Wikipedia) into their apporach.

In [41], Michelson et. al. present results on discovering Twitter users' topics of interest by examining entities mentioned in their tweets, and deriving a topic profile based on Wikipedia categories. Their findings corroborate our own results that using an external concept hierarchy can be a good indicator for categorizing general user interests, but we argue that Wikipedia categories are too disorganized to be useful, and choose to rely mostly on the more robust YAGO [64] taxonomy instead. In [2], Abel et. al. incorporate semantic expansion to an external ontology into their Twitter user modeling process for improved personalized news recommendations, showing significant improvements over simpler approaches such as relying on hashtags. Similarities between the above two works and ours end at the way user topic profiles are represented, however; they do not incorporate clustering of users using these profiles. We are not aware of research on the clustering of Social Web users specifically using ontological classes. However, significant research has been conducted into the clustering of regular documents with the assistance of ontologies. The most prominent work is by Hotho et. al. [26], who propose using WordNet [42] concepts to assist in document clustering, and show improved results. They disambiguate terms to concepts in a simple way and apply a k-means type clustering based on concept frequencies into a pre-defined

number of topics. Unlike our approach, they do not exploit a tf-idf-style weighting of concepts and rely only on concept frequency to determine similarity, require topics to be pre-defined, and do not support hierarchical topics. We also argue that WordNet does not provide sufficient granularity for concepts: only relatively high-level topics can be obtained, whereas we allow even highly specific topics to be defined by considering not only classes, but at the lowest level also the Wikipedia pages themselves.

We have previously introduced a method in which we enhanced a Twitter recommendation system based on user follow relations with a post-processing step that leveraged an external ontology [62]. We reported positive results, but the approach is not applicable for comparing different users to find if they are similar. Our new work described in this paper essentially extends this previous post-processing step into a general approach for clustering a population of users by interest.

2.7.5 Community detection-based approaches

To our knowledge, there has been little to no substantial research into hybrid topic modeling and graph-based community detection approaches for clustering. A notable exception is Lancichinetti et. al. [33], who develop a community detection-based method for document clustering applied on a graph expressing tf-idf-based similarities between documents, and show large improvements over the LDA gold standard in terms of accuracy and reproducibility. Our work can be regarded as a further exploration of this approach, substituting an ontology-based method for the topic modeling part – which we argue should work better with sparse content – and additionally allowing hierarchical clustering as well as automatic topic labeling.

Chapter 3

Entity Recognition and Ontological Expansion

As described in the contributions section (1.2), our research consists of three main contributions: two methods and one implementation. The structure of the components of the two methods we devised is shown in figure 3.1. A common component for both methods is the recognition of entities in tweets, and ontological expansion to map users to classes, which we will define first in this chapter. First, in section 3.1, we will detail how we detect entities in free text on the Social Web. We subsequently explain our approach to ontological expansion in section 3.2.

3.1 Detecting named entities

The first step is to ontologically expand each user's posts by applying *named entity recognition* (NER) to link entities to ontology classes. We do this in order to mitigate the sparsity of raw text data in a Social Web context. In order to keep the scope of our research within bounds, we have decided not to devise our own approach for NER. Instead, we opt to use DBpedia Spotlight [40][12], a proven off-the-shelf entity recognizer that detects named entities in text and generates links to DBpedia [4]. We briefly explain DBpedia in section 3.1.1, introduce DBpedia Spotlight in section 3.1.2, followed by an explanation of how we apply Spotlight to our Social Web data, in section 3.1.3. Finally, sections 3.1.4 and 3.1.5 touch on two ways in which we adapt the source text and tune the entity recognizer for better performance.

3.1.1 DBpedia

For this work we have chosen to focus on DBpedia, a centerpiece of the Linked Data cloud of the Semantic Web [4]. DBpedia can be regarded as the Linked Data equivalent of Wikipedia – in fact, all information contained in DBpedia has been automatically extracted from Wikipedia. It is therefore a type of mirror for Wikipedia – it does not produce data of its own, but simply structures data extracted from Wikipedia pages. This structuring is done in RDF[1], according to an ontological schema of classes and

[1] http://www.w3.org/RDF/

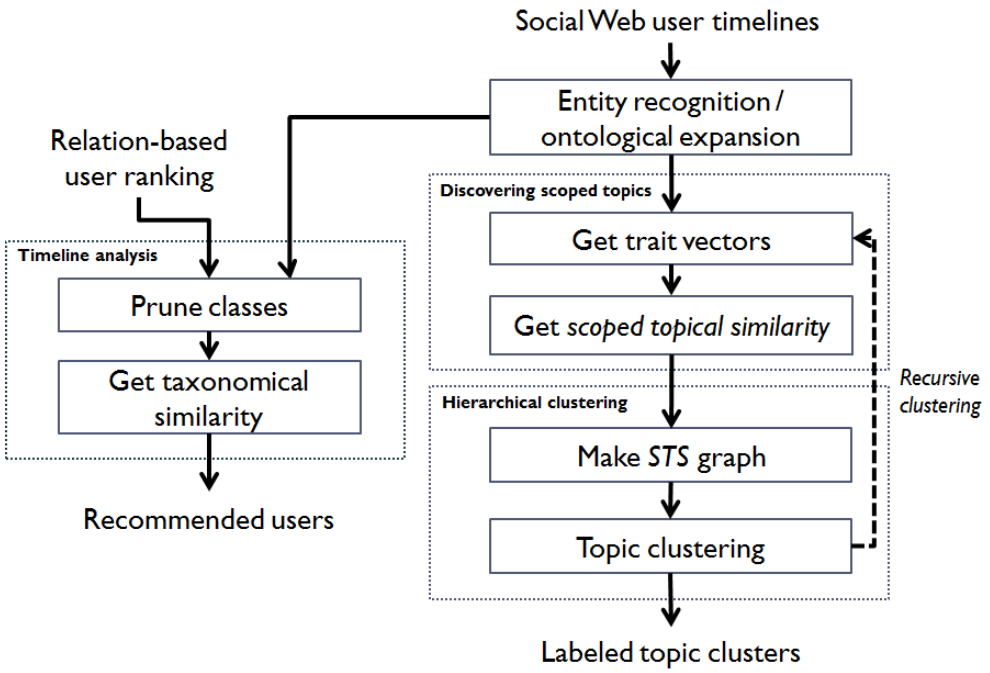

Figure 3.1: The structure of our thesis. First, we apply entity recognition and ontological expansion. From here, we develop two methods: timeline analysis to improve user recommendation (left, see chapter 4), and hierarchical clustering into topics (right, see chapter 5).

properties, which has been taken and adapted from infoboxes on Wikipedia. Measures have been taken to clean this data by merging instances where there are multiple infoboxes for the same class, or different property names for the same property.

For each Wikipedia page, there is a resource in DBpedia. This resource does not contain the full text of the page, but only structured information. Aside from infobox classes and properties, it contains an abstract for the resource (the first paragraph of the corresponding page), disambiguation pages and redirects that point to this resource, and external links to other large datasets on the Semantic Web.

3.1.2 DBpedia Spotlight

As a starting point for the entity recognition and ontological expansion we take DBpedia Spotlight [40], a powerful, off-the-shelf tool for matching textual resources to DBpedia. DBpedia Spotlight has been shown to be able to compete with established annotation systems while remaining largely configurable [40].

Spotlight works by first finding *surface forms* in the text that could be mentions of DBpedia resources (the "spotting" function), then disambiguating these surface forms to link to the right DBpedia resources based on context similarity measures (the "disambiguation" function). This is depicted in the left side of figure 3.2. Its results can be directed towards high precision or high recall by setting two parameters: a "support" threshold for minimum popularity of the associated Wikipedia page (i.e. the number

of inlinks to this page from other Wikipedia pages) and a "confidence" threshold for minimum similarity between source text and context associated with DBpedia surface forms. The confidence of a match also takes into account the difference in similarity score with the second-most similar concept: the confidence score gets an increasing downward bias as this difference gets smaller, so that ambiguous resources are assigned a lower confidence value (i.e. we cannot be fully sure that this resource is the correct one since there are some very similar resources to this one) and unambiguous resources get higher confidence (i.e. we can be pretty sure that the resource we end up with is correct since there is no other resource like it). The confidence score has been normalized to a range of 0..1.

As mentioned, the tool performs disambiguation of possible resource mentions based on surface forms that may point to these resources. These surface forms are mined from Wikipedia in a pre-processing stage. For each resource, the following surface forms and context information are collected:

- the resource labels, i.e. the Wikipedia page titles, with the entire page as context;
- redirect and disambiguation labels, i.e. titles of pages that redirect or disambiguate to the resource;
- anchor texts of hyperlinks pointing to this resource from other Wikipedia pages, with the sentence the link occurs in as context;
- anchor texts of hyperlinks pointing to this resource from the Web, with the sentence where the link occurs in as context information.

The various types of context gathered for each resource are combined and stored along with the resource URI and its associated surface forms. When trying to disambiguate a surface form to a resource, all resources associated with this particular surface form are gathered and ranked according to the similarity between the context of the surface form (if available) and the context belonging to each resource.

3.1.3 Named entities on the Social Web

For our work we do not employ the full range of Spotlight's features – we simply take the best candidate for each resource found if it meets the confidence threshold. We aim to derive per-user topic profiles, meaning we map whole users to some hierarchical distribution of topics. To this end, we take a significant portion of a user's text content – for our Twitter use case, we take the most recent 500 tweets – and apply DBpedia Spotlight on this content. We make the observation that the more input data that we have, the better entity recognition imperfection will be mitigated: even if recognition is only roughly 85% accurate [12] (although this number will likely be lower in a Social Web context), by the law of large numbers, entities should converge to the most dominantly present topics in spite of a significant ratio of error. The text content of the tweets is pre-filtered in a basic way: we remove user mentions, urls, and the hashbangs from hashtags, as well as certain very common terms that are noise more often than not (such as "Twitter", which appears often in tweets but is redundant since an interest in Twitter is obvious for Twitter users, or netslang such as "lol"). Since Spotlight already

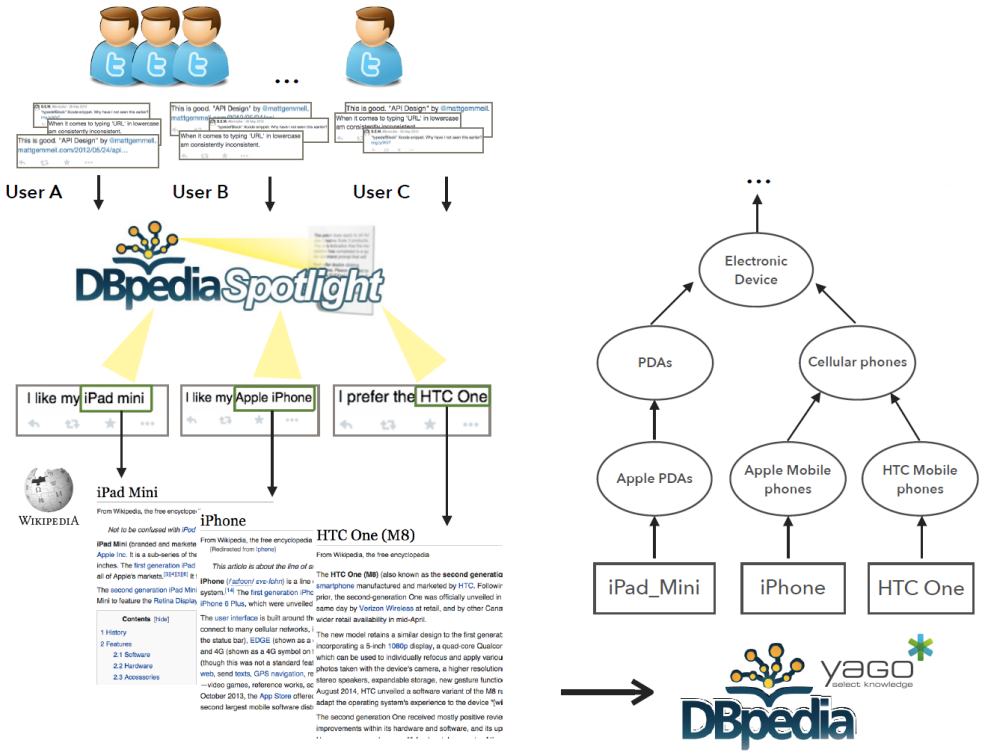

Figure 3.2: Tweets are collected from the timelines of a collection of users U. DBpedia Spotlight is then applied on the text in these tweets to find named entities, which are linked to DBpedia entities and (a simplified representation of) their class hierarchy.

applies tokenization and other common NLP techniques to its input, any further text pre-processing is left to the entity recognizer.

More formally, given a collection of users U, for each user $u_i \in U$, we apply Spotlight on all posts $\{p_1, p_2, ..., p_n\} \in P_i$ to obtain a list of unique DBpedia entities $\{e_1, e_2, ..., e_m\} \in E_i$. For each entity, we keep track of its number of occurrence within P_i. In the simple example that is illustrated in figure 3.2, we have $E_1 = \{(\texttt{iPad\_Mini}, 1)\}$, $E_2 = \{(\texttt{iPhone}, 1)\}$ and $E_3 = \{(\texttt{HTC\_One}, 1)\}$.

It is important to note that this step is a dimensionality reduction from words W to entities E: while each post $p \in P$ may contain up to 140 characters (in the case of Twitter), with an empirically observed average of around 15 words for English tweets [50], typically only one or two entities will be discovered in each p. This reduces the dimensionality and therefore sparsity of the problem by roughly an order of magnitude. Furthermore, given that Spotlight can annotate a full news article in approximately one second on a commodity PC [12], entity recognition has no significant impact on the performance of the method as a whole. In our environment, we apply Spotlight in parallel across six process threads on a modern PC[2], and find that we can annotate 175 users with 500 tweets each (87,500 tweets in total) in around 67 seconds.

[2]Experiments were performed on an Intel Core i7 4790k @ 4Ghz with 16 GB of RAM.

3.1.4 Concatenation window size

Spotlight leverages word co-occurrence between source text and text from candidate Wikipedia entries for disambiguation, so supplying extra context information can significantly improve the accuracy of the entity recognition. For microblogs such as Twitter, multiple consecutive tweets are often about the same topic (consider conversations between two or more users, or a message that does not fit in 140 characters), resulting in better accuracy for the recognizer if we have more of the same type of content to consider. We need to determine a *concatenation window size* for the number of individual posts $p \in P_i$ to concatenate before sending them to the recognizer. We set up a simple experiment in order to determine what window size values give best results; see section 7.3.1 of the evaluation for details.

3.1.5 Spotlight parameter tuning

DBpedia Spotlight has a number of parameters that can be adjusted for different results. Most significantly, we can set *support* and *confidence* values. Setting these parameters is essentially deciding on a trade-off between precision and recall. With the support parameter we can set a minimum number of in-links that we require a Wikipedia page to have, and with the confidence parameter we can filter out matches that have insufficient similarity between source and target texts. For our approach, we are interested in even the very specialized subjects that may have only a single in-link on Wikipedia, but also want to exclude orphaned pages that may have been abandoned. By this reasoning, we fix the support parameter to a value of 1. It is not clear however which confidence parameter will yield the best results, so we perform experiments to find a good value empirically. See section 7.3.1 of the evaluation for details.

3.2 Ontological expansion

In this section, we explain how we apply ontological expansion to gather class information about the entities found by DBpedia Spotlight in free text, and represent user topics in terms of these classes. Recall that Spotlight links matches to DBpedia resources, and that DBpedia is essentially a structured version of Wikipedia that includes various ontologies and taxonomies, automatically or manually extracted from the underlying Wikipedia data and external sources. For our purposes, there are three ontologies that are useful in particular, which we incorporate into our approach:

- **DBpedia ontology classes** [1]: these have been derived mainly from the info boxes on Wikipedia. They form a hierarchy of 685 classes.
- **Schema.org classes** [55]: these are originally semantic classes to mark up HTML pages in ways recognized by major search providers. They have been included for DBpedia resources through a mapping from the DBpedia ontology [23].
- **YAGO classes** [64]: this ontology is a cleaned version of the Wikipedia category hierarchy, with top-level classes represented by terms from WordNet [42]. It contains roughly 300,000 classes, and is the taxonomy we rely on the most, as it contains both broad and very specific types of classes. The accuracy of the

Table 3.1: An overview of the properties of the different types of ontological classes that we collect.

Class type	Source	Creation method	Size	Hier. type	Hierarchy depth
DBpedia classes	Infobox types	User-defined	685	Strict	Shallow (7 levels)
Schema.org classes	HTML microdata	User-defined	673	Strict	Shallow (6 levels)
YAGO classes	Categories + WordNet	Auto-generated	>300,000	Near-strict	Deep (>10 levels)

derivation employed for the mapping from categories + WordNet is claimed to be around 95%. A simplified excerpt of this ontology can be seen on the right-hand side of figure 3.2, showing the beginning of the YAGO hierarchy for some example entities.

All three of these ontologies form tree-like hierarchies, but none are strict trees – that is, they are directed acyclic graphs, which means some classes may have multiple parents. However, they are all hierarchical in the sense that concepts get broader the closer one gets to the root, and narrower the closer one gets to the leaves. We have summarized the properties of each type of class information in table 3.1.

Recall that we gathered a collection of entities E_i for each user u_i in the entity recognition step. Now, for each entity $e_j \in E_i$ we collect hierarchical class information. This means that we obtain the DBpedia, Schema.org and YAGO classes directly assigned to this entity, as well as the parent class(es) of these classes, and their parent classes, and so on, until the root of the hierarchy is reached. We keep count of the number of occurrence of each unique class as we did for entities. We end up with a bag of unique classes $\{c_1, c_2, ..., c_n\} \in C_i$ per user u_i, where each c is a tuple consisting of the class name along with its number of occurrence. For our example in figure 3.2, we get:

$C_1 = \{(\texttt{Apple\_PDAs}, 1), (\texttt{PDAs}, 1), (\texttt{Electronic\_Device}, 1)\}$
$C_2 = \{(\texttt{Apple\_Mobile\_phones}, 1), (\texttt{Cellular\_phones}, 1), (\texttt{Electronic\_Device}, 1)\}$
$C_3 = \{(\texttt{HTC\_Mobile\_phones}, 1), (\texttt{Cellular\_phones}, 1), (\texttt{Electronic\_Device}, 1)\}$

3.2.1 Including entities as classes

Wikipedia, and by extension DBpedia, are crowd-sourced data sources, and are therefore noisy: not all entities have classes associated to them, assigned classes may be incorrect, or the classes may not be sufficiently descriptive. To mitigate this, aside from including the three different ontologies described for extra redundancy, we include the entities themselves as classes as well. This means that we count not only class occurrences, but also entity occurrences. From here on, we will consider $\text{cf}_{i,c}$ to

be the *class frequency* map consisting of the union between the entities E_i and classes C_i:

$$\text{cf}_{i,c} = E_i \cup C_i = \text{ the number of times entity } e \text{ or class } c \text{ occurs for user } u_i. \quad (3.1)$$

These class frequency maps associated to users are the fundamental building blocks for two novel methods for the representation of topics in terms of classes: *class pruning* and *cf-iuf weighting*. We will explain these methods in the next two sections.

3.2.2 Topic representation: class pruning

We can trim the set of classes to obtain an accurate representation of a user's topic(s) of interest. Since entity recognition on informal tweets is far from perfect, we need to discover and remove any classes that have been linked to mismatched entities. Second, since we collect classes up to the root of the hierarchy, we need to filter out classes that are too generic and cover too broad of a topic.

Mismatch removal We rely on the law of large numbers to remove mismatched entities: as long as the accuracy rate of our entity recognizer is not so low as to be equivalent to a random selection of DBpedia resources for each possible entity mention in the text, then classes related to the topic being talked about by the user will statistically have the highest number of occurrence in the long run. Since DBpedia Spotlight has been shown to be able to attain an accuracy rate that is far better than a random selection [40], we assume that, given enough tweets of a user to process, classes related to his/her interests will appear the most in the collection.

We can therefore choose to remove classes with a particularly low number of occurrence, as these are the most likely to be mismatches. Since we cannot be sure exactly how many classes we collect (for example, since we collect classes up to the root of the hierarchy, more classes are collected for specific topics than for generic ones), we define a threshold for removal as a percentage $t\%$ of the total number of occurrence for all classes gathered for a user. Any class with a number of occurrence lower than this percentage will be removed.

Filter generic classes We also need a way to remove classes that are too broad and cover too many classes. We apply the algorithm described in our earlier work [61]. That is, we filter out all super-classes where the sum of the numbers of occurrence of their *direct* sub-classes is greater than $p\%$ of the number of occurrence of the super-class. In other words, for every class $Super$, if

$$\sum_{i=1}^{n} occs(Sub_i) > \frac{p}{100} occs(Super), \quad (3.2)$$

where n is the number of direct sub-classes Sub_i of $Super$, and function $occs$ counts the number of occurrence of each class, then $Super$ is removed from the collection.

See figure 3.3 for an illustrative example of pruning a taxonomy with a dominant topic of "musical artists". In this figure, class occurrence numbers are shown in

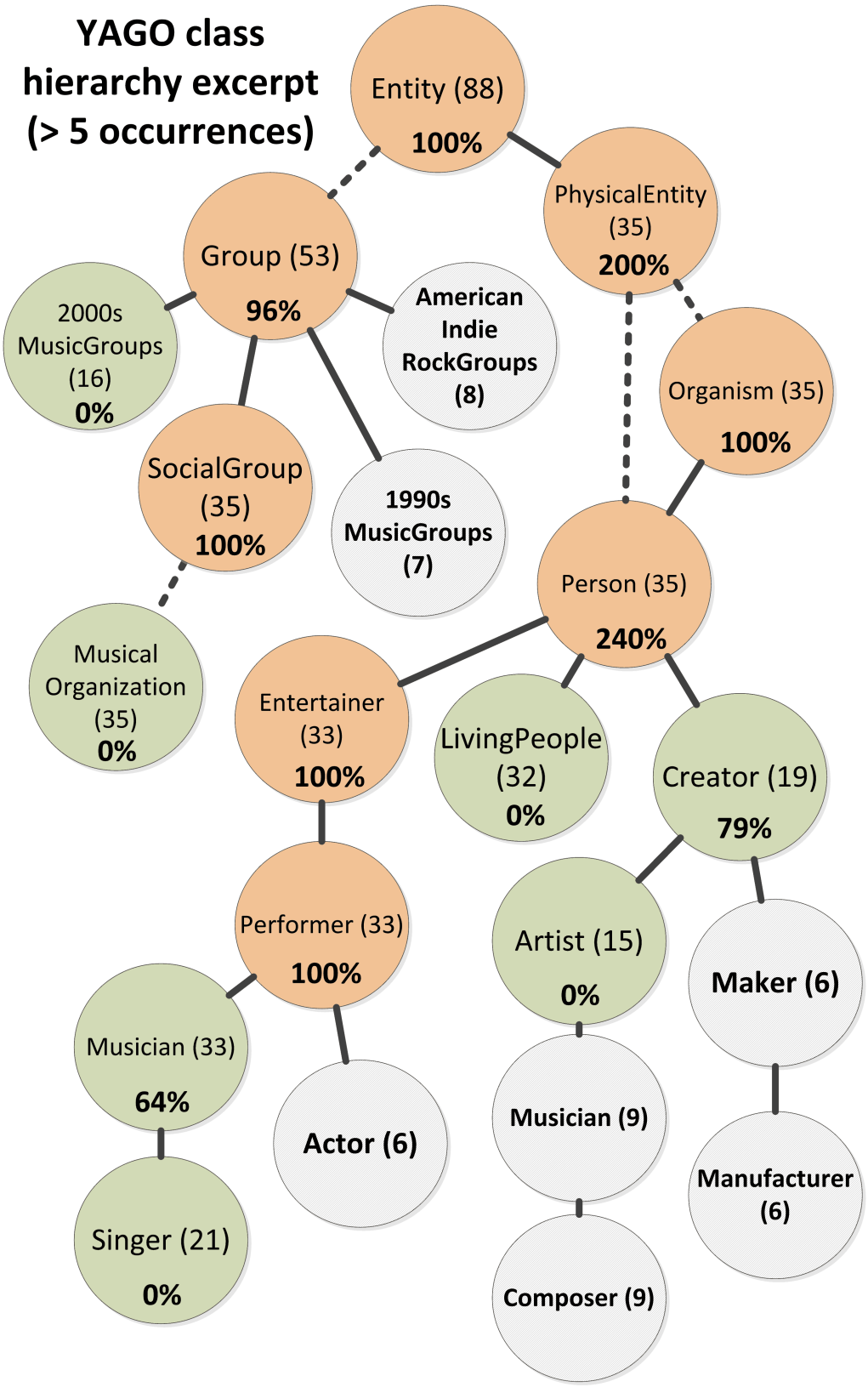

Figure 3.3: Example excerpt of a YAGO class taxonomy after filtering, with a mismatch removal cutoff of $t = 1\%$ (14 occurrences), and $p = 80\%$.

parentheses. Dotted lines abstract some unimportant intermediate classes, and classes occurring 5 times or less are also not shown. The percentages are the proportions of direct sub-class occurrences to this class' occurrences. Classes in light gray are discarded during mismatch removal; classes marked red are discarded during generic class filtering. The white classes represent the final topic. As can be seen, most generic and unrelated or too specific classes are successfully removed. However, there is one particular YAGO class, *LivingPeople*, that is often assigned directly to a person entity, making it not possible to filter out using this algorithm. Since this is an exceptional class, and is too generic for our purposes, we exclude this class from the result manually.

3.2.3 Topic representation: cf-iuf weighting

Note that given the ontologies' hierarchical properties, highly specialized classes will tend to get lower cf values, and more general classes will get higher cf values. So while the entities and the classes of the three different ontologies have been combined, flattened and assumed independent from each other, their hierarchical properties remain intact. The most common classes are the root classes for each ontology: for example, the root class of the DBpedia ontology is *Thing*, which means that *Thing* will occur as often as the total number of entities found in all posts P_i of u_i.

We introduce a weighting strategy for classes that is based on tf-idf, a common strategy used in document processing [63]. On top of the class frequency map $\text{cf}_{i,c}$ described in section 3.2.1, we now additionally calculate a *user frequency* map uf that records for every unique c found how many users in U have at least one instance of each c:

$$\text{uf}_c = \text{ the number of users that have at least one instance of } c. \tag{3.3}$$

We can see that the higher the uf_c for some c, the more common a class is among the collection of users – and the more common a class, the less this class contributes to the uniqueness of a particular user that has this class. Hence, as with traditional inverse document frequency, we take the inverse of this measure, so that more common classes get a lower weight:

$$\text{iuf}_c = \log \frac{|U|}{\text{uf}_c} \tag{3.4}$$

Combining equations 3.1 and 3.4 leads us to our *cf-iuf* weighting strategy for each u_i:

$$\text{cf-iuf}_{i,c} = \text{cf}_{i,c} \times \text{iuf}_c, \text{ where } c \in C_i. \tag{3.5}$$

We can calculate a cf-iuf weight for each unique class that occurs for each user. It follows that, using this equation, classes that occur many times for a select group of users get assigned high weights, while classes that are relatively common, occurring for a large portion or all of the users, get assigned low weights.

Chapter 4

Improving Relation-based User Recommendation on Twitter with Ontological Timeline Analysis

The representations of users in terms of ontological classes defined in the previous chapter are now used as the foundation for two specific methods. In the first, which we will explain in this chapter, we use our ontological expansion technique to improve a follow relation-based method for Twitter user recommendation through analysis of individual user timelines. Although our second and main contribution is designed to be useful in a more general sense, this method is explicitly Twitter-based.

We focus our attention on how we can use our method for ontological expansion to improve user recommendation based on user relations, in order to find influential users that post about their topic of interest consistently. We first give a short summary of how a ranking of users by follow relation for a certain Twitter keyword search is obtained. This is followed by a description of how we further improve this ranking based on ontological expansion and an approach for calculating the degree of taxonomical similarity between the tweets we obtained through the keyword search and the remaining tweets from a user's timeline.

The chapter is divided in two parts. First, we describe how obtain the Twitter user ranking using keyword search (TURKEYS), in section 4.1. Then, we apply this technique to improve follow relation-based Twitter user recommendation, in section 4.2.

4.1 Twitter user rank for keyword search (TURKEYS)

In this section, we explain the Twitter user rank for keyword search, or TURKEYS. Given an input query representing the topic of interest, we find tweets matching the query using Twitter keyword search, then extract user information and user relations from them. After we create a reference graph based on the user relations, the TURKEYS score of each user is calculated. We will give a summarized explanation for TURKEYS; see [46] for a more in depth explanation. The main assumptions for this approach are:

1. Users who post many valuable tweets and retweets about the topic are worth following;

2. Valuable tweets attract attention from many users;

3. Each user pays attention to tweets he/she retweets or replies to, but also watches the other tweets to some extent (not more than retweets and reply tweets).

Based on these assumptions, the TURKEYS score of each user u is defined as follows.

$$\mathrm{TURKEYS}(u) = \mathrm{TC}(u)^w \times \mathrm{UI}(u)^{1-w} \tag{4.1}$$

$\mathrm{TC}(u)$ and $\mathrm{UI}(u)$ are respectively *tweet count* score and *user influence* score of user u, ranging between 0 and 1. Weight w also ranges between 0 and 1. The tweet count score is based on the number of tweets for each user and reflects the first assumption; the user influence score is based on user relations among users and reflects the second and third assumptions.

Users and tweets are collected by keyword search. Let the set of tweets obtained and the set of users obtained be T_{all} and U_{all} respectively. Then the tweet count score is calculated by counting not only original tweets but also retweets in T_0 as each user's own tweets. The score is dampened by a logarithm function and is normalized so that the largest possible value is 1:

$$\mathrm{TC}(u) = \frac{\log\left(1 + |\{t | t \in T_0 \wedge t.user.id = u.id\}|\right)}{\max_{u' \in U_{all}} \log\left(1 + |\{t | t \in T_0 \wedge t.user.id = u'.id\}|\right)}, \tag{4.2}$$

where $t.user.id$ indicates poster's ID of the tweet t and $u.id$ indicates the ID of the user u.

Next, we define the *user influence score* and *tweet influence score*. The user influence score of each user is calculated using the tweet influence score of original tweets and retweets the user posted, and the tweet influence score of each tweet is calculated using the user influence score of users who pay attention to the tweet.

We create a tweet-user reference graph from the tweet set T_{all} consisting of user nodes, tweet nodes, and directed edges between a user node and a tweet node. Using adjacency matrix transformations for this graph, we obtain matrices B_t and B_r, which encode our third assumption above numerically: that is, each user pays attention to tweets he/she retweeted or replied to, and watches all tweets (regardless of his/her activity of retweet and reply) at a certain rate.

The user influence score and the tweet influence score are then calculated as follows.

$$\mathbf{u} = B_t^T \mathbf{t} \qquad\qquad \mathbf{t} = B_r^T \mathbf{u} \tag{4.3}$$

where $\mathbf{u}$ and $\mathbf{t}$ indicate the column vector of the user influence score of all users and the column vector of the tweet influence score of all tweets respectively. We can calculate the user influence score and the tweet influence score using the power iteration method. The iterative processes are as follows.

$$\mathbf{u}_k = B_t^T B_r^T \mathbf{u}_{k-1} \qquad\qquad \mathbf{t}_k = B_r^T B_t^T \mathbf{t}_{k-1} \tag{4.4}$$

$\mathbf{u}_0 = (\frac{1}{|U_{all}|}, \frac{1}{|U_{all}|}, \ldots, \frac{1}{|U_{all}|})$;
$\mathbf{t}_0 = (\frac{1}{|T_{all}|}, \frac{1}{|T_{all}|}, \ldots, \frac{1}{|T_{all}|})$;
$k = 1$;
Repeat
  $\mathbf{u}_k = B_t^T B_r^T \mathbf{u}_{k-1}$; $\mathbf{t}_k = B_r^T B_t^T \mathbf{t}_{k-1}$;
  $k = k+1$;
until $|\mathbf{u}_k - \mathbf{u}_{k-1}| < \varepsilon_u$ and $|\mathbf{t}_k - \mathbf{t}_{k-1}| < \varepsilon_t$;
return $\mathbf{u}_k$ and $\mathbf{t}_k$;

Figure 4.1: Calculation algorithm for the user influence score and the tweet influence score.

where $\mathbf{u}_k$ and $\mathbf{t}_k$ indicate the user influence score and the tweet influence score at the k-th iteration respectively. The calculation algorithm for the user influence score and the tweet influence score is shown in Figure 4.1. ε_u and ε_t are error tolerance parameters. Lastly, the user influence score of each user is normalized so that the largest value should be 1, as UI:

$$\mathrm{UI}(u_j) = \frac{\mathbf{u}(j)}{\max_k \mathbf{u}(k)} \tag{4.5}$$

The TURKEYS score defined in equation 4.1 is calculated using equations 4.2 and 4.5. We take the top-k ranked users as candidates for the next phase.

4.2 User recommendation using ontological timeline analysis

Once we have obtained a ranking of users based on keywords and user influence, we want to select only those users that continuously post about topics closely related to the keyword. However, a user may not always be using the same keywords when talking about a topic. Consider, for example, the topic "space development". There are many keywords that relate to this topic (e.g. "NASA", "SpaceX", "Curiosity", etc.). Users will likely not be mentioning any one of these all the time, and it is difficult and time-consuming to come up with an exhaustive search query. Another example is a user searching by a keyword "whaling". This user likely also be interested in discussions about the hunting of dolphins as well. Some of these posts may go undetected when relying solely on keywords. Therefore, we aim to encapsulate multiple keywords that can be traced back to the same topical domain by using ontological background knowledge. Only if a user consistently posts about the same topic do we consider this user to be appropriate to follow.

There are several difficulties that need to be overcome, which are listed below. The solutions to these difficulties comprise the sequential steps of our proposed approach.

1. *User tweet collection*: this time, we need tweets from specific users rather than tweets related to a certain topic.

2. *Named entity extraction and classification*: we need some way to extract named entities from tweets, then use a background ontology to obtain class information for extracted keywords (we use the approach described in section 3.2).

3. *Topic representation*: we need a way to represent the topic(s) of interest. Here, it is important to encapsulate just the right amount of generality. For example, when a user is interested in baseball, "sports" would be too generic, but "major league baseball players" might be too specific to determine topic consistency. For this part, we use the *class pruning* method we proposed earlier, in section 3.2.2.

4. *Topic consistency checking*: once we have a topic representation, we need to determine whether a user posts about this topic consistently. To achieve this, we propose the *taxonomical similarity score*.

We take the output of the user recommendation method described in the previous section; for the top users, we derive a topic representation for (1) the tweets made during the period of tweet mining for the user relation-based recommendation, and (2) a selection of tweets made outside this period. We then compare the two topic representations and determine their similarity; if this is above a certain similarity threshold, we can say that this user posts about a topic consistently.

Our method thus applies a binary selection to the user ranking: starting at the best ranked user output in the previous step (user recommendation based on user relations), we decide whether this user is valuable or not based on a threshold of *taxonomical similarity* across his timeline. We continue down the ranking, until we have found the target k of top-k users.

In the following sections we will explain each step of the approach in detail.

4.2.1 User Tweet Collection

We take the top ranked users and their tweets that are output from the recommendation step based on user relations. We call this set of tweets belonging to one user T_{mine}. Additionally, for each user we collect earlier tweets from their timeline using the Twitter API. Although the Twitter API allows one to retrieve a maximum of 3200 tweets from a user's timeline, the subsequent entity recognition step that we perform on the tweets is somewhat computationally intensive, so we decide to collect a maximum of 500 tweets per user profile. We call this set of tweets $T_{profile}$. Here, we exclude tweets that were made during the mining period for the previous step, i.e. $T_{mine} \cap T_{profile} = \varnothing$.

We apply entity recognition using DBpedia Spotlight and perform ontological expansion on both sets of tweets separately, as described in section 3.2. Recall that we collect all entities and class types and combine them into class frequency maps $\mathrm{cf}_{i,c}$ for each user, containing each unique class and how often it occurs. We construct these maps for both tweet sets T_{mine} and $T_{profile}$. For convenience, we denote the generic cf maps for the mined tweets and profile tweets as C_{mine} and $C_{profile}$, respectively.

4.2.2 Topic Representation

Next, we trim the sets of classes C_{mine} and $C_{profile}$ to obtain an accurate representation of a user's topic(s) of interest. Since entity recognition on informal tweets is far

from perfect, we need to discover and remove any classes that have been linked to mismatched entities. Second, since we collect classes up to the root of the hierarchy, we need to filter out classes that are too generic and cover too broad of a topic.

We apply the class pruning method that we proposed in section 3.2.2 for this part. We end up with a compact selection of classes per user that represents their topic of interest best.

4.2.3 Topic consistency checking

Finally, we need to compare the topic representations of the tweets collected in the user relation mining period and remaining tweets collected from a user's timeline, in order to determine whether or not a user posts about the same topic consistently. We have two sets C_{mine} and $C_{profile}$ of classes c_i, and each class has a number of occurrence, which we will denote as $occs(c_i)$. We devise a simple way to determine the similarity between the two sets.

First, we take the smallest of the two sets, in terms of the total number of class occurrences. For simplicity, we will assume this to be C_{mine} as this is true in virtually every case. We then count for each class c_i in C_{mine} how many occurrences of this same class c_i are also covered in $C_{profile}$. Finally, we normalize by dividing the total number of common occurrences found by the total number of class occurrences in C_{mine}. Formally:

$$s = \frac{\sum_{c_i \in C_{mine} \cap C_{profile}} \min\{occs(c_i)|c_i \in C_{mine}, occs(c_i)|c_i \in C_{profile}\}}{\sum_{c_i \in C_{mine}} occs(c_i)}. \tag{4.6}$$

The *taxonomical similarity score s* that is obtained is a number between 0 and 1. We can subsequently define a threshold parameter θ_s between 0 and 1 to determine whether two taxonomies are similar enough to consider the user to be posting about the same topic consistently. We apply a binary selection on the user ranking, starting with the highest ranked user. For each user, if s is above θ_s, the user is kept in the ranking. If s is below θ_s, the user is removed from the ranking. We continue until the desired top-k ranked users are obtained, or until we reach a stop condition (e.g. process only the top 50 users).

The evaluation for this approach, detailing the degree in which it can improve user recommendation on Twitter, can be found in section 7.4.1.

Chapter 5

Ontology-assisted Discovery of Hierarchical Topic Clusters on the Social Web

In this chapter, we will detail the methods we devised for discovering ontology-based, hierarchical representations of user interests, and for topic clustering. It is our second and most comprehensive method, in which we extend and generalize the ontological timeline analysis from the previous chapter to allow comparison not just within single timelines, but among arbitrary users, based on *scoped topical similarity*. This scoped topical similarity allows us (1) to detect clusters of users at a chosen topic scope (a slice of the topic hierarchy present in a collection of users), without having to pre-define the number of topics beforehand; (2) to automatically label clusters with machine-readable topic tags; and (3) divisively cluster users into a topic hierarchy. This method is designed to be especially useful in a generic, noisy Social Web environment, rather than any specific application. However, for simplicity's sake, we will still focus on Twitter as a use case for the majority of this chapter.

The chapter is organized as follows. First, in section 5.1, we give a short introduction into user interests on the Social Web, and our formal definition of a Social Web user. We then explain how we represent users and topics at different topic scopes, in section 5.2. This is followed by an explanation of how we use community detection techniques to divide users into scoped and labeled clusters, in section 5.3.

5.1 User interests on the Social Web

We assume that we have a collection of Twitter users that we want to cluster by topic. For simplicity, we make the assumption that each user has some current, static topic of interest. Realistically, user interests are dynamic and evolve over time, so an accurate topic model would need to incorporate the time dimension in some form. However, as this would significantly increase to complexity of the problem at hand, we choose to simplify this by simply taking the latest posts on a user's timeline as their current topic of interest. Furthermore, we also assume this topic of interest is hierarchical in nature: that is, a user generally interested in "sports", might be specifically interested in "hockey" and "golf" to different degrees. We aim to develop a method that can accurately capture this type of information.

For practical purposes, there are a number of ways to obtain an initial set of Twitter users, depending on our specific objective. If we want to group a user's followers or friends into shared-interest groups, we can collect this direct neighborhood of the user and process the gathered collection of users. If we want to find out for some ambiguous keyword, say "football", which users are interested in American football and which are interested in soccer (known as "football" outside of the US), we can collect those users by keyword search. Or, we might have a pre-defined list of users that we want to cluster.

To differentiate Social Web-oriented clustering from traditional document clustering, we consider a collection of users U, as opposed to documents D. These notions are conceptually similar: each u_i in U contains some number of posts $\{p_1, p_2, ..., p_n\} \in P_i$, similar to how a document $d_i \in D$ would consist of lines or paragraphs, which are both in turn collections of words $\{w_1, w_2, ..., w_m\} \in W_i$. In order to keep our approach application-agnostic, we consider only the raw text content associated to users – usually their raw text (micro-)posts P_i. We have already introduced the class frequency map cf in section 3.2.1. Over the course of our approach, we will additionally compute a trait vector $\mathbf{t}_i$ for each user. These concepts will be explored in greater detail in the coming sections.

At this point, we can define a Social Web user as follows.

Definition 1. *A Social Web user u_i in a collection of users U is a 3-tuple*

$$u_i = (P, cf, \mathbf{t}), \quad u_i \in U, \tag{5.1}$$

where P is a collection of (micro-)posts; cf is a class frequency map, which is a bag of all entities and classes discovered in P with their number of occurrence; and $\mathbf{t}$ is a trait vector composed of weights expressing topic similarity.

5.2 Discovering scoped topics

For our second method, rather than determining the relevance of some user with regard to a keyword, we want to determine the relevance level of one user compared to another user: that is, calculate the topical similarity between pairs of users. We can achieve this by generalizing the user recommendation method from the previous chapter in a number of important ways. First, in order for the approach to work on different users, we no longer divide individual user timelines into mining/non-mining sections; instead, we consider only full user timelines. Second, we need a more generalizable and robust approach for deriving user topics: we replace the heuristic for calculating the taxonomical similarity based on class occurrences with a more formal and generally applicable approach. Third, we introduce the assumption that topics are hierarchical. We assumed flat interests for the previous approach, but this is not a realistic assumption in real-life situations. We assume there to exist a topic hierarchy within a collection of users: we want to be able to cluster users by topics of different scopes (that is, different slices of the hierarchy). In this section, we will describe our methods and equations for achieving this.

5.2.1 cf-iuf weighting and trait vectors

Recall our novel cf-iuf weighting strategy, proposed in section 3.2.3. We reprint the equation here for convenience:

$$\text{cf-iuf}_{i,c} = \text{cf}_{i,c} \times \text{iuf}_c, \text{ where } c \in C_i. \tag{5.2}$$

We can calculate a cf-iuf weight for each unique class that occurs for each user. It follows that, using this equation, classes that occur many times for a select group of users get assigned high weights, while classes that are relatively common, occurring for a large portion or all of the users, get assigned low weights. We use this weighting strategy as components for a *trait vector* $\mathbf{t}_i$, to project characteristic user traits into a novel *trait vector space* model (as opposed to the common term vector space):

$$\mathbf{t}_i = [w_1, w_2, ..., w_{|C_i|}]^T, \text{ where } w_k = \text{cf-iuf}_{i,c_k} \text{ and } c_k \in C_i. \tag{5.3}$$

In other words, each user has as many traits as he has unique classes, and these traits are weighted according to their overall uniqueness compared to other users. The traits with the highest weights characterize this user best within the full collection of users.

5.2.2 Scoped topical similarity

Given a collection of users $u_i = (P, \text{cf}, \mathbf{t}),\ u_i \in U$ (as first introduced in definition 1), we can group similar users by their common topics of interest. It is hard to define "topics of interest" concretely, for a number of reasons. First of all, the topic scope can be almost arbitrarily narrow or broad (consider e.g. *Premier league players* $\rightarrow$ *Football* $\rightarrow$ *Sports* $\rightarrow$ *Activity*): there is no "true" scope for the topics, as this depends completely on what we are interested in finding. A second reason is that there is ambiguity as to the delineation of topics, and how many and what kind of concepts can even be considered "topics". This is often called the "user's dilemma" [27]: each observer has different ideas about what constitutes a "topic" and how to draw the right delineation between different topics. We deal with these problems by (1) making the scope of topics sought controllable: different use cases require different topic allocations, so we allow the retrieval of topics at arbitrary levels of the latent topic hierarchy; and (2) inferring mutual topics of interest and their boundaries from the underlying data based on users' *scoped topical similarity*.

Controlling topic scope The scope of the topics that we want to find largely depends on the application – as explained above, the scope of the topics we seek is application-specific, and there is no "right" or "wrong" scope or number of topics. Exploiting the hierarchical properties of the ontological classes that form the basis of our user trait vectors, we introduce a *topic scope* parameter, γ, into the equation for the cf-iuf weighting strategy. This allows us to forego the definition of a target number of topics for the clustering that is common to most existing topic modeling and document clustering techniques.

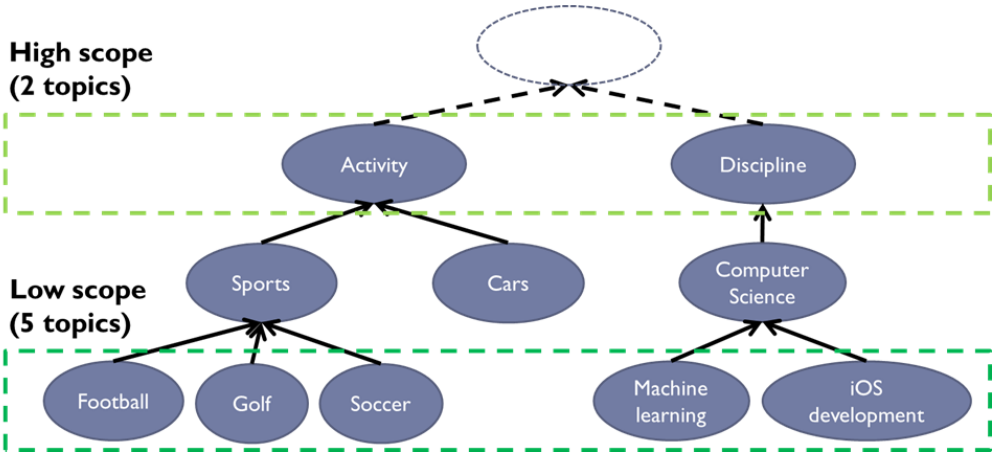

Figure 5.1: Manipulating the *topic scope* parameter γ in order to cluster users at an arbitrary height of the topic hierarchy. In this example, we can cluster by two broad topics (high scope) or five specific topics (low scope).

$$\text{cf-iuf}_{i,c} = \text{cf}_{i,c}^{1+\gamma} \times \text{iuf}_{c}^{1-\gamma}, \quad -1 \leq \gamma \leq 1, \quad \gamma \in \mathbf{R} \tag{5.4}$$

γ can take on any real value between -1 and 1. If we set γ closer to -1, we put more weight on the inverse user frequency and less on the class frequency, leading to a bias towards rare classes (a lower slice of the topic hierarchy). Conversely, if we set γ closer to 1, we put more weight in the class frequency and less on the inverse user frequency, leading to a bias towards common classes (a higher slice of the topic hierarchy). This behavior was depicted in figure 1.2 of the introduction; we reprint it in 5.1 for convenience.

The exact generality of classes that we obtain at a given, absolute value of γ depends on a number of other dependent variables of our approach, which we will introduce in the coming sections. Therefore, there is a need to *calibrate* γ to yield a certain topic scope at a certain value, and optimize the remaining variables around this calibration. We describe this calibration and optimization process in section 7.3.4.

Inferring mutual interests: *scoped topical similarity* Given each user's trait vector $\mathbf{t}$, we can find which users are similar to each other at scope level γ by calculating the pair-wise cosine similarity between all users in U. This is done by calculating the dot product over each user's trait vectors and dividing by the product of their Euclidean length. We call this the *scoped topical similarity* (STS) function between users.

$$STS_{\gamma}(u_i, u_j) = \frac{\mathbf{t}_i \cdot \mathbf{t}_j}{|\mathbf{t}_i||\mathbf{t}_j|} \tag{5.5}$$

Since the cf-iuf weights for traits were normalized, it follows that $0 \leq STS_{\gamma}(u_i, u_j) \leq 1$. At any point, we can alter the topic scope γ and re-calculate the trait vectors $\mathbf{t}$ and the STS_{γ} between all users for different topic distributions. However, this is not enough

the determine topic boundaries within a collection of users; the next section will detail how we cluster users into distinct topics using community detection techniques.

5.3 Hierarchical topic clustering

In this section, we describe how we use graph-based community detection techniques to perform the clustering of users into topics. The key characteristic of this type of approach is that we do not need to pre-define the number of topics to cluster in. First, we explain how we represent users and their scoped topical similarity as a weighted graph in 5.3.1, followed by a detailed description of the methods we developed for scoped clustering in 5.3.2, topic labeling in 5.3.3 and recursive clustering to construct topic hierarchies in 5.3.4.

5.3.1 Scoped topical similarity graph

Recall that we have a collection of users $u_i = (P, \mathrm{cf}, \mathbf{t}),\ u_i \in U$, where a user's posts P and class frequency map cf are static, and the composition of trait vector $\mathbf{t}$ is calculated given the desired topic scope γ. We can construct a *scoped topical similarity graph* $G_\gamma = (V, E)$. G_γ is an undirected, weighted graph where each vertex is a user and each edge connects two users, with their STS_γ as edge weight. Initially, we calculate edge weights between every pair of users to obtain a fully connected graph.

Such a graph is not an ideal form if we want to find topic clusters; it is hard to find clusters within a fully connected graph (although not impossible since our graph is weighted), and the number of edges $|E|$ will be exponential in $|V|$, leading to intractable computational complexity when dealing with large graphs. We solve this by first pruning the edges E, imposing a minimum STS_γ threshold τ. In other words, if two users are less than τ similar, we consider them too different to have any kind of interest in common. It is not immediately clear which value of τ would yield the best results. We include τ as one of three dependent variables that we attempt to derive an optimal configuration for through hyperparameter optimization; see section 7.3.4 for details.

Pruning helps us discover topic-based clusters in the graph. When visualized, as in figure 5.2 for example, the topological graph structure gives an intuitive idea of where clusters are located, but is not sufficient to say anything explicit about the clusters and their content. We can apply community detection techniques to isolate highly-connected clusters into distinct topics. We can expect to find a high or low number of clusters if the graph G_γ has a low or high value for γ, respectively. In other words, we can find topics of an arbitrary scope by taking different slices of the inherent topic hierarchy. How many topics we end up with depends entirely on the underlying data: in the evaluation, we will see that we can find different and roughly the appropriate number of topics at the same setting of γ for different datasets. Figure 5.3 gives an example of the different results between clustering a testset of Twitter users at $\gamma = 0.8$ and $\gamma = 0$. We explain how we arrived at this result in the next section.

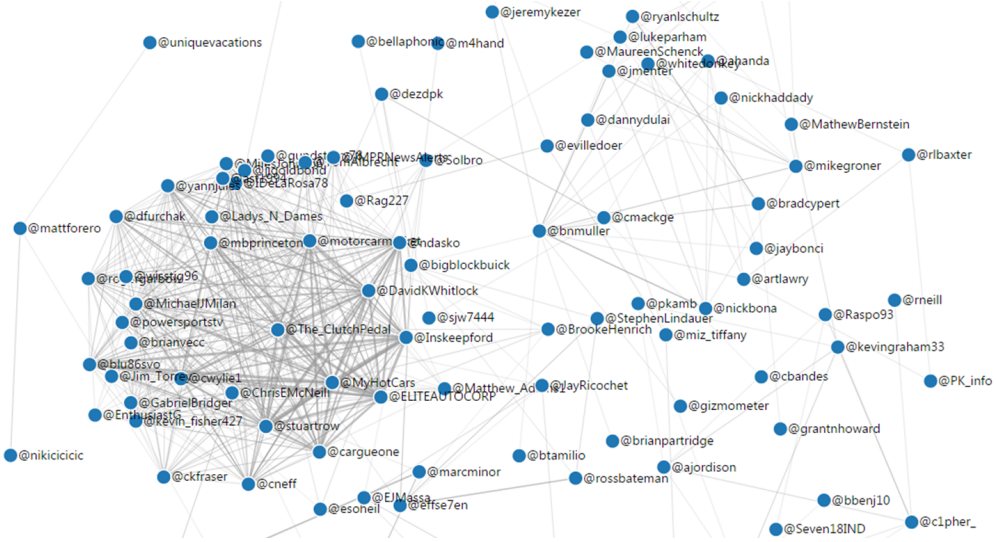

Figure 5.2: An example of a partial, pruned topical similarity graph of Twitter users.

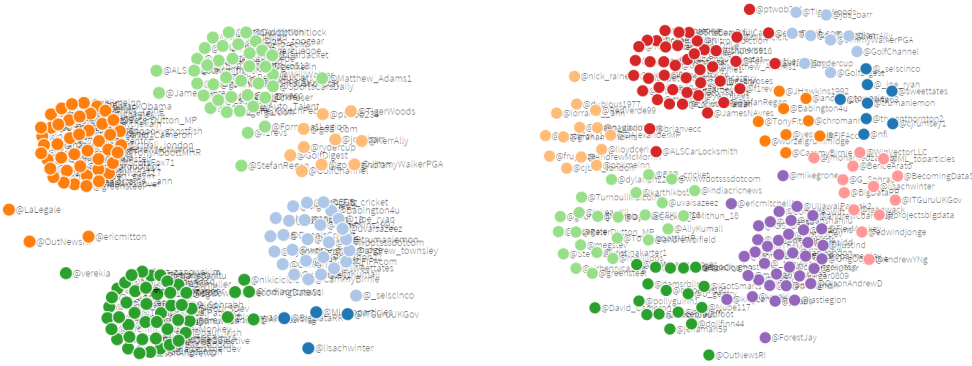

Figure 5.3: Clustering a testset of Twitter users at $\gamma = 0.7$ (left), yielding 6 clusters, and $\gamma = 0$ (right), yielding 9 clusters.

5.3.2 Finding highly-connected subgraphs

We now describe how we perform topic clustering on the pruned STS_γ graph G_γ, using a custom version of the Highly Connected Subgraph (HCS) clique-based, strict partitioning clustering algorithm [22]. Originally, this algorithm works by dividing a graph into subgraphs that fulfill some minimum edge degree condition. First, a minimum cut is determined, along which the graph is cut. The resulting subgraphs are examined to see if they are *highly connected*; each vertex in the subgraph must have an edge degree that is at least greater than half the number of vertices in the subgraph. If a subgraph is not highly connected, the algorithm is repeated on this subgraph, assuming there are at least 2 vertices left. HCS allows outliers: singleton and twin vertices are not considered clusters, and therefore discarded. The final result is a clustered graph containing however many highly connected subgraphs were found.

For our purposes, we modify this algorithm in three ways. Firstly, the original algorithm was developed for unweighted graphs. Since we deal with a weighted graph, we apply a minimum cut[1] algorithm that takes weights into account. Secondly, the original full-clique constraint for determining highly-connectedness of a cluster is too strict for our purposes; we relax these constraints to obtain a *quasi-clique* algorithm. Lastly, we do not want to drop outliers from the result, thus also apply a *singleton adoption* heuristic as described in [22]. We explain these three modifications in brief.

Weighted minimum cut An important consideration when using the HCS algorithm is which algorithm to use for the sub-task of finding a minimum cut in the graph. Technically speaking, we seek the *maximum* cut, since a higher weight means a higher similarity between users, but for simplicity's sake we maintain the common terminology. Since we have a weighted graph, G_γ, we use a modified version of Kruskal's minimum spanning tree algorithm [31]. First, we sort all edges $E \in G_\gamma$ by STS_γ in descending order. Then we start constructing the minimum spanning tree as per Kruskal's algorithm using the sorted edge list, but stop at the second-to-last step, at which point we have two trees that contain all of the vertices. The last step would connect the two trees to form the minimum spanning tree; since we constructed the trees in descending order of edge weight, it follows that the cut between the two trees represents the minimum cut. A proof of this can be found in [13].

Highly-connectedness constraint We argue that the original, strict constraint to determine highly-connectedness of subgraphs may not be optimal for all use cases. We can rewrite this constraint as follows, parameterizing the numerator for the proportion of vertices that must be higher than the minimum vertex degree of a subgraph as α:

$$\min\{degrees(v_j) \mid 1 \leq j \leq |V_{sub}|,\ v_j \in V_{sub}\} > \frac{|V_{sub}|}{\alpha} \tag{5.6}$$

In the original algorithm, $\alpha = 2$; we experiment with different values for α. A larger α means a less strict lower bound requirement for a highly connected graph, hence we find larger subgraphs compared to low values of α. We call subgraphs determined to be highly connected *topic clusters*.

Singleton adoption With the standard version of the algorithm, lowly connected or isolated nodes will get excluded from the resulting clustered graph. This is not desirable for our purposes; we want to allocate users to at least one topic, even if their affinity to the other users belonging to this topic is low. This also makes evaluation against a ground truth and other clustering methods easier. We apply a singleton adoption heuristic where for all remaining, unassigned users after one iteration of the standard algorithm, we recalculate the STS_γ between each unassigned user trait vector $\mathbf{t}_i$ and each cluster-based trait vector $\mathbf{t}_k$. The latter has been calculated from the union of all classes of all users assigned to that cluster, compared to those of other clusters. The next section explains this in more detail.

[1] Strictly speaking, we seek the *maximum* cut, since a higher weight means a higher similarity between users, but for simplicity's sake we maintain the common terminology.

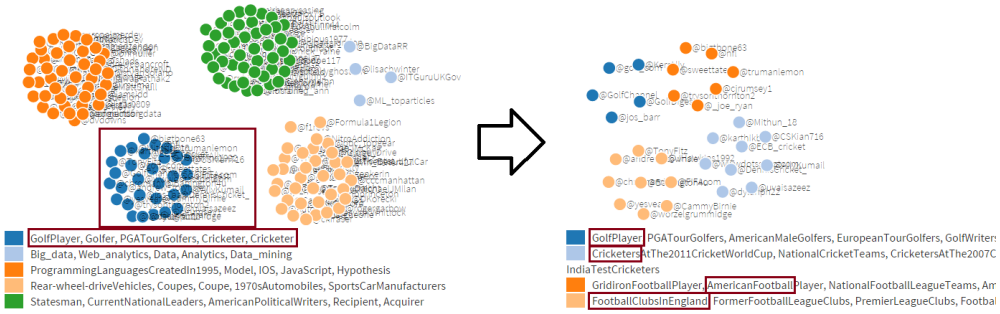

Figure 5.4: An example of cluster labeling and recursive clustering. For each cluster k, the top-5 cluster-based traits from $\mathbf{t}_k$ are listed. A "Sports" cluster is isolated and further sub-divided into the individual sports that made up the cluster.

5.3.3 Topic clusters and labeling

An advantage of using ontology classes to model topics is that it allows us to reason about the semantic content of topic clusters. One important result is that we can obtain intuitive labels for the clusters by regarding clusters as single entities, and re-calculating STS_γ; this time, we calculate the similarity between *clusters* rather than *users*, and take class names of the top scoring cluster traits as labels. We formally define *topic clusters* as follows.

Definition 2. *A topic cluster T_k in a clustered graph G'_γ with K clusters is a 3-tuple*

$$T_k = (U_k, cf_k, \mathbf{t}_k), \quad T_k \subseteq G'_\gamma, \ 1 \leq k \leq K, \ U_k \subseteq U, \tag{5.7}$$

where U_k is the subset of users assigned to the k-th topic cluster; cf_k is the combined class frequency map for the cluster taken from U_k; and $\mathbf{t}_k$ is a trait vector that expresses cluster-based topical affinity.

In other words, we merge the class frequency maps for every user $u_i \in U_k$ in a cluster T_k to form a cluster-based class frequency map cf_k. From all cf_k, we calculate a *cluster frequency map* clf which records for each class in how many clusters this class occurs at least once (analogous to the user frequency map uf). We can then take the same steps as before to calculate cluster-based, cf-iclf-weighted trait vectors $\mathbf{t}_k$. Sorting these trait vectors by their topical affinity component in descending order provides us with the most characteristic traits for each cluster at the top of the list; we use the names of the top traits as topic cluster labels. See the left side of figure 5.4, for example, which shows the result of clustering a testset of 175 Twitter users into $K = 5$ dominant topics ($\gamma = 0.8$). For each cluster k, the top-5 cluster-based traits from $\mathbf{t}_k$ are listed.

5.3.4 Recursive topic clustering

While we consider altering the topic scope γ and re-calculating trait vectors a form of hierarchical clustering (since we can obtain topics at an arbitrary height in the latent

topic hierarchy – something we cannot with traditional approaches), our approach also allows divisive clustering to generate a full topic hierarchy in a top-down fashion. After one clustering iteration, we can apply the clustering once again on each resulting cluster separately. That is, we ignore all users that do not belong to the selected cluster, and re-evaluate the trait vectors of the remaining users. Since dominant traits of each user are now calculated only with regard to the other users originally in the same cluster, we can further divide users into more specific topics of interest within this more general topic of interest. We call this type of recursive topic clustering *hierarchical STS*, or STS_h, and it allows us to extract a full topic hierarchy, rather than just a slice, from a collection of users. An example of one iteration of recursive clustering is shown in figure 5.4, where we have a cluster of users that is about "Sports", as is evident from the topic labels. We can sub-divide this topic cluster into sub-clusters lower in the topic hierarchy. We find the cluster consists of "Golf", "Cricket", "American football" and "Football". A simple pseudocode representation of the recursion we perform is shown in algorithm 1.

Algorithm 1 STS_h: Recursive topic clustering to obtain a full topic hierarchy.

```
1: procedure CLUSTERGRAPH(G(V,E)):
2:     V ← calculateTraitVectors(V, γ)
3:     for e in E do
4:         e.weight ← STS(V[e.i], V[e.j])
5:     subGraphs[] ← HCS(G)
6:     for subGraph in subGraphs do
7:         clusterGraph(subGraph)
```

Chapter 6

Implementation and Algorithmic Analysis

In the previous chapter, we have described an approach for the hierarchical clustering of Social Web users. Now, we will implement all of these methods and techniques into a prototype Web application, in order to demonstrate the advantages of our approach when dealing with real Social Web data gathered from Twitter. We build a system that obtains users from Twitter given an input seed user or keyword, applies all the steps of the approach to calculate scoped topical similarity between users, then visualizes construction and clustering of a graph of users in real-time. Additionally, we provide a time complexity analysis of the algorithms we implement, and compare them to popular algorithms for the clustering task (Lloyd's *k*-means clustering algorithm and LDA with Gibbs sampling).

The chapter is organized as follows. First, We give a system overview of Twinterest Explorer, the application we designed, in section 6.1. This is followed by an analysis of the algorithms used and their complexity, scalability considerations, and some benchmarks between our method and other approaches, in section 6.2.

6.1 Twinterest Explorer

In this section, we will explain Twinterest Explorer[1], a Web application that enables the exploration of interests on Twitter. See figure 6.1 for a screenshot of the interface. It does real-time topic clustering and automatic labeling of users on Twitter, based on their unstructured timelines of microposts. A demo set of 175 Twitter users is included as well – this is the same set we use for the evaluation of topic clustering, and is explained in section 7.1.3.

Twinterest Explorer can group and visualize streams of users by topic (or interests). Users are gathered directly from Twitter through keyword search, by collecting the neighborhood (followers/friends) of a seed user, or by providing a user-defined list of screen names. Desired topic scope can be arbitrarily narrow or broad (e.g. *AngularJS* $\rightarrow$ *WebFrameworks* $\rightarrow$ *ProgrammingLanguages* $\rightarrow$ *Structure*): this can be adjusted in real-time with a slider. Detected clusters are visualized as new users are processed in

[1] Source code and installation instructions can be found at: `https://github.com/ktslabbie/TwinterestExplorer`

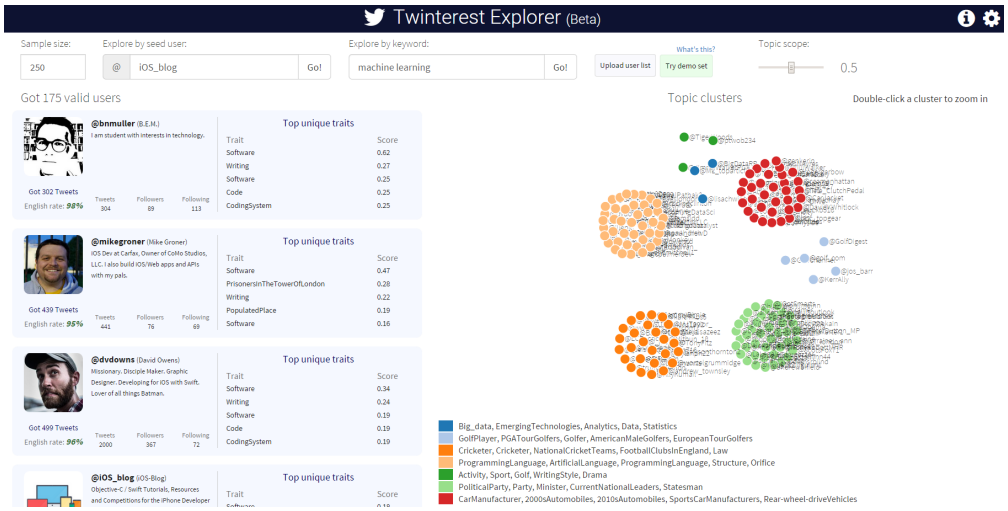

Figure 6.1: A screenshot of TwinterestExplorer, a prototype application for the real-time clustering of Twitter user streams.

the background, using a colored and labeled graph. This graph can be explored by adjusting topic scope and zooming in on clusters. Zooming in will allow a cluster to be divided into sub-clusters (sub-topics within the broader topic). Clusters are labeled automatically with the most characteristic traits for that cluster.

On a small scale where real-time visualization is still possible such as in this application, real-life usefulness is somewhat limited. Still, there are some interesting things it can show. We can find out which followers/friends of some user post about conceptually similar things, and have them grouped and labeled by interest automatically. Or we can discover which users are consistently interested in a given topic to get a better idea of who to follow (is their mention of "machine learning" a one-time event, or are they consistently interested in it?). For ambiguous keywords, we can separate people into concrete interest groups. Spambots tend to get grouped together, so it could theoretically be used as a spam filter as well. Finally, we can apply semantic labels to clusters. Generally, using traditional topic modeling methods, the best we can is to generate word clouds from the terms that appear most in each document, and actual semantic labeling is not possible due to the reliance on word co-occurrence.

6.1.1 Technical implementation

We have created a diagram of the main system architecture and its components in figure 6.2. The project can be roughly divided into a frontend and backend, in a client-server type of architecture. We will explain the details of each component in the following two sections.

Backend The backend is a standard Java application with a REST interface that is exposed to the client. Restricting scope to the backend, it communicates with:

1. a DBpedia Spotlight server to annotate tweet content with links to DBpedia resources;

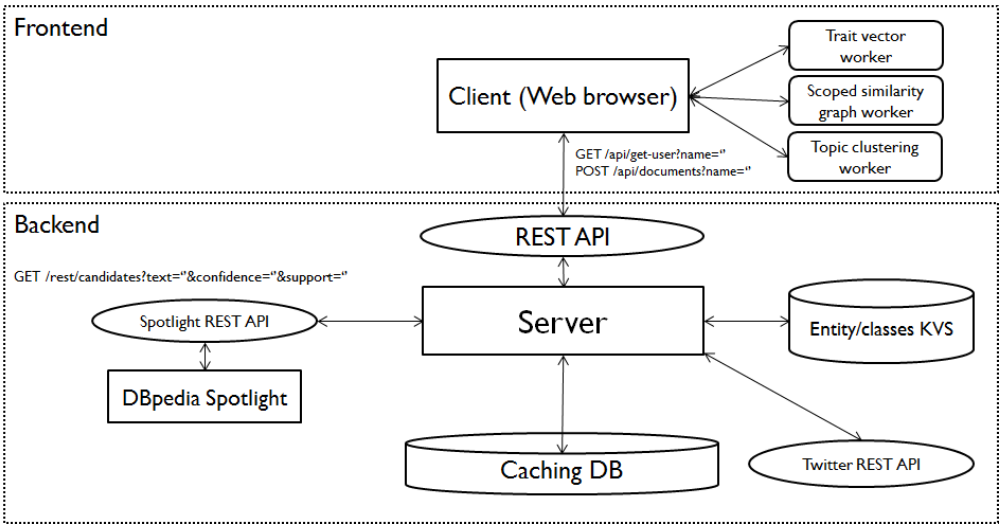

Figure 6.2: An overview of the client-server architecture of the Twinterest Explorer system.

2. a Redis [58] key-value store instance, which contains flattened versions of the DBpedia, YAGO and Schema.org class hierarchies from DBpedia (lists of classes up to the root of the hierarchy for each resource);

3. a PostgreSQL [43] database, mostly used for caching users, and tweets and classes collected for them.

The general process flow for the backend is as follows:

1. get a request for a user, as well as some parameters, through the RESTful API;

2. check database for containment of this user:

   a) if not contained, call Twitter API and apply named entity recognition on tweets;

   b) otherwise, apply only named entity recognition or return immediately.

Applying named entity recognition means sending (concatenated) tweet text to DBpedia Spotlight and collecting and counting all classes (DBpedia, Schema.org, YAGO) up to the root of their hierarchies for all DBpedia resources detected. The backend API can handle many requests concurrently; the client is set up to send several requests at a time asynchronously.

Frontend The frontend is a Javascript application using the AngularJS [28] Web framework, that sends HTTP GET or POST requests to the backend to obtain users and their classes. As users come back, there are three processes that are performed in the client, sequentially and in parallel:

1. The users' scoped trait vectors consisting of the cf-iuf weights for each unique class of each user are calculated. We compute and update a *user frequency* hashmap as users come in, and keep using it as the current data is manipulated through the frontend interface and needs to be re-calculated. Recall that the user frequency for each class is the number of users for which this class was discovered in their tweets at least once. As explained in section 5.2.2, we can put extra weight on either generic or specific classes to bias the user trait vectors towards one or the other. This is done using a controllable *topic scope* correction parameter, which is implemented by a slider that can be set to a range of real numbers between -1 and 1. The topic scope can be changed at any time; whenever a change is detected, the trait vectors are re-caclulated and the similarity graph and cluster graph re-constructed.

2. The cosine similarity between all pairs of users' class collections is calculated, based on their scoped trait vectors scores. Essentially, users and classes are represented as sparse matrices, with users in the rows and trait vectors in the columns. Calculating the cosine similarity between all user pairs is then a matrix multiplication operation where we calculate the dot product between trait vectors of each possible user pair. Since classes are stored in hashmaps, calculating this takes $O(n^2c)$ time, where n is the number of users and c is the number of unique classes, given naive matrix multiplication. Since our matrix is sparse, some optimizations may be possible [71][73], but this is outside the scope of our thesis. This similarity is then used as edge weights for the *scoped topical similarity graph* $G_\gamma(V,E)$, where each vertex is a user and each edge weighted with the *scoped topical similarity* STS_γ between these users.

3. Lastly, the similarity graph is clustered using a custom version of the Highly Connected Subgraph (HCS) algorithm. We have explained how we modified HCS in section 5.3.2. The Kruskal minimum spanning tree algorithm outputs two or more sets of nodes as clusters. For each cluster found, we take all the edges that would fall within this cluster (i.e. edges between two nodes within the same cluster). The corresponding vertices and edges are added to a graph object. Edges between nodes of different clusters are dropped from the result.

The final scoped topic cluster graph G'_γ is displayed to the user using a D3[2] force graph, and can be manipulated by altering the topic scope (recalculating from trait vectors onward with a different topic scope γ), or by zooming in on a cluster: the algorithm is re-run for those isolated users; essentially a recursive step to drill down into the topic hierarchy an arbitrary number of times for increasingly narrow topics. This was explained in section 5.3.4.

Parallel processing with Web Workers The three calculation steps in the frontend described above are performed continuously and concurrently as new users come in, using HTML5 Web Workers [68] to make full use the client machines' (presumably) multi-core environments, as well as to prevent the UI thread from blocking. One

[2]D3.js - Data-Driven Documents: `http://d3js.org`

worker is used for each of steps 1 and 3; we spawn 6 parallel workers for step 2, as this is the most demanding (matrix multiplication).

One pitfall regarding Web Workers we must be wary of is that objects cannot be passed by reference to Worker threads. What this means is that each object (a standard JavaScript object) must be cloned before a Worker can start – this cloning is done in the main UI thread, causing some UI blocking, and takes time for large objects. In our case, the largest objects we deal with are each user's class frequency maps: to calculate the trait vectors of a collection of users in a Web Worker, we must send an array of all users and their class frequency maps, which can consist of thousands of classes each. Deep cloning this full array is not trivial and takes a large amount of time. It is the same on the way back: when we have finished calculating the trait vectors, they must be cloned before they can be received back by the main UI thread. We mitigate this by only sending out class frequency maps to the Worker once for each new user; the map is then stored in the Worker itself, and does not have to be re-cloned each time the topic scope or collection of users changes. Unfortunately, it is not possible to eliminate cloning when sending trait vectors back.

6.2 Algorithms and complexity

In this section we will briefly expand on the algorithms used in the LDA and k-means clustering baselines that we explained in the background chapter and will use for the evaluation, and our STS_γ-clustering, and compare their time complexity, in section 6.2.1. We touch upon the scalability of each method for larger data in section 6.2.2. We will then give some actual execution benchmarks of the different components of STS_γ-clustering and compare them to standard implementations of the baselines, in section 6.2.3.

6.2.1 Algorithmic time complexity

We examine the algorithmic time complexity of the main algorithms we handle in this thesis: LDA, k-means clustering, and STS_γ-clustering. Aside from deriving their big-O time complexity, we make an estimation of the number of operations needed assuming we apply the algorithms on our main evaluation testset of 175 Twitter users with 500 tweets each, clustering into 11 topics (see section 7.1.3 of the evaluation for more info).

LDA For LDA, we use a variation of the algorithm that uses Gibbs sampling included in the MALLET package [39]. MALLET implements one of the fastest known algorithms for Gibbs sampling [49]. Since this is the most computationally intensive component of LDA, we may consider its big-O time complexity to be the worst-case complexity for LDA as a whole. Gibbs sampling generates a Markov chain of samples that estimate the probabilities of each word in vocabulary v belonging to one of n documents, and each document in collection n belonging to one of k topics. It follows that each sample takes $O(nkv)$ time, and due to the probabilistic nature of the algorithm, we need to iterate the sampling i times (a common default for LDA is 1500 iterations). Therefore the total worst-case complexity of LDA is $O(nkvi)$. As an estimation for the number of operations given our testset: we have $n = 175$ users $k = 11$

topics, $v \approx 10,000$ words, and $i = 1500$ iterations. Rounding to orders of magnitude, we get an estimated worst-case of 10^{10} operations.

k-means clustering For k-means clustering, although we calculate trait vectors in the same way as for STS_γ-clustering, the most intensive part is the clustering itself, which we implement using a standard Lloyd's algorithm [47] (there are faster algorithms, but we consider these out-of-scope). Using this algorithm, we first randomly pick k nodes as means for the k clusters, then between all users n and each k, we calculate the cosine similarity between the trait vectors of n and the cluster-based trait vectors of the k centroid clusters. Given d trait vector dimensions (unique classes), this can be done in nkd time. Users are then grouped by cluster by minimizing the within-cluster sum of squares (WCSS), and a new mean is calculated for the cluster, which we can keep track of during the cosine similarity calculation. The algorithm is iterated i times until the k-means clusters converge: typically around 10 times. Given the initial random assignment, the result of a single execution cannot be assumed to be optimal: we need to iterate the whole algorithm j times (depends on the dataset, but typically 100 times) and pick the result with minimum overall WCSS. The total time complexity of this algorithm is therefore $O(nkdij)$. Given our testset, $n = 175$, $k = 11$ and $d \approx 1000$, so we can expect roughly on the order of 10^9 operations.

STS_γ-clustering For STS_γ-clustering, we briefly summarize the various parts of the procedure and their complexity. First, we apply entity recognition and ontological expansion. DBpedia Spotlight uses the Aho-Corasick string matching algorithm for entity recognition [3], which has a worst-case complexity of $O(n+m+z)$, where n is the corpus length (over 4 million Wikipedia pages), m is the query length (assuming a tweet concatenation window of 10, at most 1400 characters), and z is the number of matches (typically 1-2 per tweet, so 10-20 per query). Wikipedia pages are additionally indexed using a Lucene-based system, and Spotlight can be accessed concurrently. Ontological expansion is done using a pre-populated hashmap, and can thus be done in $O(1)$ time. Therefore performance of entity recognition and ontological expansion is not a significant bottleneck.

The next step is trait vector calculation. We need to calculate one inverse user frequency map iuf_c and n $\text{cf} - \text{iuf}_{i,c}$-weighted trait vectors, where n is the number of users and c the number of unique classes. This can be done in $O(n+nc)$ time, a relatively low polynomial.

Then we need to calculate the STS_γ between each user to build graph G_γ. This is a matrix multiplication step that can be naively performed in $O(n^2c)$ time, yielding on the order of 10^7 operations on our testset, but is highly parallelizable. We distribute the problem across six process threads and calculate in parallel, as explained in section 6.1.1.

Lastly, we need to perform the HCS clustering. We must recursively compute the minimum spanning tree using Kruskal's algorithm: n times in the worst case (no set of nodes is highly-connected), but typically less. Kruskal can be implemented in $O(|E|\log|V|)$ time, where $|V|$ is the number of users and $|E|$ the number of edges between users: there are n^2 edges in the worst case, but usually much less, since we apply an edge pruning step (section 5.3.1. The minimum edge degree per cluster

Table 6.1: Scalability of each algorithm from a dataset with $n = 1000$ to a dataset with $n = 1,000,000$. Bold is best, italics is worst.

Worst-case number of operations	LDA $O(nkvi)$	k-means $O(nkdij)$	STS_γ $O(n^3 \log n)$
$n = 1000$	*10^{10}*	10^9	**10^6**
$n = 1,000,000$	10^{13}	**10^{12}**	*10^{15}*

can be kept track of during the spanning tree calculation. Therefore the total time complexity is $O(n|E|\log|V|) \approx O(n^3 \log n)$. Due to the difficulty of parallelizing this algorithm, it is the most time-consuming part of our STS_γ-clustering approach. Given 175 users, in the worst case we have $175^2 = 30625$ edges. With at most n recursive steps, the total expected number of operations is on the order of 10^6 operations. This is significantly less than the other baseline approaches; however, it needs to be performed in sequence with G_γ, which is also time-consuming of itself. Still, we can expect better performance compared to the baselines overall.

6.2.2 Scalability

The approximations of the number of operations required for each of the algorithms described in the previous section assumed datasets of sizes ≤ 1000, but this does not necessarily generalize to larger datasets. A larger dataset means a larger n term in our big-O time complexity notations. Let us consider a large dataset of $n = 1,000,000$. The estimated number of operations then become as shown in table 6.1.

As we can see, due to the n^3 term in the time complexity for STS_γ-clustering and the difficulty of parallelizing the HCS algorithm, it does not scale well to huge datasets. In order for it to scale, we will need to substitute the community detection algorithm for a faster one. A promising candidate for a fast, near-linear community detection algorithm for weighted graphs is *label propagation*, described by Raghavan et. al. [52]. This algorithm can find community structures in large-scale networks in $O(n + n^2)$ time. The implentation of this algorithm is outside the scope of this thesis, and left as future work.

6.2.3 Benchmarks

We have conducted a number of profiling experiments in order to determine the effectiveness of enabling parallel processing, either using parallel requests to the server and DBpedia Spotlight for ontological expansion, or splitting the load across multiple Web Workers for calculating the scoped topical similarity graph in a distributed way. Additionally, we test the performance of STS_γ-clustering against LDA and k-means baselines. All experiments are performed on an Intel Core i7 4790k @ 4Ghz with 16 GB of RAM. This CPU has 4 physical cores and 8 logical cores, through hyper-threading.

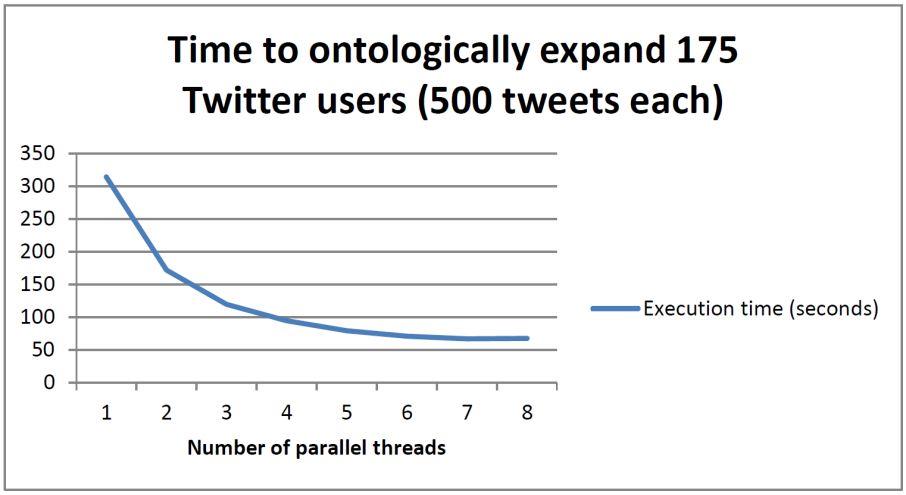

Figure 6.3: A plot of the time it took to ontologically expand 175 users with 500 tweets and obtain their $\text{cf}_{i,c}$ maps.

Parallel ontological expansion First, we profile how much time it takes to ontologically expand 175 Twitter users with 500 tweets each and obtain their class frequency maps given different numbers of processing threads. In practice, this means that we make several requests to the server API in parallel: each request results in a user and his tweets being taken from the caching database and the text content concatenated and sent to a (multi-threaded) DBpedia Spotlight instance. In our setup, Spotlight runs on the same machine as the backend code – we argue that the time spent inserting and retrieving users and tweets from the database is negligible given no additional load. The results are plotted in figure 6.3: each data point is the average processing time of five attempts. We see that for 7 threads, we get the best result: it takes 66.9 seconds to process and obtain the class frequency maps of all 175 users. For 8 threads, this actually grows slightly, as we run out of threads here (the backend code also needs a thread) and suffer some overhead.

Parallel scoped topical similarity graph In figure 6.4, we have plotted how much time is took to create G_γ of 175 Twitter users with 500 tweets each, yielding trait vectors of roughly a few thousand classes each. Settings for our clustering method are as we define them in section 7.3.4. Each data point is the average amount of time it took out of five test runs, for different numbers of parallel Web Workers. We split the computation by assigning each Worker an even share of the possible pair combinations between users (there are $\frac{n^2-1}{2}$ such pairs: $STS_\gamma(u_i,u_j)$ is the same as $STS_\gamma(u_j,u_i)$). We can see from the results that, after 6 Workers, which takes 4.8 seconds, we see diminishing returns. The reason for this is, aside from the core limitation of the CPU, there is some computational overhead to spin up a new Web Worker, and clone the data to send to them. For a larger number of users, more Workers may be useful.

Comparison to baselines Now we will compare the computation time of STS_γ-clustering against our main evaluation baselines, LDA and k-means (see section 7.2.2

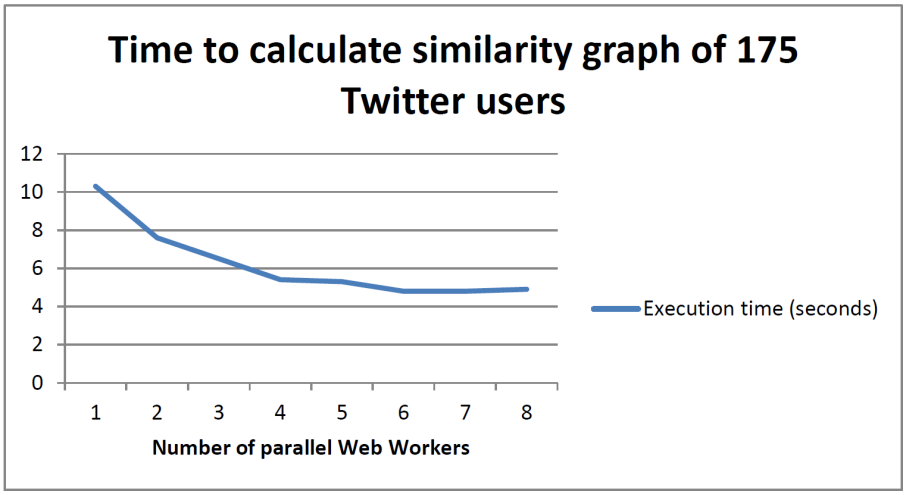

Figure 6.4: A plot of the time it took to create the STS_γ graph G_γ for 175 users with 500 tweets.

Table 6.2: Execution times for clustering 175 Twitter users with 500 tweets each into topics.

Execution time	LDA	k-means	STS_γ
Time (seconds)	87.1s	75.3s	70.6s

of the evaluation chapter). We again measure the time it takes to cluster our test set of 175 Twitter users from start to end into 11 topics. For STS_γ-clustering, we measure the time it takes to execute our method from start to end, using 7 threads for ontologically expansion and 6 Web Workers for calculating the similarity graph. The rest of the calculations, including HCS clustering, are single-threaded. We set γ to 0. For LDA, the MALLET package includes a fast, state-of-the-art parallel implementation of the algorithm, with default settings (1500 iterations), and the number of topics set to 11. For k-means, we implement the standard Lloyd's algorithm to cluster the STS_0 graph into 11 clusters. We again take the average time of five runs. The results are displayed in table 6.2. We see that our method is slightly faster than both LDA and k-means. The difference in execution time is in line with the complexity approximation conducted in section 6.2.1.

Chapter 7

Evaluation

In this chapter we describe the evaluation of our approach. First, we describe our target keywords, users and ground truths that we have prepared, in section 7.1. This is followed by an explanation of our experimental methodology; that is, the used evaluation metrics and baselines we compare our methods to, in 7.2. In 7.3, we describe how we derive parameter settings and calibrate the topic scope by tuning the hyperparameter configuration of our second method. Section 7.4 describes the experimental results of the timeline analysis, topical similarity calculation, scoped topic clustering and hierarchical clustering, respectively. We end this section with a discussion of the results, in section 7.5.

7.1 Ground truths

We now explain the ground truths we created in order to evaluate our methods. First, for timeline analysis for user recommendation, we examine four different keywords, for which we mine tweets over five consecutive days. We evaluate how accurately we can recommend users consistently interested in these keywords. We do this by comparing the ranking we obtain with a relevance map of users for which we manually checked whether each user was interested in the keyword or not.

There are two main aspects of our approach that we must evaluate: (1) the quality of the topical similarity calculation within a noisy Social Web context; and (2) the quality of the topic clustering, i.e. whether discovered topics are appropriate (content, number and size) for the selected scope or full hierarchy, and whether the topic labels are correct. We perform two different experiments, and therefore have two separate types of test sets: for (1), relevance maps, expressing relevance of a number of semi-random users to a known target user; and for (2), ground truths of users divided into topics and subtopics that we identified beforehand for Twitter, as well as an existing dataset of categorized newsgroup posts. For Twitter, we want human users that tweet actively and have enough content that we can process, so we only choose those users that tweet in English, have at least 500 tweets, and, excluding seed users, have between 100 to 10,000 followers and friends.

7.1.1 User recommendation: keywords mined

We have mined the Twitter search API over the course of five days, from Dec. 15th, 2012 to Dec. 19th, 2012. We gathered all tweets that contained any of the following keywords: *mars rover* ∪ *curiosity rover*, *malware*, *nuclear power*, and *genetically modified*. For *mars rover*, we take the union with keyword *curiosity rover*, as they refer to the same thing and are both used often. For this keyword, we consider tweets concerning Mars or space probes in general relevant to the topic as well. For *malware*, we also consider viruses, software vulnerabilities, etc. to be related. We feel that this reflects a real-world scenario, as a user searching for "malware" will likely be using this term as a catch-all for cyber security related topics, and not just the fairly narrow subject of "malicious software". For *nuclear power*, we aim to cover topics related to nuclear energy. Keyword *genetically modified* is used to catch topics related to genetically modified organisms (GMOs) and food. The evaluation will be performed on these four keywords separately.

7.1.2 Scoped topical similarity: user relevance maps

We prepare two sets of 100 Twitter users each. Each set has one *seed user* with a clearly identifiable interest that was manually determined by the authors by checking their Twitter profiles. The remaining 99 users are collected semi-randomly from their immediate follower neighborhood. We gather direct followers, followers of those followers, and so on, in a breadth-first fashion until we have enough users that satisfy our constraints. We pick one seed user that tweets consistently about **iOS development** (@iOS_blog), and one seed user that tweets consistently about **cars** (@CCCManhattan). For the remaining 99 users of both sets, we manually determine relevance of the user's timeline to the seed user, assigning a score of 0, 1 or 2, as follows:

0: This user is irrelevant; the user does not have tweets related to the topic

1: This user is somewhere in between; in this category we include users that post some tweets related to the topic, but also a considerable number of unrelated tweets

2: This user is very relevant; most of the user's tweets are directly related to the topic

7.1.3 Topic clustering: ground truths

To evaluate the topic clustering results, we prepare a primary ground truth consisting of 175 Twitter users. The ground truth contains 4 main topics, which are further subdivided into 11 subtopics. Each user belongs to one main topic and one subtopic. Since our method takes slices of a topic hierarchy, we expect it to be able to discover both clusterings, depending on the setting of the topic scope parameter γ. The topics are distributed as follows:

- **Computer science** (50 users): *iOS development* (20 users), *Web development* (15 users), *Data Science* (15 users)

Table 7.1: The 20 newsgroups dataset (20 topics over 6 subject matters).

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

- **Sports** (40 users): *Soccer* (10 users), *Football (American)* (10 users), *Golf* (10 users), *Cricket* (10 users)

- **Cars** (40 users): *Cars* (40 users)

- **Politics** (45 users): *US politics* (15 users), *UK politics* (15 users), *Australian politics* (15 users)

Note that for topic cars, we could not distinguish any clear further subtopics, therefore we expect the same result for main and subtopic. For each user, the authors made sure that the users were primarily interested in the subtopics defined above by manually checking their recent tweets. To make this process less laborious, some users were collected from follower neighborhoods of well-known seed users for each subtopic (e.g. *@BarackObama* for US politics, *@BBCTopGear* for cars, etc.). Others were found through the keyword search function on Twitter. The result is a well-defined ground truth that forms a good basis for comparison. The clusters have different sizes, which was a deliberate choice; it allows us to evaluate performance when dealing with asymmetric cluster sizes.

We prepare a second ground truth based on the 20 newsgroups dataset by Ken Lang [34], which is a commonly used dataset in the field of document clustering [53]. This dataset consists of over 20,000 posts across 20 newsgroups. These 20 newsgroups can be roughly divided into 6 main subject matters (see table 7.1). Due to performance limitations, we take a subset of 1800 randomly selected posts, evenly divided over the 20 newsgroups (90 posts per group). This dataset differs from Twitter data in that the text is generally well-structured and grammatically correct. However, posts can be short: in the extreme case, a post may consist of only a single line. Our approach requires an appropriate amount of context information (a portion of a user's timeline) for ontological expansion, which may not be available here, so we expect this to impact performance of our method. For a more even comparison given this limitation to our approach, we also prepare a second subset of all posts larger than 10.0 kilobytes in size (there are 239 such posts).

7.2 Experimental methodology

In this section, we describe the evaluation metrics we employ and the baseline approaches we will compare to. We evaluate user recommendation rankings and scoped topical similarity using the normalized Discounted Cumulative Gain (nDCG) [29] ranking based on the recommendation ranking and user relevance maps described in the previous section, and topic clustering based on overlap between the clustering results and ground truths (we calculate accuracy and Matthew's Correlation Coefficient, or *MCC* [38]) and a graph entropy-based measure (normalized mutual information, or *NMI*). We subsequently compare the obtained results to relation-based approaches for user recommendation. For STS, we compare to the results of calculating topical similarity using a traditional tf-idf-weighted term vector model. For scoped topic clustering, we compare to latent Dirichlet allocation (LDA) and k-means clustering, and for hierarchical clustering to hLDA and hierarchical k-means.

7.2.1 Evaluation metrics

To evaluate a ranking of users given relevance maps, we calculate the DCG according to the formula

$$\mathrm{DCG_k} = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i)}, \tag{7.1}$$

where k corresponds the top-k ranked users based on taxonomical similarity for the user recommendation method, and based on unscoped topical similarity STS_0 for the topic clustering method. rel_i is the relevance score assigned to the user at rank i. Since we want to evaluate for different top-k, we will obtain different $\mathrm{DCG_k}$ scores for different k, therefore we need to use the normalized version of the DCG, nDCG:

$$\mathrm{nDCG_k} = \frac{DCG_k}{IDCG_k}. \tag{7.2}$$

Here IDCG refers to the *Ideal* DCG, i.e. the maximum possible DCG until position k:

$$\mathrm{IDCG_k} = 2 + \sum_{i=2}^{k} \frac{2}{\log_2(i)}. \tag{7.3}$$

For topic clustering, since our method is already based on similarity, any cluster similarity-based internal evaluation would give biased results. Therefore, we apply an external clustering evaluation by representing our ground truths and cluster graphs as truth tables containing *true positives* (TP), *false positives* (FP), *true negatives* (TN), and *false negatives* (FN). Let $T_k(u_i)$ be the k-th topic cluster containing some user u_i. Then we can define the contingency table as in table 7.2. Using this table, we can calculate the *precision*, *recall* and F_1-*score*, as well as the Matthew's Correlation Coefficient (*MCC*) [38] for a topic clustering result over a collection of users U as follows.

$$Precision = \frac{TP}{TP+FP}, \quad Recall = \frac{TP}{TP+FN}, \quad F_1\text{-}score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{7.4}$$

Table 7.2: Determining true/false positives/negatives between the clustering result and the ground truth.

		Ground truth	
		$T_k(u_i) = T_k(u_j)$	$T_k(u_i) \neq T_k(u_j)$
Clustering result G'_γ	$T_k(u_i) = T_k(u_j)$	TP	FP
	$T_k(u_i) \neq T_k(u_j)$	FN	TN

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{7.5}$$

Note that precision, recall and F_1 do not take into account true negatives. This makes it susceptible to bias, since the TN:FN ratio gets increasingly skewed towards more TNs the more clusters we have. The MCC accounts for this bias by giving FPs and FNs equal weight to TPs and TNs, taking on a value between -1 and 1, where a value of 0 means the result is no better than random; towards -1 increasingly worse than random; and towards 1 increasingly better than random (with 1 being a perfect score).

Additionally, we evaluate according to an entropy-based measure: *normalized mutual information* (*NMI*). This is a measure of the difference between the number and sizes of the ground truth clusters with the resulting clusters, without regard for their content (whether the right users have been assigned to the right clusters). It is calculated as follows:

$$NMI(X;Y) = \frac{\sum\limits_{y\in Y}\sum\limits_{x\in X} p(x,y)\,\log\left(\frac{p(x,y)}{p_1(x)p_2(y)}\right)}{\sqrt{\Big(-\sum\limits_{x\in X} p(x)\,\log\,p(x)\Big)\Big(-\sum\limits_{y\in Y} p(y)\,\log\,p(y)\Big)}} \tag{7.6}$$

Here, the X and Y variables are substituted with representations of the ground truth and clustering result. For details, refer to [66].

In order to evaluate full hierarchical clustering, we require some way to evaluate each level of the hierarchy as a whole rather than separately in order to get an accurate assessment of algorithm quality. We borrow the *hierarchical F-score* as introduced in [30], and further recommended as a generally usable measure to evaluate hierarchical classification algorithms in the survey presented in [59].

$$hP = \frac{\sum_i \mid \hat{P}_i \cap \hat{T}_i \mid}{\sum_i \mid \hat{P}_i \mid}, \quad hR = \frac{\sum_i \mid \hat{P}_i \cap \hat{T}_i \mid}{\sum_i \mid \hat{T}_i \mid}, \quad hF = \frac{2 \cdot hP \cdot hR}{hP + hR} \tag{7.7}$$

Here $\hat{P}_i$ is the set consisting of the most specific topic(s) predicted for user i and all their ancestor topics and $\hat{T}_i$ is the set consisting of the *true* most specific topic(s) of user i and their ancestor topics. Although designed for the evaluation of document classification (matching to an existing class hierarchy) rather than clustering (creating an entirely new class hierarchy), the measure is flexible enough to be adapted for

clustering by regarding each topic in our two-level ground truths as the target classes we expect to see when clustering data hierarchically. This means that $\hat{T}_i$ will always have size 2. If a result cluster consists of more than one of the ground truth topics, we consider this a new class not in $\hat{T}_i$; we continue clustering hierarchically until cluster topics closely represent the topics defined in the lower layer of the ground truth in the result. To judge this, we look at the top terms identified per topic for hLDA, and the top labels generated for STS_h.

7.2.2 Baseline approaches

Timeline analysis We will compare the taxonomical timeline analysis method with different values for similarity threshold θ_s to each other, as well as to several baseline approaches to showcase the merit of the proposed method over other methods. The following baselines are applied:

- **Tweet count score (TC)**: rank users simply based on the amount of tweets that contained the keyword they posted.
- **User influence score (UI)**: rank users based on the user influence score described in section 4.1.
- **TURKEYS score**: rank users based on the TURKEYS score from equation 4.1.

Timeline analysis is performed on top of the user ranking based on TURKEYS scores. We try $\theta_s = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Note that the higher the similarity threshold θ_s, the higher the number of users that we discard from the ranking, meaning we need to process more than just the top 20 users even in the case of $k = 20$. We decide to process at most the top 50 users. Usually, this does not lead to a sufficient amount of resulting users for higher thresholds, so we stop at 0.5. Note, however, that even for $k = 5$, we found no performance improvements beyond $\theta_s = 0.5$ given the keywords that we experimented with.

STS and topic clustering We compare our STS_γ similarity calculation to two baselines: (1) traditional tf-idf-based cosine similarity, and (2) a similarity calculation based on the simple taxonomical similarity of classes between users [62]. For (1), we take all words in all tweets per user (after basic filtering), calculate the document vectors with tf-idf component weights, and calculate the cosine similarity-based ranking compared to the seed users. For (2), taxonomical similarity s_{tax}, we apply the following formula between the seed user u_{seed} and each subject user u_i:

$$s = \frac{\sum\limits_{c_j \in \mathrm{cf}_{seed} \cap \mathrm{cf}_i} \min\{\mathrm{cf}_{seed}(c_j),\ \mathrm{cf}_i(c_j)\}}{\sum\limits_{c_j \in \mathrm{cf}_{seed}} \mathrm{cf}_{seed}(c_j)}. \tag{7.8}$$

That is, we sum the class frequencies for all classes that u_{seed} and u_i have in common, and then divide that by the sum of all class frequencies of all classes in u_{seed}.

We evalute four versions of our STS_γ-based topic clustering, each using different types of ontology information: Wikipedia resources only ($STS_{\gamma,res}$), DBpedia/Schema.org

classes only ($STS_{\gamma,DBpedia}$), YAGO types only ($STS_{\gamma,YAGO}$) and all types combined (STS_{γ}). We compare all versions to four baselines:

1. *Random clustering*: we take the average of 100 random clusterings of the data.

2. *Latent Dirichlet allocation (LDA)*: we apply LDA on text content (after basic filtering). For Twitter, each user profile is considered one document. Since LDA requires us set a number of topics, we calculate the results for 2 to 24 topics. We cluster users into the topic to which LDA assigns them the highest probability. We use the state-of-the-art LDA implementation that is part of the MALLET package [39], with 800 iterations but otherwise default settings. We use 800 iterations since this finishes in roughly the same calculation time for 175 Twitter users as our implementation of STS_{γ}-clustering on our environment (around 67 seconds).

3. *Twitter-LDA* [74]: we apply the Twitter-optimized version of LDA as implemented and described in [51] for an evaluation compared to the state-of-the-art in Twitter topic modeling. We apply the same hyperparameter settings as regular LDA, and maintain 800 iterations.

4. *k-means clustering*: we cluster the STS_{γ} graph G_{γ} using standard k-means [21]. As with LDA, we calculate the results for the number of topics K ranging from 2 to 24 topics. For each K, we iterate 10 times and take the result for which the within-cluster sum of squares (WCSS) is maximum (since we maximize on cosine similarity):

$$\arg\max_{\mathbf{T}} \sum_{k=1}^{K} \sum_{\mathbf{t} \in T_k} \|\mathbf{t} - \mu_k\|^2 \tag{7.9}$$

   Here, $\mathbf{T}$ is the set of K topic clusters, and μ_k is the centroid vector of all trait vectors $\mathbf{t}_i$ of users $u_i \in U_k$ that belong to topic cluster $T_k \in \mathbf{T}$.

Lastly, we evaluate hierarchical clustering, where we apply our recursive topic clustering strategy (algorithm 1) to generate a full topic hierarchy. We compare STS_h against two baselines:

1. *Hierarchical latent Dirichlet allocation (hLDA)*: similar to the per-layer evaluation, we use the state-of-the-art hLDA implementation that is part of the MALLET package [39] to infer a topic hierarchy, with 800 iterations (we are not aware of any Twitter-specific hierarchical LDA algorithm). hLDA requires a hierarchy depth to be pre-determined; we set this to 3 (the first level of hLDA consists of the full data, so 3 levels matches our ground truth).

2. *Hierarchical k-means*: we cluster the data hierarchically using the k-means algorithm in a recursive fashion, with the expected number of topics at each level as value for K (for the Twitter dataset, $K = 4$ for the first level, and $K = 3$ or $K = 4$ depending on the number of subtopics expected). Since k-means works stochastically, we iterate 10 times and average the result.

7.3 Parameter selection

There are a large number of parameters to set for both of our methods, and it is unclear which settings will yield the best results. We want to derive optimal parameter settings that will provide good results on unseen data with high likelihood. To keep dimensionality of this optimization problem in check, we decide to tune the parameters for the named entity recognizer and for the clustering method separately. Recall that for entity recognition, we must tune the confidence parameter and the concatenation window size. We perform a simple grid search over a discretized range of values to find which parameter values give the best results on average when checked against the nDCG relevance rankings. For topic clustering, we must first decide on a neutral topic scope (an appropriate level of topic generality given $\gamma = 0$), then calibrate the remaining variables around this by applying a hyperparameter optimization approach, using grid search and a random selection-based cross-validation on the cluster ground truth for regularization.

7.3.1 Named entity recognition parameter tuning

For NER, key parameters we need to decide on are (1) the confidence value of the entity recognizer (DBpedia Spotlight); and (2) the size of the windows of concatenated tweets. We determine a good confidence value using a simple grid search optimization. We calculate the top-k nDCG rankings on both relevance map-based datasets in terms of their seed users, for $k \in \{3, 5, 10, 25, 50\}$ and for a range of values between 0 and 1 for the confidence parameter. For this initial experiment, we concatenate 10 tweets at a time before applying NER. The results are shown in figure 7.1.

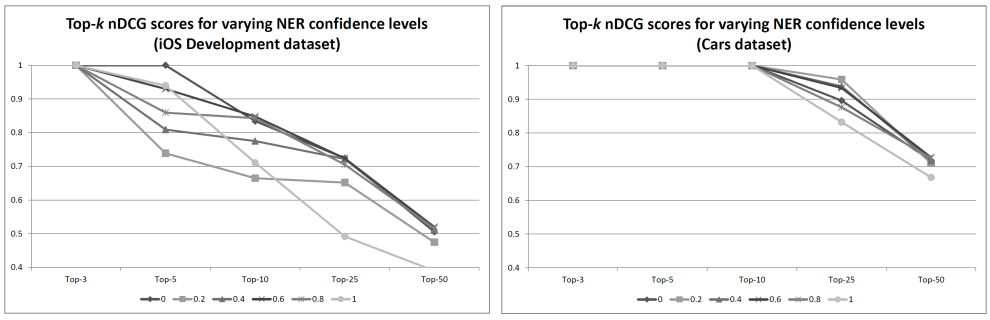

Figure 7.1: nDCG scores for different NER confidence settings for the iOS development and cars datasets.

We see that the results are relatively close to each other. For top-50, which should be least affected by random noise, all settings perform roughly equal with the exception of the highest setting of 1. Given these results, we decide to keep the confidence value at 0 – aside from giving the best results on average (by a small margin), in situations where we have little content available the higher settings may no longer yield enough classes to determine topic clusters accurately.

Additionally, we must decide on window sizes for concatenating tweets (recall that the entity recognizer works by leveraging the surrounding context of source terms). We

again experiment on both datasets, this time varying tweet window sizes between 3, 5, 10, 25 or 50 tweets. Results are shown in figure 7.2. Again, we see only minor differences in the results. We set the tweet window to 10 tweets, which performed best at larger sample sizes.

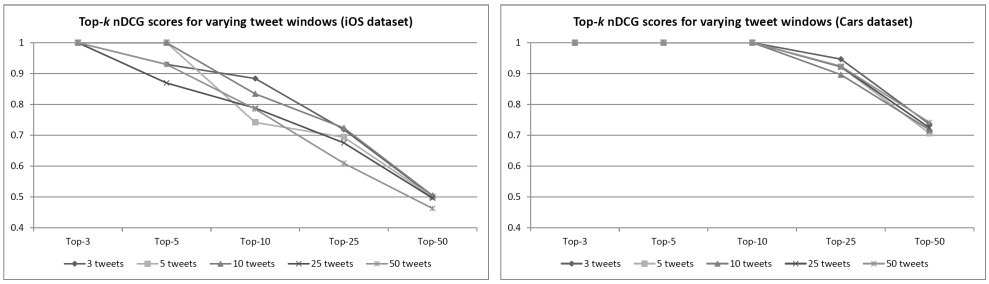

Figure 7.2: nDCG scores for different tweet window sizes for the iOS development and cars datasets.

7.3.2 TURKEYS and class pruning settings

For calculating the TURKEYS score in the evaluation of recommendation based on user relations, we need to define how to weight tweet count score TC vs. user influence score UI (equation 4.1). We will fix the weight w to 0.5, so that both scores are weighed equally, as this weight was shown to perform the best on average in previous experiments [46].

For the pruning of classes, described in section 4.2.2, we need to set a parameter t as a cutoff threshold for mismatch removal, and a parameter p for the filtering of generic classes. We set t to 0.1% of the total number of class occurrences, and p to 100%, meaning there should be at least as many direct sub-class occurrences as the number of occurrence of their super-class as a condition to remove this super-class. Again, experimentation showed that these values resulted in reasonable topic representations (not too many or too few classes, not too broad or too specific classes), but a more detailed investigation is needed in order to find optimal values.

Lastly, we need to set the similarity threshold θ_s to decide tweeting consistency of users for the final taxonomical analysis step. Given that the evaluation of this step is the focus of this work, we will experiment with different values to determine which provides the best result on average.

7.3.3 Class frequency distribution and trait cut-off threshold θ

The combined distribution of class frequencies over our ground truth of 175 users with 500 tweets each is plotted in figure 7.3. As is evident, class occurrences are skewed at the tail-end of the distribution, with almost 10% of all unique classes occurring 10 or less times. Classes with small occurrence numbers lead to negligible contributions to the cosine similarity, therefore consider it safe to prune the trait vectors up to a certain *trait cut-off threshold* θ to reduce the sparsity and dimensionality of data. Which value of θ would be best is not immediately clear. Along with the previously introduced τ and

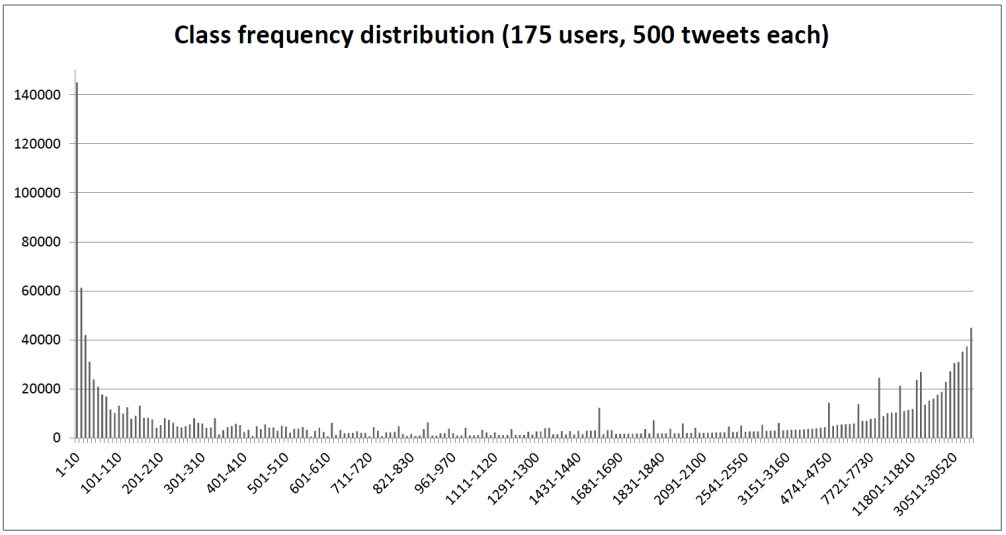

Figure 7.3: The combined class frequency distribution for 175 Twitter users with 500 tweets each. There were 1,535,108 classes found in total.

α, this variable is the last of three dependent variables that we set by hyperparameter optimization.

7.3.4 Hyperparameter optimization and scope calibration

We apply a hyperparameter optimization approach to calibrate θ, τ and α around a neutral topic scope $\gamma = 0$. This is necessary since all four variables are dependent on one another. Since our dataset is not that large, and we have only three dimensions, it is feasible to apply a grid search on the parameters – that is, check each combination exhaustively within a reasonable discrete range of possibilities. We can make surface plots to visually determine appropriate values. We test the following parameter ranges:

- *Trait cut-off threshold*: $\theta = \{0.0, 0.001, ..., 0.01\}$
- *Similarity threshold*: $\tau = \{0.0, 0.01, ..., 0.1\}$
- *Highly-connectedness parameter*: $\alpha = \{2, 3, 4, 5, 6\}$

Next, we need a data set and some fitness function to optimize on. We choose to use our cluster ground truth of 175 users, testing the MCC score of the resulting cluster graphs compared to the 11 subtopics we defined in the ground truth. This means that our method gets calibrated to the extent that $\gamma = 0$ will detect topic clusters roughly corresponding to the scope expressed by these 11 subtopics. This choice is made for both intuitive and practical reasons. Intuitively, the subtopics represent a mid-level position in the hierarchy: given a topic such as Football, it is easy to come up with topics that are more general ("Sports" $\rightarrow$ "Activity") and more specific ("Premier League clubs" $\rightarrow$ "Arsenal F.C."). Practically, we need a concrete ground truth to perform the calibration, and our 11-topic testset is the most comprehensive we have available.

To avoid overfitting, we must incorporate some form of regularization, such as cross-validation. However, since our approach does not involve a training phase, we cannot split the data into a training set and test set as with traditional cross-validation. Therefore, we regularize using random-fold cross-validation in the following way:

1. For each parameter combination, do the following 50 times:
    a) Take 7 out of 11 sub-topics from the ground truth uniformly at random
    b) For each sub-topic, take between 50-100% of the users assigned to that topic uniformly at random
    c) Apply the approach from start to end to these users and calculate the MCC of the resulting cluster graph G'_0 in comparison to the reduced, 7-topic ground truth
2. Take the average of the 50 iterations as the final MCC value

In figure 7.4, we have plotted the resulting MCC value surfaces for all combinations of θ and τ, with each graph keeping α fixed at one of 2, 3, 4 or 5. For $\alpha = 6$, the results were very clearly substandard to the other parameters, so we omitted the surface plot here. It is now important to pick a combination of values that has a high probability of yielding a good result when dealing with new, unseen data. In other words, we must take care to choose values so that the area around combinations of θ and τ – and from one α to the next – is of consistently good quality. This leads to a higher probability that the results are reproducible. The values that best fit this condition are $\theta = 0.003$, $\tau = 0.07$, and $\alpha = 4$. These will be the values we will use in the coming experiments where we compare our approach to baselines and established approaches.

Standard deviation and standard error of folds Regularizing the parameter selection only works if the cross-validation is adequate given the size of the testset. Since our testset of 175 users is rather small, it is not immediately clear if our random-fold cross-validation is reliable. Recall that for each parameter combination, we take the average score of 50 iterations on random selections of topics and users. For a reliable method, we expect that scores between iterations do not fluctuate in a random or otherwise significant way. In other words, by calculating the average standard deviation and margin of error of the sampling, we can obtain a statistical measure of the reliability of the hyperparameter optimization.

We calculated the standard deviation σ of 50 iterations for all parameter combinations as described above. There are $11 \times 11 \times 4 = 484$ such parameter combinations. The resulting distribution function of standard deviations is plotted in figure 7.5.

In order to calculate the standard error from these standard deviations, we can construct the 95% confidence interval for the average standard deviation. The average standard deviation is best calculated by taking the square root of the mean of variances σ^2:

$$\overline{\sigma} = \sqrt{\overline{\sigma^2}} = \sqrt{0.0067} \approx 0.08 \tag{7.10}$$

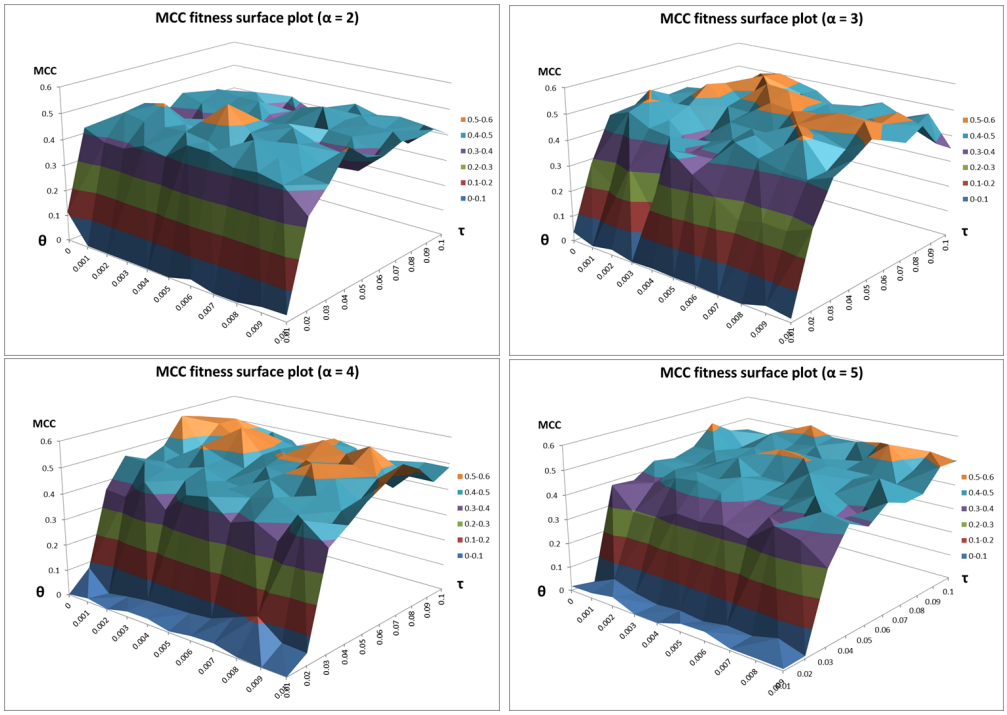

Figure 7.4: MCC surface plots for varying values of θ and τ. From top-left to bottom-right, $\alpha = 2$, $\alpha = 3$, $\alpha = 4$ and $\alpha = 5$. Each MCC data point is the average over 20 iterations with semi-random cluster selections.

It follows that the 95% confidence interval is $[-0.08 \times 1.96, +0.08 \times 1.96]$. See figure 7.6. This result means that we can say with 95% confidence that the resulting mean MCC score for a given parameter combination lies within approximately -0.157 and 0.157 of the true mean MCC score. Since MCC ranges from -1 to 1, this is a standard error of around 7.8%. This is a reasonable enough value to conclude that the hyperparameter optimization is adequate.

7.4 Experimental results

In this section, we will show the experimental results for each of the methods. Section 7.4.1 details the results for timeline analysis. For topic clustering, section 7.4.2 shows the topical similarity results and section 7.4.3 the hierarchical topic clustering results.

7.4.1 Experimental results: timeline analysis

First, we show the evaluation for enhancing user recommendation based on user relations with a taxonomical similarity approach. The results for the four keywords that we investigated are listed below. In case not enough users could be gathered for the taxonomical analysis approach, the result is left empty in the charts.

Figure 7.7 shows the result for keyword(s) *mars rover* $\cup$ *curiosity rover*. We can see a significant improvement in nDCG scores for the taxonomical analysis approach, especially for higher similarity thresholds. Although for a top-20 ranking, we can no

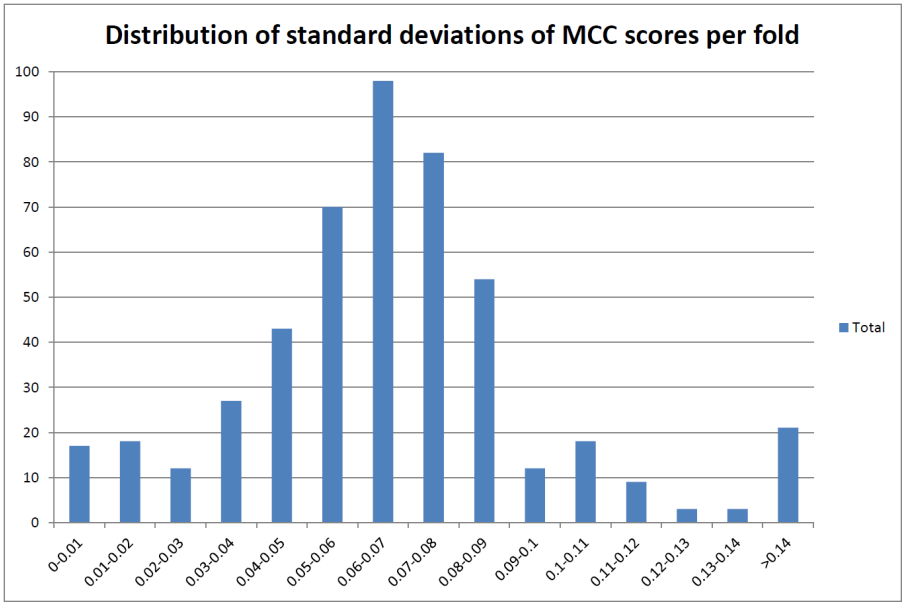

Figure 7.5: Distribution of standard deviations for each fold of the cross-validation. To save space, a long tail of outliers on the right side of the graph have been aggregated as > 0.14.

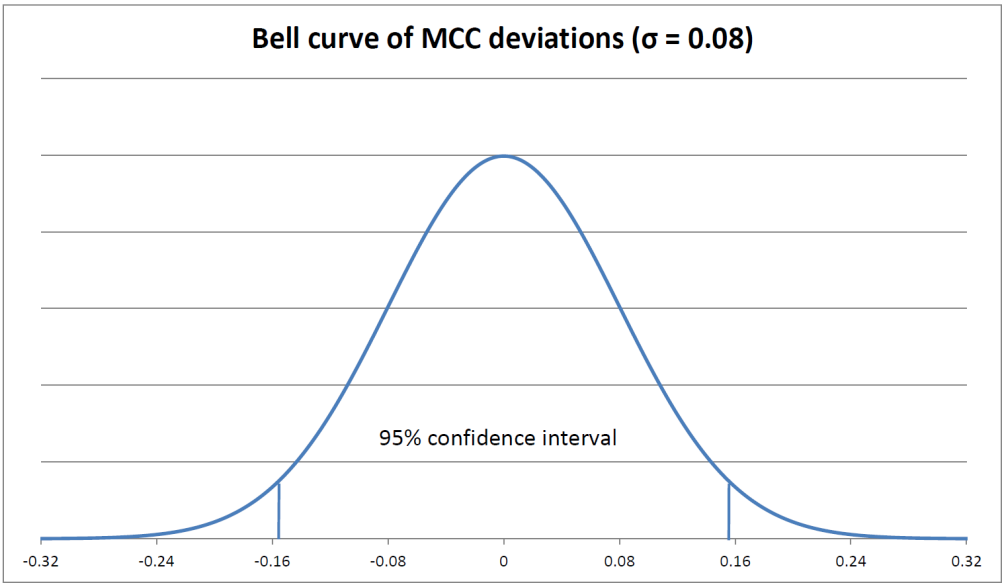

Figure 7.6: The 95% confidence interval of standard deviations. In other words, 95% of measurements fall within roughly -0.157 and 0.157 of the real mean MCC.

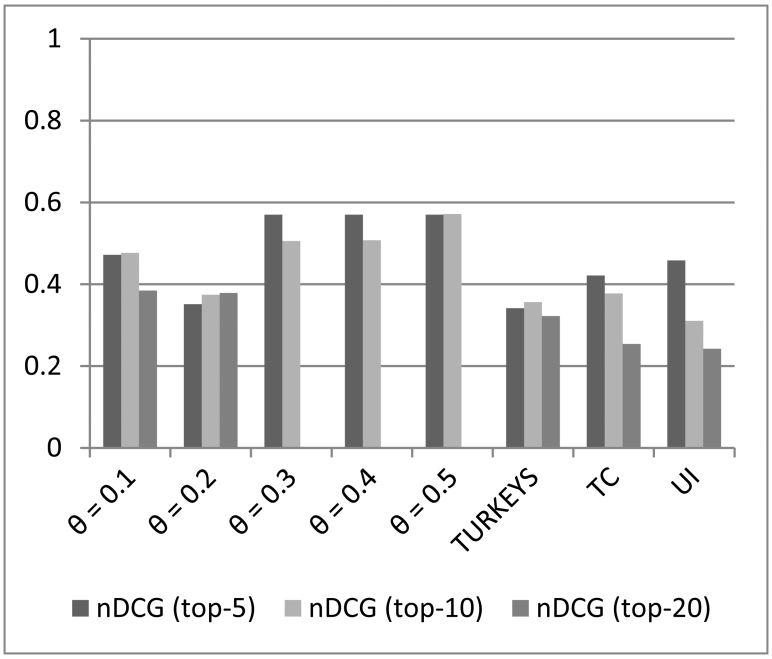

Figure 7.7: Results for keyword *mars rover* ∪ *curiosity rover*.

longer obtain a sufficient amount of users past $\theta_s = 0.2$, even $\theta_s = 0.1$ and $\theta_s = 0.2$ already outperform the baselines.

For the results of keyword *genetically modified* in figure 7.8, we see a peak in performance at a θ_s value of 0.2. For this keyword, it is more difficult to obtain a sufficient amount of users, as the topic of genetically modified foods and animals is much less focused than the mars rover, making it harder to concretely define a topic covering and determine whether two coverings are similar. Again, there are significant improvements to be seen for $k = 5$ and $k = 10$. For $k = 20$, however, tweet count performs best.

Results for keyword *malware* are shown in figure 7.9. Performance of the taxonomical approach is equal to the tweet count score for $k = 5$, and inferior to tweet count score and TURKEYS for $k = 10$ and $k = 20$. We found that the topic of malware is consistently captured into the same classes (most often occurring classes are *Software*, *ComputerSecuritySoftwareCompanies* and *ComputerCrimes*), making it easy to end up with enough users even at high thresholds. We discuss an explanation for the poor performance for this keyword in the next section.

Lastly, figure 7.10 shows the result for keyword *nuclear power*. We can see that all methods perform well in terms of absolute nDCG score. For $k = 5$ and $k = 10$, the proposed approach with θ_s at 0.1 or 0.2 is the best performer. For $k = 20$, tweet count score performs best again, although $\theta_s = 0.1$ comes very close. For this keyword, it is again difficult to obtain enough users for $k = 10$ and $k = 20$ at higher thresholds, for the same reasons as keyword *genetically modified*.

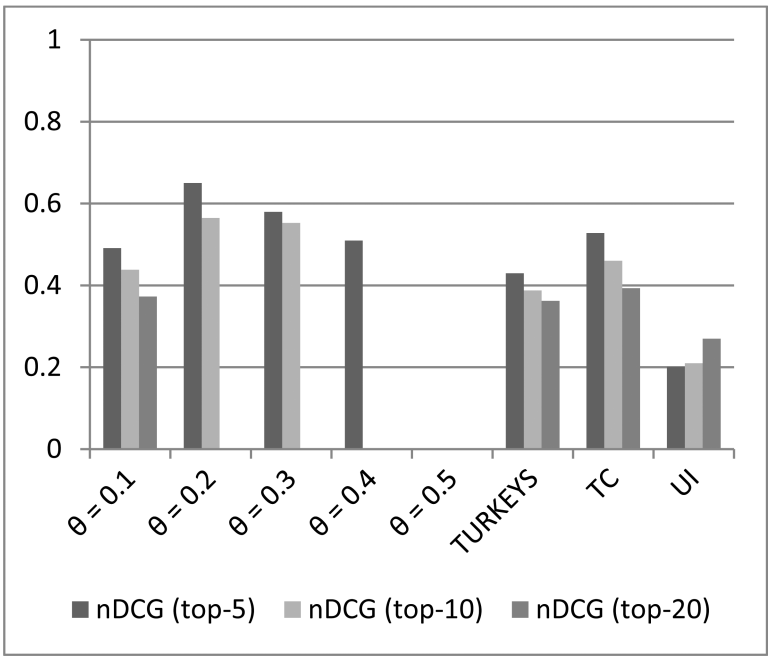

Figure 7.8: Results for keyword *genetically modified*.

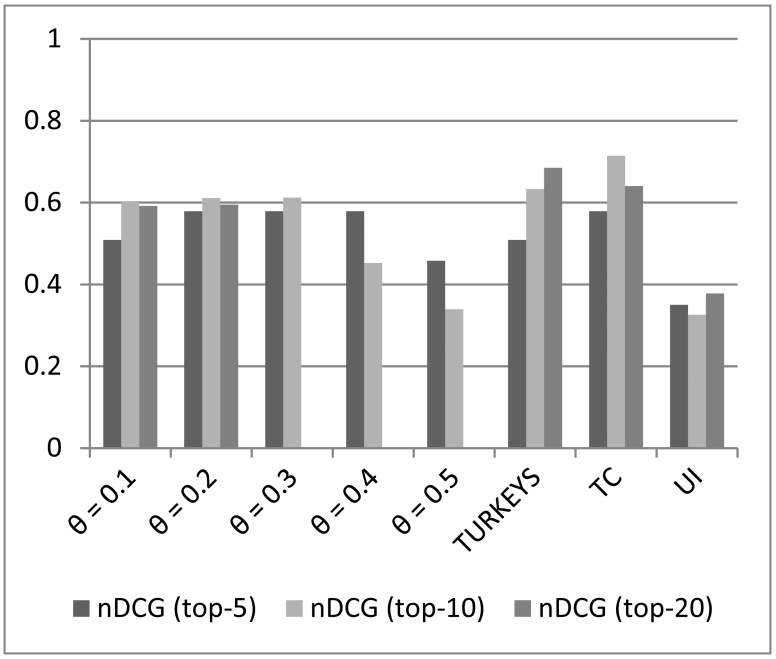

Figure 7.9: Results for keyword *malware*.

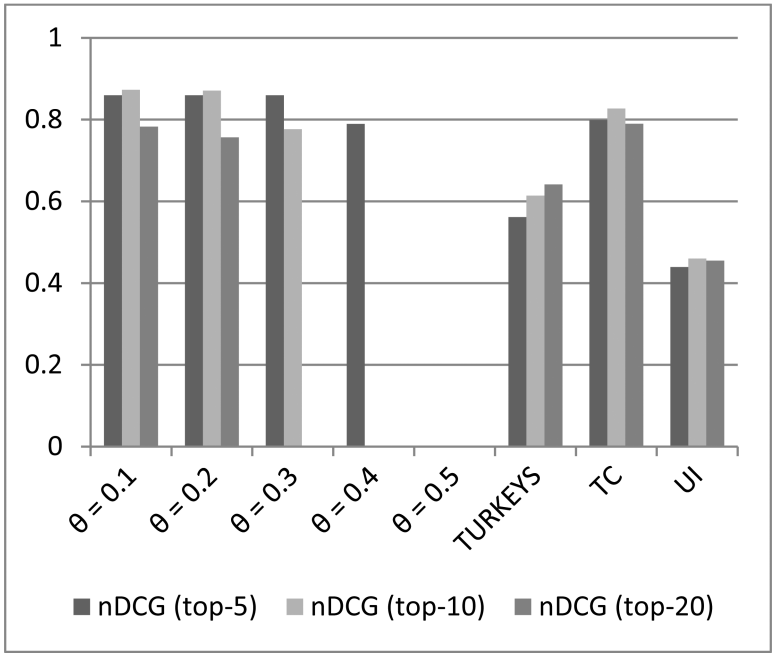

Figure 7.10: Results for keyword *nuclear power*.

7.4.2 Experimental results: topical similarity

We now evaluate the topical similarity portion of our approach in terms of two nDCG rankings to seed users with known interests (*iOS development* and *cars*), as explained in section 7.2.2. These evaluations will be similar to the entity recognition confidence and tweet window experiments, although this time we compare the quality of our STS_γ-based rankings to the two baseline approaches: (1) traditional tf-idf-based cosine similarity, and (2) taxonomical similarity of classes between users [62].

We again evaluate by comparing the top-k nDCG rankings, for $k \in \{3, 5, 10, 15, 20, 25\}$. Since the baseline methods do not incorporate topic scope, we keep γ fixed to 0 for a fair comparison. The results are plotted in figure 7.11. A consistent improvement of roughly 15% on average over the second-best baseline approaches (taxonomical for iOS development, tf-idf for cars) can be observed. We leave further interpretation and discussion of these results for the discussion section 7.5.

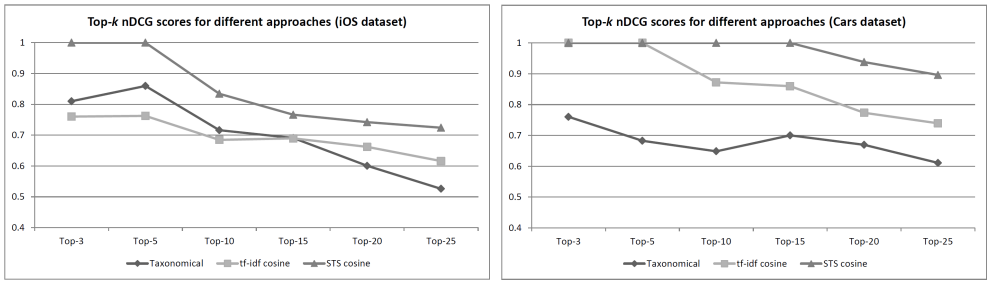

Figure 7.11: nDCG scores for different similarity calculation approaches for the iOS development and cars datasets.

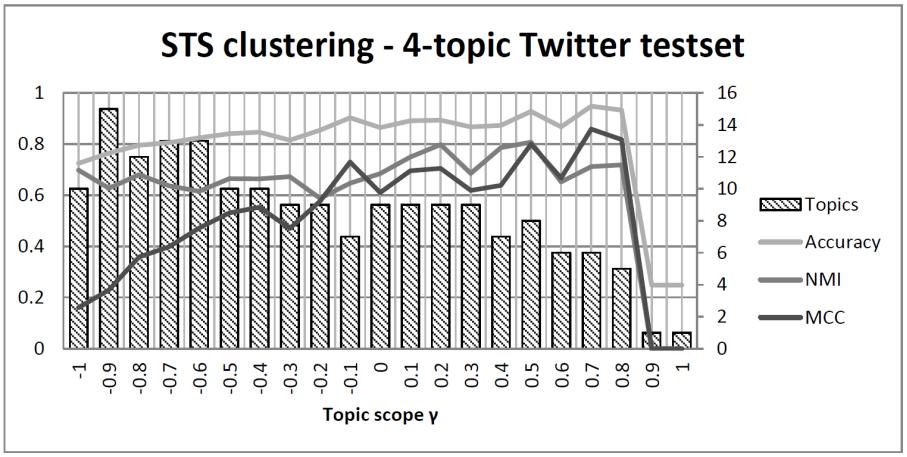

Figure 7.12: Results for STS_γ-clustering on the 4-topic Twitter ground truth.

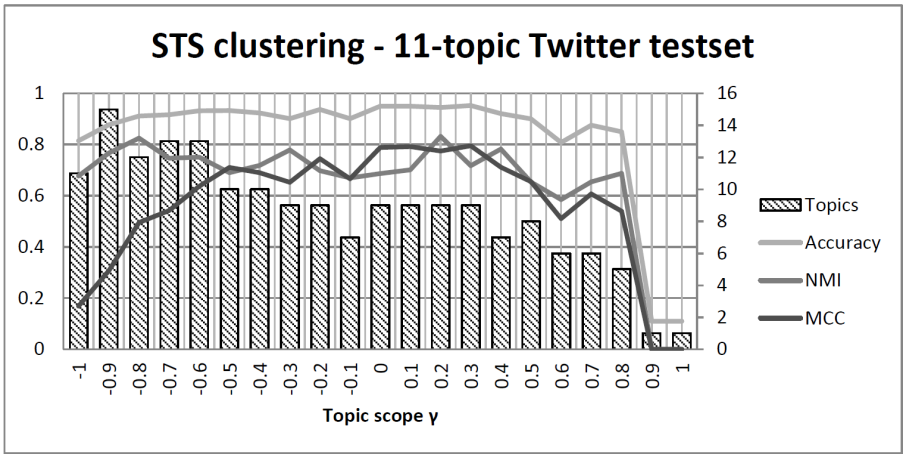

Figure 7.13: Results for STS_γ-clustering on the 11-topic Twitter ground truth.

7.4.3 Experimental results: topic clustering

In this section we detail the results for scoped and hierarchical topic clustering. First, we show the scoped clustering results we obtained for the Twitter dataset, followed by the results for the newsgroups datasets. We conclude with an evaluation of hierarchical clustering.

Twitter dataset We apply STS_γ-clustering for the Twitter ground truth, distinguishing between 4 topics and 11 sub-topics that we aim to cluster the same set of 175 users in. The goal is to be able to cluster into the right 4 main topics *or* the right 11 sub-topics depending on our desired topic scope setting. The $P/R/F_1$, NMI and MCC scores are calculated for a range of topic scope parameter γ values between -1 and 1, in increments of 0.1. The results are compared to random clustering, LDA and Twitter-LDA-based clustering, and k-means clustering.

For STS_γ, for both the 4-topic and 11-topic testsets, a progression of the F-score, NMI and MCC metrics (lines) and the number of topics they yielded (columns) for different topic scopes (x-axis) is plotted in figures 7.12 and 7.13. The same metrics for LDA, Twitter-LDA and *k*-means, trying topic selections from 2 to 24, are plotted in figures 7.14, 7.15 and 7.16, respectively. Table 7.3 shows a summary of the best values with corresponding settings that were obtained for each method. This summary includes the best results for all 4 versions of our method: resources-only ($STS_{\gamma,res}$), DBpedia/Schema.org classes only ($STS_{\gamma,DBpedia}$), YAGO types only ($STS_{\gamma,YAGO}$) and all types combined (STS_γ. Tables 7.4 and 7.5 list topic labels discovered for the best results: terms with highest probability per topic for Twitter-LDA, and top class labels calculated as described in section 4.3 for STS_γ-clustering. *k*-means clustering does not provide topic labels.

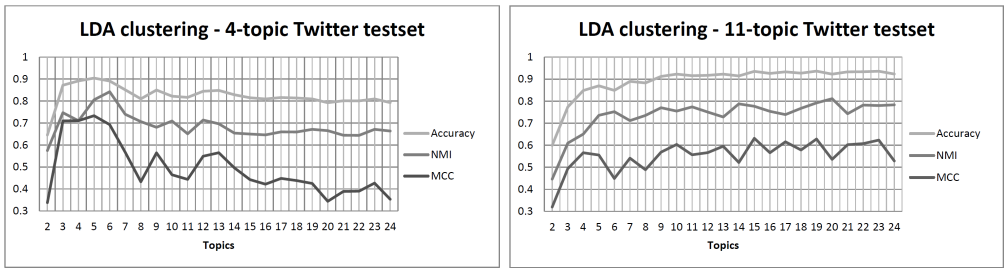

Figure 7.14: Results for LDA-based clustering on the Twitter ground truth.

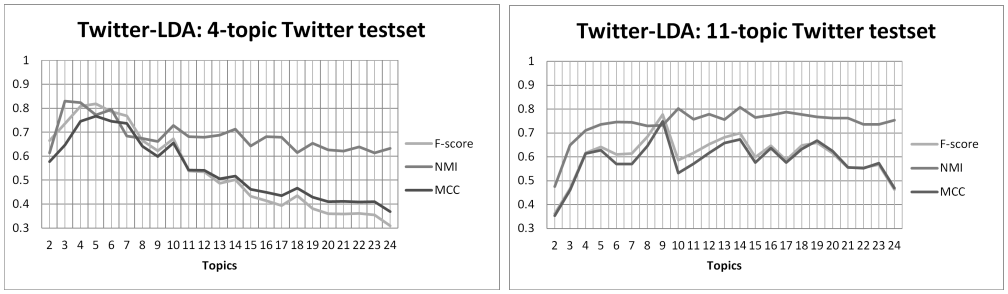

Figure 7.15: Results for Twitter-LDA-based clustering on the Twitter ground truth.

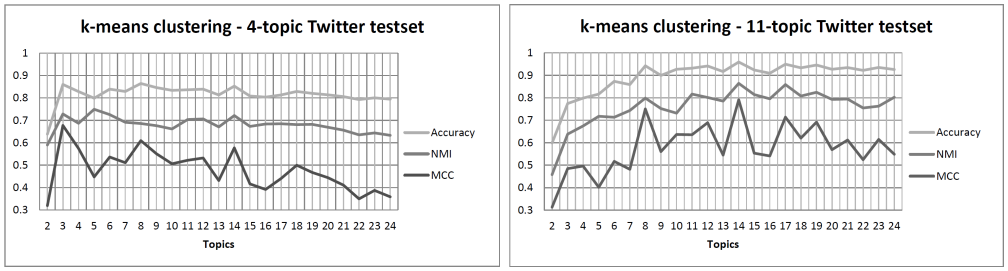

Figure 7.16: Results for *k*-means clustering on the Twitter ground truth.

Looking at the results in figures 7.12 and 7.13, we observe an expected progression in terms of the number of topics detected: the higher the topic scope, the less topic clusters we find. For the 4-topic testset, the best result in terms of the MCC score

Table 7.3: Summary of optimal results for the random clustering baseline, LDA, Twitter-LDA, *k*-means and our STS_γ-clustering method with different ontology class selections.

175 Twitter users, 4-topic testset

	Precision	*Recall*	*F-score*	*NMI*	*MCC*	*Topics*
Random	0.248	0.250	0.249	0.885	-0.001	4
LDA	0.870	0.722	0.789	0.806	0.733	5
Twitter-LDA	0.877	0.767	0.819	0.772	0.767	5
k-means	0.663	0.887	0.759	0.729	0.677	3
$STS_{0.7}$	0.945	0.840	0.889	0.711	**0.858**	6
$STS_{0.3,res}$	0.972	0.633	0.767	0.787	0.736	9
$STS_{0.8,DBpedia}$	0.926	0.824	0.872	0.781	0.836	7
$STS_{0.6,YAGO}$	0.959	0.807	0.876	0.611	0.846	7

175 Twitter users, 11-topic testset

	Precision	*Recall*	*F-score*	*NMI*	*MCC*	*Topics*
Random	0.110	0.092	0.100	0.769	0.001	11
LDA	0.799	0.552	0.653	0.777	0.632	15
Twitter-LDA	0.736	0.822	0.777	0.732	0.749	9
k-means	0.819	0.810	0.814	0.864	*0.791*	14
STS_0	0.700	0.947	0.805	0.686	*0.788*	9
$STS_{0.35,res}$	0.791	0.794	0.793	0.768	0.767	6
$STS_{-0.1,DBpedia}$	0.484	0.938	0.639	0.677	0.621	8
$STS_{0.35,YAGO}$	0.655	0.927	0.768	0.735	0.748	9

Table 7.4: Top topic labels discovered for the best results for the 4-topic testset (top terms per topic for Twitter-LDA with 5 topics, top traits per cluster for STS_γ).

4-topic set	Twitter-LDA terms	$STS_{0.7}$ classes
Computer Science	1: swift, ios, app 2: data, bigdata, big	1: Big_data, EmergingTechnologies, Web_analytics 2: Model, ProgrammingLanguagesCreatedIn1995, IOS
Sports	golf, game, win	1: Cricketer, NationalCricketTeams, SoccerPlayer 2: GolfPlayer, PGATourGolfers, Golfer
Cars	car, f1, ferrari	CarManufacturer, SportsCarManufacturers, Coupes
Politics	auspol, people, obama	Statesman, NationalLeaders, PoliticiansFromSydney

Table 7.5: Top topic labels discovered for the best results for the 11-topic testset (top terms per topic for Twitter-LDA with 9 topics, top traits per cluster for STS_γ).

11-topic set	Twitter-LDA terms	STS_0 classes
iOS development	swift, ios, app	-
Web development	-	Model, RichInternetApplicationFrameworks, IOS
Data Science	data, bigdata, big	Big_data, EmergingTechnologies, Data
Soccer	-	FootballClubsInEngland, FIFAWorldCupPlayers
Football	good, game, day	AmericanFootballLeagueTeams, GridironFootballPlayer, AmericanFootballPlayer
Golf	golf, cup, rydercup	GolfPlayer, PGATourGolfers, AmericanGolfers
Cricket	ausvind, cricket, india	-
Cars	car, f1, ferrari	CarManufacturer, Coupes, SportsCarManufacturers
US politics	obama, president, people	DemocraticPartyUSSenators, AmericanLegalScholars, HarvardLawSchoolAlumni
UK politics	labour, people, nhs	NationalistPartiesInTheUK, UKIndependenceParty, ConservativePartiesInTheUK
Australian politics	auspol, abbott, australia	PoliticiansFromSydney, AustralianPoliticians, AustralianRhodesScholars

is a scope of $\gamma = 0.7$ (6 topics found), and for the 11-topic set a scope of $\gamma = 0.0$ (9 topics found). For LDA, Twitter-LDA and *k*-means in figures 7.14, 7.15 and 7.16, the progression looks similar (although reversed). For LDA we get the best results for 5 topics and 15 topics respectively; for Twitter-LDA, for 5 and 9 topics; and for *k*-means, for 3 and 14 topics.

Regarding the different versions of STS_γ used, we see a somewhat expected result: using only DBpedia classes (of which there are few and they are generic) we do well at discovering the generic topics but does poor at specific topics; using only resources we see the opposite result. Using only YAGO classes and using all classes combined do well at both parts, with STS_γ outperforming $STS_{\gamma,YAGO}$ by a small margin.

Overall, our best-performing method, STS_γ, outperforms standard LDA by 17% for 4 topics and by 26% for 11 topics, and Twitter-LDA by 11.9% for 4 topics and by 5.2% for 11 topics. STS_γ outperforms *k*-means by 26.7% for the 4-topic testset, but performs roughly equal for the 11-topic testset. This conspicuous result can be explained by the fact that *k*-means does not take any topic hierarchy into account; we discuss this further in the discussion section (5.6). Lastly, the random baseline

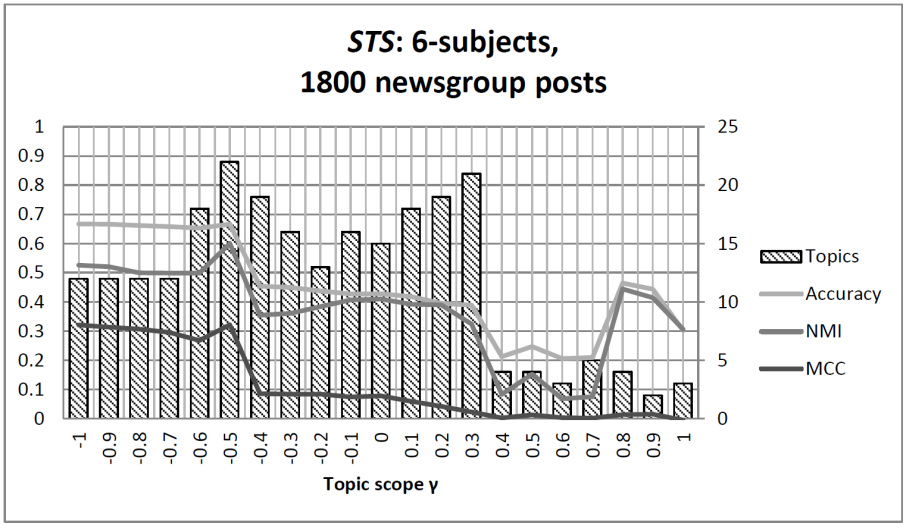

Figure 7.17: Results for STS_γ-clustering on the 6-subject testset of the 1800 newsgroup articles.

performs as expected, with the MCC score hovering around 0.

We now look at the discovered topics and their labels for the best results, taking the top terms for Twitter-LDA and using our topic labeling technique described in section 4.3 to find the top classes for STS_γ. From table 7.4, we see that for the 4-topic set, the best Twitter-LDA result (5 topics) has wrongly split *Computer Science* into two sub-topics. For $STS_{0.7}$, *Computer Science* and *Sports* are erroneously split up into sub-topics. Upon inspection of the ontologies for the *Sports*-related topics and entities, we find that many entities simply do not link to a "Sports" class: in fact, there is no such class in the YAGO hierarchy. For *Soccer*, for example, common entities found in tweets are team names. Entities such as "Arsenal F.C." expand to "PremierLeagueClub" $\rightarrow$ "Club" $\rightarrow$ "Association" $\rightarrow$ ... , never reaching a class that could identify it with *Sports*. This is why we only see "CricketTeams", "SoccerPlayer", "Golfer", etc. in our result.

Looking at table 7.5 for the 11-topic testset, we see that for the best Twitter-LDA result (9 topics), the *Web development* and *Soccer* topics have not been properly identified. For STS_0, we fail to identify *Cricket*, and *iOS development* gets erroneously (although the two topics are highly related) combined with the *Web development* cluster, as is evident from the "iOS" label. Of note is that STS_γ, unlike LDA, correctly distinguishes between *Soccer* and *Football*.

Not listed are the top terms found for topics using standard LDA. These were generally similar to the terms Twitter-LDA found, but contained a number of extra "noise" topics, containing words such as "great", "today", "make". This noise was successfully filtered out by Twitter-LDA, explaining its better performance over standard LDA.

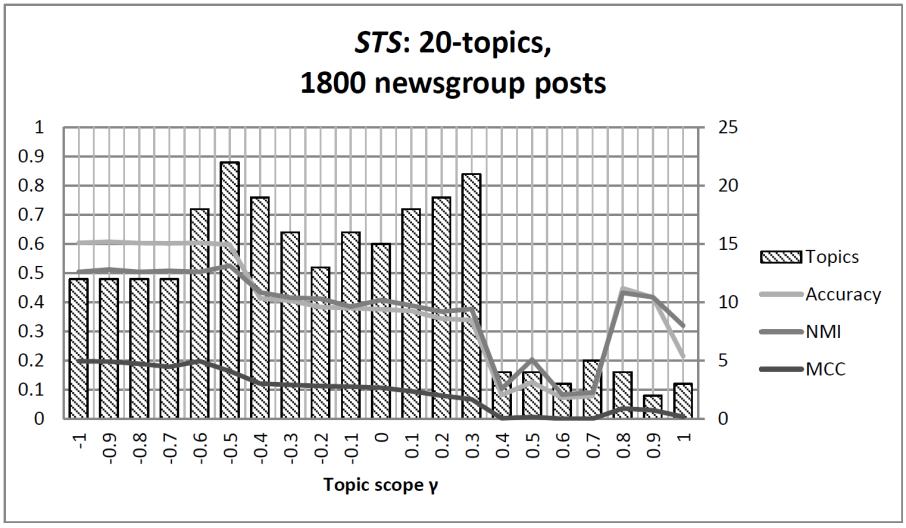

Figure 7.18: Results for STS_γ-clustering on the 20-topic testset of the 1800 newsgroup articles.

Newsgroups dataset Next, we apply the same methods with the same settings to two configurations of the newsgroups dataset. We consider clustering in either the 6 subject matters or in the full 20 newsgroups (see table 7.1). First, we try the set of 1800 newsgroup posts consisting of 90 posts evenly and randomly taken from each newsgroup. The STS_γ, LDA, Twitter-LDA and k-means results are plotted in figures 7.17, 7.18, 7.19, 7.20 and 7.21. The best results, including a comparison to the random baseline, are summarized in table 7.6. We omit the clearly inferior versions of STS_γ for this evaluation, comparing only the variant using all types of ontology classes.

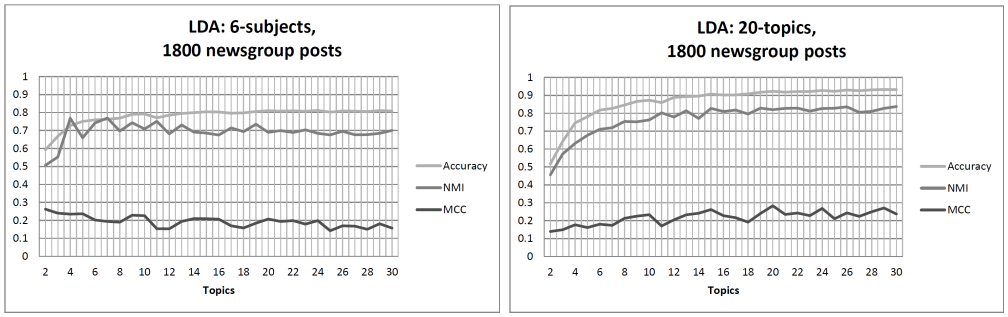

Figure 7.19: Results for LDA-based clustering on the 1800 newsgroup articles.

As expected, we see that results in terms of absolute F-score and MCC are significantly poorer compared to the Twitter dataset when we aim to detect all 20 topics, some of which we lack enough information about due to the brevity of posts. For the 6-subject test set, however, STS_γ still manages to outperform (Twitter-)LDA, but

oddly enough it does so for lower topic scopes. STS_{-1} found 12 topics, while LDA and Twitter-LDA gave wildly different results at 2 and 9 topics respectively. *k*-means peaked at 13 topics, but with a significantly lower score. A likely reason for the poor performance of LDA is that the cluster sizes for the 6-subject testset are very uneven, ranging from 90 to 450 documents (see table 7.1): this is a known weakness of LDA, which is biased towards even-sized clusters. This is especially visible in the NMI scores for the 20-topic testset: both LDA variants score highly here due to all 20 clusters having roughly the right sizes (interestingly, the random clustering has a significantly higher NMI than even LDA: this makes sense, since we distribute nodes uniformly at random over 20 topics, and take an average of 100 iterations, leading to a very even distribution).

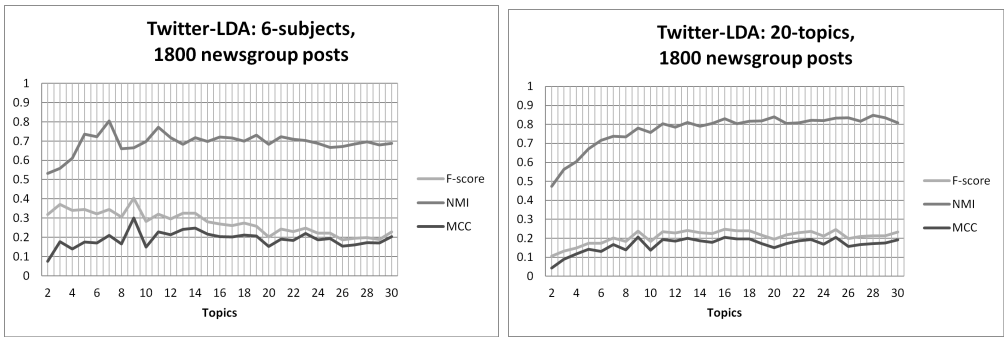

Figure 7.20: Results for Twitter-LDA-based clustering on the 1800 newsgroup articles.

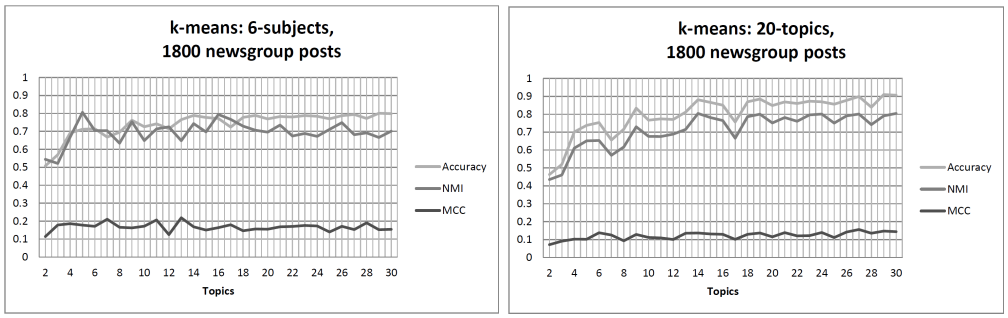

Figure 7.21: Results for k-means-based clustering on the 1800 newsgroup articles.

For 20-topics, standard LDA performs best, with a very significant improvement over the Twitter-specific version. STS_{γ} performs very poorly here. We interpret these results in more detail later.

Table 7.6: Summary of optimal results for the 1800 newsgroup article dataset.

1800 Newsgroup posts, 6-subject testset

	Precision	*Recall*	*F-score*	*NMI*	*MCC*	*Topics*
Random	0.190	0.167	0.177	0.684	0.000	6
LDA	0.289	0.786	0.423	0.506	0.262	2
Twitter-LDA	0.488	0.345	0.404	0.664	0.300	9
k-means	0.373	0.352	0.362	0.648	0.219	13
STS_{-1}	0.334	0.760	0.464	0.526	**0.321**	12

1800 Newsgroup posts, 20-topic testset

	Precision	*Recall*	*F-score*	*NMI*	*MCC*	*Topics*
Random	0.050	0.051	0.050	0.873	0.001	20
LDA	0.282	0.370	0.320	0.820	**0.283**	20
Twitter-LDA	0.239	0.254	0.246	0.833	0.206	25
k-means	0.167	0.259	0.203	0.801	0.156	27
$STS_{-0.6}$	0.099	0.862	0.177	0.504	0.198	18

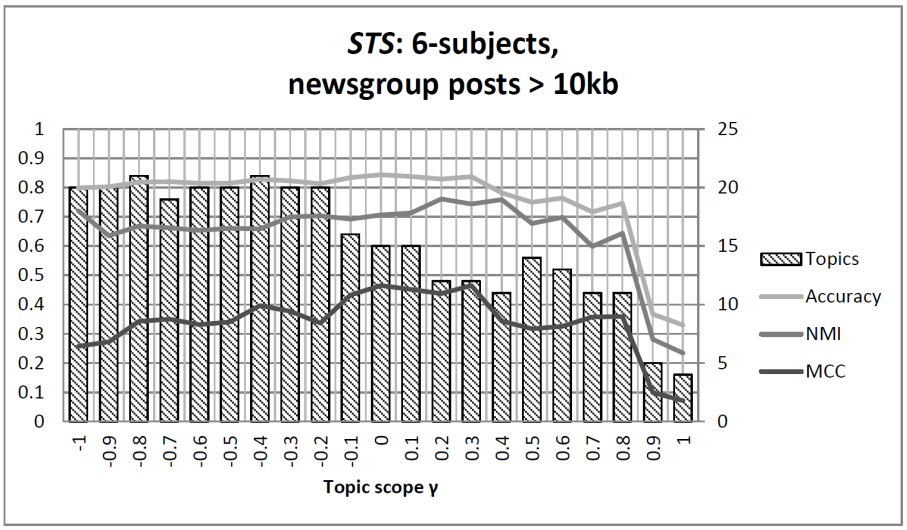

Figure 7.22: Results for STS_γ-clustering on the 6-subject testset of the newsgroup articles larger than 10.0 kb.

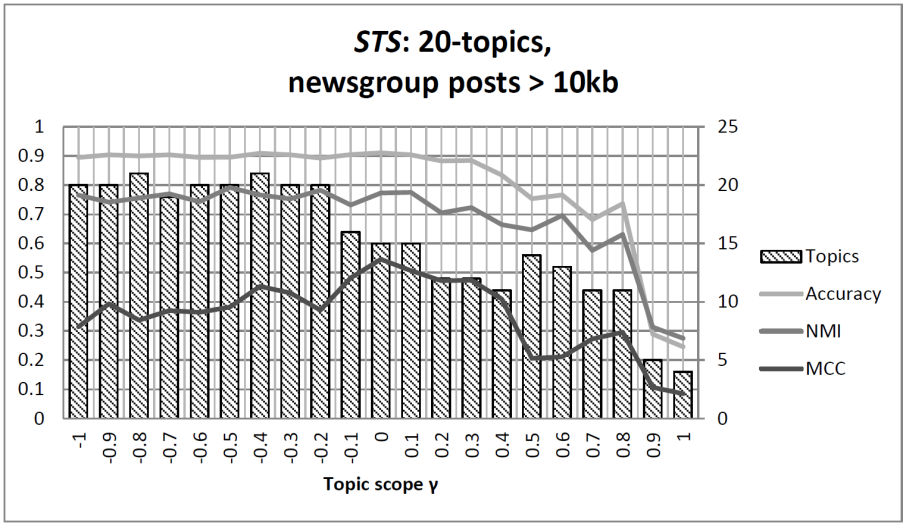

Figure 7.23: Results for STS_γ-clustering on the 20-topic testset of the newsgroup articles larger than 10.0 kb.

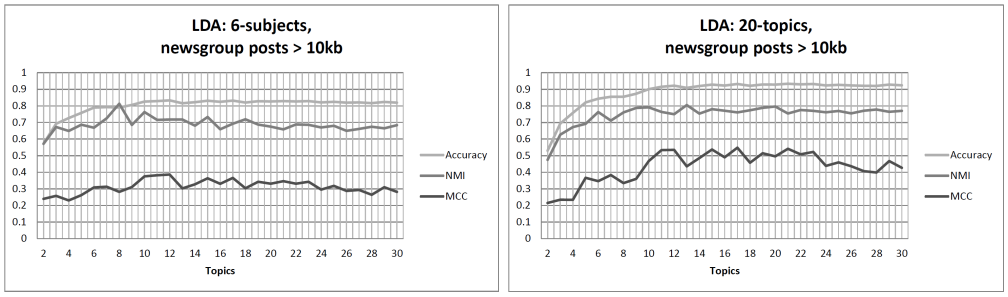

Figure 7.24: Results for LDA-based clustering on the newsgroup articles larger than 10.0 kb.

Next, we try the set of 239 newsgroup posts larger than 10.0 kilobytes in size. The STS_γ, LDA and k-means results are plotted in figures 7.22, 7.23, 7.24 and 7.25. The best results, including a comparison to the random baseline, are summarized in table 7.7.

This time, results in terms of the absolute MCC score are higher, as expected; both for LDA, but especially for STS_γ and k-means. The STS_γ-based approach outperforms LDA and k-means for the 6-subject testset again, with 12 topics detected at a topic scope of $\gamma = 0.3$. All methods performs roughly equal now for the 20-topic testset, which is a big improvement for STS_γ over the 1800 posts dataset. We note that we find the best result for the 20-topic testset with the topic scope parameter γ set to 0.0, yielding 15 topics; this is the same setting for which we had the best result for the Twitter 11-topic testset, which yielded 9 topics. $\gamma = 0.0$ yielded 15 topics for the 1800 newsgroup post set as well. This is an encouraging result that shows that, using STS_γ-clustering, we do not have to know how many topics there are, nor does the size or

Table 7.7: Summary of optimal results for the larger than 10.0 kb newsgroup article dataset.

Newsgroup posts larger than 10 kilobytes, 6-subject testset

	Precision	*Recall*	*F-score*	*NMI*	*MCC*	*Topics*
Random	0.213	0.164	0.186	0.756	0.000	6
LDA	0.683	0.319	0.434	0.718	0.387	12
Twitter-LDA	0.528	0.731	0.613	0.669	*0.497*	4
k-means	0.756	0.377	0.503	0.647	*0.457*	15
$STS_{0.3}$	0.669	0.467	0.550	0.744	*0.465*	12

Newsgroup posts larger than 10 kilobytes, 20-topic testset

	Precision	*Recall*	*F-score*	*NMI*	*MCC*	*Topics*
Random	0.092	0.048	0.063	0.778	0.000	20
LDA	0.683	0.497	0.576	0.761	0.548	17
Twitter-LDA	0.562	0.564	0.563	0.765	0.518	14
k-means	0.537	0.619	0.575	0.751	0.530	15
STS_0	0.512	0.688	0.587	0.773	0.546	15

content of the dataset matter; we can specify a desired topic scope, and automatically discover roughly the number of topics that exist at that scope.

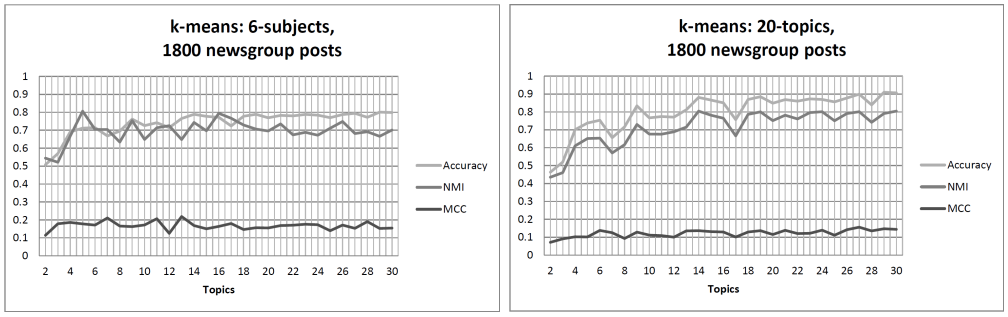

Figure 7.25: Results for k-means-based clustering on the newsgroup articles larger than 10.0 kb.

Hierarchical topic clustering Lastly, we will evaluate hierarchical clustering. We use the Twitter ground truth to try and cluster 175 Twitter users into a full hierarchy that contains both the 4 main topics and the 11 subtopics. For STS_γ, we use our recursive topic clustering algorithm (see algorithm 1), starting at the clustering that gave the best results for the 4-topic testset ($STS_{0.7}$). We compare against hierarchical LDA (hLDA)

Table 7.8: Summary of optimal results for hierarchical topic clustering.

175 Twitter users, 4 main and 11 sub-topics

	hP	*hR*	*hF*	*Levels*	*Topics per level*
hLDA	0.633	0.543	0.585	3	L1: 1, L2: 6, L3: 16
Hier. k-means	0.696	0.763	0.728	2	L1: 4, L2: 11
$STS_{0.7,recursive}$	0.877	0.837	0.857	3	L1: 6, L2: 8, L3: 2

and hierarchical k-means, which we apply in the manner described in section 7.2.2. We also initialize hierarchical k-means to the best 4-topic clustering. hLDA requires no initialization; instead we fix the depth to 3 levels. We evaluate the results using hierarchical precision, recall and F-score as explained in section 7.2.1. The results are summarized in table 7.8. Table 7.9 gives an overview of the actual topic hierarchies created for hLDA and STS_h (k-means does not support topic labeling), detailing the location of each cluster within each hierarchy using a number-based notation.

We see that STS_h gives the most accurate results, followed by hierarchical k-means. We found that even with 800 iterations, hLDA was unable to converge to the appropriate number of topics, leading to a poor hF score. hLDA could not detect a single topic of the 4 topics in the top layer accurately: either the clusters in level 2 consisted of more than 1 of the topics (e.g. cluster 1 of level 2 in table 7.9 comprised both *cars* and *computer science*), or of the topics we expected in the lower half of the hierarchy.

Statistical significance of results In [19], it is shown that given a sample of the full collection of users/documents (i.e. our ground truths), and p the true proportion of samples produced that is correct (which is unknown), n the size of the sample used to approximate p (the size of the ground truths: in our case 175 for Twitter users, and 1800 and 239 for each newsgroups set respectively) and $\hat{P}$ the approximation of p based on the ground truth, then this approximation lies in the interval

$$\hat{P} \in [p-\delta, p+\delta] \text{ where } \delta = \frac{1}{\sqrt{n}} \tag{7.11}$$

with 95% confidence. δ is thus the 95% confidence margin of error for the result. For example, if one result falls within this range of another result, then these two results do not differ sufficiently from each other in order for us to state with certainty that one is better than the other. We can apply this calculation on the resulting MCC scores for each approach to verify their statistical significance: we want to make sure that a good result according to the ground truth is sufficiently generalizable to a good result on larger, unseen data, and that an improvement of our method over the baselines is actually significant enough to draw conclusions about it.

For the Twitter users, we have $n = 175$, so the margin of error δ becomes $\frac{1}{\sqrt{175}} = 0.076$. Referring back the summary of results in table 7.3, we can conclude that for the 4-topic testset, our STS_γ-clustering gives a significant improvement over the baselines – the smallest difference, between STS_γ and Twitter-LDA, is $0.858 - 0.767 = 0.091 > 0.076$. For the 11-topic testset, our method again provides a significant improvement

Table 7.9: Clusters discovered at each level **L** for hierarchical clustering (top terms per topic for hLDA, top traits per cluster for STS_h). The numbers represent the level and position each cluster was located in the hierarchy.

L	hLDA topics	STS_h topic clusters
1	1: good, time, great	1: Big_data, EmergingTechnologies, Web_analytics 2: Model, ProgrammingLanguagesCreatedIn1995, IOS 3: Cricketer, NationalCricketTeams, SoccerPlayer 4: GolfPlayer, PGATourGolfers, Golfer 5: CarManufacturer, SportsCarManufacturers, Coupes 6: Statesman, NationalLeaders, PoliticiansFromSydney
2	1.1: car, cars, bmw 1.2: byu, game, coach 1.3: australia, today, minister 1.4: ballondor, live, fifa 1.5: data, learning, science 1.6: wt, pakvnz, pakistan	2.1: DistributedFileSystems, Apache_Hadoop, Run_batted_in 2.2: WebApplicationFrameworks, RichInternetApplicationFrameworks, Swift 3.1: NationalFootballLeagueTeams, GridironFootballPlayer, AmericanFootballPlayer 3.2: FootballClubsInEngland, PremierLeagueClubs, FootballClubsInLondon 3.3: CricketersAtThe2011CricketWorldCup, NationalCricketTeams, IndiaTestCricketers 6.1: RepublicanPartyStateGovernorsOfTheUnitedStates, UnitedStatesAirForceOfficers, Patient_Protection_and_Affordable_Care_Act 6.2: PoliticiansFromSydney, AustralianPoliticians, AustralianLeadersOfTheOpposition 6.3: NationalistPartiesInTheUnitedKingdom, UK_Independence_Party, ConservativePartiesInTheUnitedKingdom
3	1.1.1: swift, ios, app 1.1.2: vettel, webber, video 1.1.3: gsl, youtube, http 1.2.1: cowboys, bro, mfjs 1.3.1: auspol, abbott, manus 1.3.2: spotmyride, spotted, ferrari 1.4.1: golf, rydercup, year 1.4.2: league, goal, goals 1.5.1: obama, people, president 1.5.2: indyref, labour, scotland 1.5.3: car, ford, cars 1.5.4: data, bigdata, big 1.5.5: porsche, atlanta, photo 1.5.6: initializr, yelp, css 1.5.7: http, eurotour, sqcom 1.6.1: ausvind, india, cricket	2.2.1: CascadingStyleSheets, Sheet, WebDev 2.2.2: OS_X, Swift, Macintosh

over LDA, but the difference between STS_γ and Twitter-LDA lies within the margin of error ($0.039 < 0.067$): we cannot state that one method is better than the other with 95% confidence. Similarly, results for STS_γ and k-means can be considered equivalent, which is in line with our earlier observations, and subject of discussion in the next section.

For the main newsgroup dataset, $\delta = \frac{1}{\sqrt{1800}} = 0.024$. Due to the size of this testset, we obtain a much smaller margin of error compared to the set of Twitter users. Referring back to table 7.6, we see that all pair-wise results are statistically significant. Finally, for the set of newsgroup posts larger than 10kb, we obtain $\delta = \frac{1}{\sqrt{239}} = 0.065$. From table 7.7, we learn that for the 6-subject testset, the improvement of STS_γ over LDA is significant again, but we cannot make conclusions about the difference with Twitter-LDA and k-means clustering. For the 20-topic testset, we see that all results (excluding random) are essentially equivalent.

7.5 Results discussion

In this final section, we discuss the impact and implications of the results we obtained in greater detail. We first discuss the user recommendation method results, followed by the topic clustering method results.

7.5.1 Enhancing user recommandation

For most of the keywords analyzed and top-k values evaluated, our results show an improvement over the baseline approaches. Keyword *mars rover*, focusing on a very specific topic shows the most significant improvement. There are two main reasons for this. First, a specific topic is easier to model consistently, as the same specific set of classes tend to be associated to related entities (most commonly occurring classes are *NASAProbes* and *TerrestrialPlanets*, which would never appear in a more general context). Second, due to the popularity of the rover among the general public, many users not truly interested in Mars may mention the rover in passing when news concerning it is released, but are not consistently talking about the rover or Mars-related topics, making approaches that do not take tweeting consistency into account select the wrong users.

For "genetically modified" and "nuclear power", improvements are less significant but still visible, as these topics are more generic and thus more difficult to concretely define in terms of Wikipedia classes.

For "malware", performance is equal to the best baseline for $k = 5$, but worse for $k = 10$ and $k = 20$, where tweet count and TURKEYS outperform it. This can be explained by the fact that a large portion of the top users for "malware" seem to be either news aggregation bots that automatically post malware related topics (such as discoveries of new viruses, exploits or security holes), or users that automatically retweet these bots. These users typically have very high tweet counts. Since the taxonomical analysis approach is not perfect, it is incorrectly excluding some valuable users from the ranking.

Compared to the experiments performed in our previous work, on average the results obtained with the TURKEYS ranking are somewhat worse. This is because the

TURKEYS mechanism relies on the existence of a tightly knit user community and the mentions and retweets among them. This works particularly well when processing Japanese twitter users who are concentrated in a single location, use the same language and share the same culture, but it appears to be less effective on a global scale.

7.5.2 Scoped topical similarity and clustering

For Social Web content, our ontology-assisted topical similarity calculation and graph-based STS_γ-clustering results show a significant improvement over traditional tf-idf weighting, LSA-based topic modeling such as LDA and Twitter-LDA and common document clustering approaches such as k-means clustering. An important reason for the poor performance of tf-idf and LDA is the absence of overlapping terms due to the high dimensionality (and therefore high sparsity given the limited and noisy content on the Social Web) of the term vector space compared to our trait vector space. For k-means, despite using the same similarity calculation to construct graph G_γ as our STS_γ-clustering, γ was fixed to 0, which we found discovered appropriate topics at roughly the scope expressed by the 11-topic testset. However, even though we could force k-means to cluster the graph into 4 topics, it performed poorly in terms of F-score and MCC for the 4-topic testset. This is because k-means clustering does not take the existence of a latent topic hierarchy into account: the topology of the graph G_0 is shaped with a bias towards how many topics exist within the data at that particular topic scope. These results make clear the advantages of a hybrid approach – hierarchical topic modeling combined with graph-based community detection – compared to traditional methods.

For regular documents, we see that our ontology-assisted topic modeling approach yields results that are on par with LDA only when there is enough content available per document. It appears that ontologically expanding text content only works well when there is enough context information available – when the dataset contains many documents that sometimes consist of only one sentence, such as with the 20-newsgroups dataset, STS_γ-clustering gives poor results. This is a significant weakness of our approach. This is in line with results reported in [15]. Topic modeling approaches that iterate over the whole dataset when deriving a model, such as LDA, work better in these cases.

When there is enough content per document – as was the case for the dataset of newsgroup posts over 10.0 kb in size – ontology-assisted topical similarity as employed in STS_γ-clustering and k-means performs roughly equivalent to document-level word co-occurrence as employed in LDA. This is not a surprising result, since the newsgroup posts use more formal language and contain less noise than their Twitter counterparts, leading to more word overlap between documents.

Another reason for discrepancies is due to the (un-)evenness of clusters. Our clustering algorithm can deal with uneven clusters well, since clusters are determined solely based on the degree of similarity and number of similar users in the dataset. LDA is biased towards evenly sized topics, and it is very difficult for the method to detect topics of vastly different sizes, unless some form of iterated hyperparameter optimization is employed. In comparison to our approach, LDA gave the overall best result when dividing the 1800 newsgroup posts into 20 topics; this is the only dataset where all desired clusters are of even size (90 documents per cluster).

Table 7.10: qualitative comparison between the LDA and k-means baselines and STS_γ-clustering.

	LDA	Twitter-LDA	k-means	STS_γ-clustering
Perf. on Social content	-	+	+-	++
Perf. on documents	++	+	+-	+
Pre-defined topics	Yes	Yes	Yes	No
Hierarchical topics	Yes	Yes	No	Yes
Topic labels	Top terms	Top terms	None	Top classes (machine-readable)
Time complexity	$O(nkvi)$	$O(nkvi)$	$O(nkcij)$	$O(n^2c + n^3 \log n)$

A important observation is that classes from our trait vectors originally form a concrete hierarchy; this is not the case for terms, where this hierarchy needs to be guessed from the terms available in the corpus. This hierarchical information is retained within the trait vectors we calculate, making it easier for the original class ontology to be reconstructed in terms of traits. A nice side effect of this is that for traits, we may simply use the most characteristic ontology classes for a cluster as labels; and these labels are machine-readable and directly linked to DBpedia and the rest of the Linked Data cloud. Using LDA, the best we can do is to pick the top terms per topic that actually occurred within the text content, which are often less descriptive and harder for machines to reason about.

In conclusion, we summarize the results of the evaluation by means of a qualitative comparison between our approach and the baselines, in table 7.10. This overview includes computational time complexity per method, which we will touch upon again briefly. Due to STS_γ's reliance on a full connectivity graph based algorithm, its big-O complexity is in the $n^3 \log n$ order of magnitude, which is the highest of all the algorithms. This means that in its current form, the method does not scale well to large datasets. This is a significant limitation of the approach. A substitution of the current HCS-based algorithm with e.g. a density-based one such as ES clustering would improve scalability, but this is outside the scope of this thesis.

Chapter 8

Conclusions and Future Work

In this thesis, we have presented our work on using ontologies to enhance user recommendation, and the hierarchical clustering of Social Web users by their shared topics of interest. We have shown that we can bypass common limitations of the term vector space model by leveraging an external ontology to express user topic profiles in terms of trait vectors, and subsequently calculating scoped topical similarity, or STS_γ, between users; a measure that expresses the distinguishing characteristics of groups of users compared to the full collection of users at a certain level of topic generality.

We demonstrated the benefits of applying semantic analysis of timelines to user recommendation on Twitter. We first created an initial ranking of potentially valuable users by analyzing user relations, and ranking users according to a score based on a combination of tweet count and user influence (TURKEYS). We then took the top users from this ranking and analyzed their timelines to create topic taxonomies by matching terms in their posts to a background knowledge base (Wikipedia). In case topics were not steady across the user's timeline, we judged this user not to post consistently about these topics and excluded them from the ranking.

We applied community detection techniques on a graph constructed from the STS_γ between users and showed that, by incorporating the concept of topic scope into our calculations, we can get results based on the desired scope rather than the desired number of topics – with the same topic scope values, we managed to discover roughly the appropriate numbers of topics even for different data sources of different sizes. Furthermore, the approach could be used to generate human- and machine-readable labels for clusters, and to divisively cluster a group of users or documents in order to generate a full topic hierarchy[1].

For user recommendation, our evaluation showed that, depending on the keyword, applying an extra binary selection to the ranking of users allows us to obtain significantly more accurate user recommendations. We found that our method is particularly useful for specific topics that are easily modeled in terms of YAGO classes. For more generic keywords, improvements are less significant, but still visible.

For topic clustering, the experimental results presented showed an improvement of up to 14.7% over standard latent Dirichlet allocation, 11.9% over Twitter-LDA and up to 26.7% over k-means clustering on Social Web data. We also showed that we

[1]The source code used to obtain the results described in this paper, including an interactive visualizer for hierarchical clustering, can be found at `https://github.com/ktslabbie/TwinterestExplorer`

can correctly detect topics at different scopes by changing the topic scope parameter, whereas changing the number of topics for *k*-means clustering failed to detect the correct topics. Results on traditional documents were mixed, with results equivalent to or worse than the LDA state-of-the-art. For full hierarchical clustering, STS_h outperformed *hLDA* and a hierarchical version of *k*-means. Notably, we observe similar improvements over LDA using community detection-based clustering as was shown in Lancichinetti et. al. [33]. This is an encouraging result, suggesting that perhaps graph-based community detection methods for document clustering are a better fit for the task than probabilistic LSA-based methods, at least when the topics are not known in advance.

8.1 Future work

Throughout this work we have assumed that users can be cleanly divided into disjoint topic clusters, but in the real world this is obviously not the case: users can have multiple identifiable interests, or users may not post about any specific interest at all. We need to further develop and evaluate methods for overlapping community detection. One possible way is to replace the current HCS community detection algorithm with a clique percolation method [16] to discover overlapping cliques in a graph.

A significant untapped area of potential lies in multi-lingual topic clustering. Since the semantic classes we collect from DBpedia are language-agnostic, and DBpedia resources exist in multiple languages, it becomes trivial to link together equivalent entities from different languages and collect the same classes for both languages. Similarly, the DBpedia Spotlight entity recognizer is easily extensible with other languages and class models (although non-alphabetic languages such as Japanese will require additional processing). This will allow us to detect similar content in different languages without requiring any knowledge about these languages.

The current graph connectivity-based clustering algorithm used does not scale to large datasets. Future work would experiment with more modern density-based clustering algorithms as well

Lastly, we relied largely on YAGO, DBpedia and Schema.org classes to model user topics of interest. In reality, these class hierarchies are still quite limited in what they can express, as many entities simply do not have classes in DBpedia, or they are not sufficiently connected with superclass relations to the classes we are really looking for (such as most sports-related entities not actually being connected to a “Sports” class). Other than these types of super- and subsumption relations, there are a great many other relations available in DBpedia (e.g. “team of” linking players to teams, “field of” linking academic disciplines to famous researchers, etc.). How to make good use of these relations is another promising area to explore.

Bibliography

[1] Dbpedia wiki: The dbpedia ontology (2014). *http://wiki.dbpedia.org/Ontology2014*, Retrieved on April 14 2015.

[2] Fabian Abel, Qi Gao, Geert-Jan Houben, and Ke Tao. Analyzing user modeling on twitter for personalized news recommendations. In *User Modeling, Adaption and Personalization*, pages 1–12. Springer, 2011.

[3] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, chapter 52, pages 722–735. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.

[5] Tim Berners-Lee and Mark Fischetti. *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, September 1999.

[6] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[7] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, Mar 2009.

[8] David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

[9] David M Blei, Thomas L Griffiths, and Michael I Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):7, 2010.

[10] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[11] Chris K Carter and Robert Kohn. On gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994.

[12] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 121–124. ACM, 2013.

[13] S Dasgupta, CH Papadimitriou, and UV Vazirani. Algorithms–chapter 5, 2006.

[14] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[15] Leon Derczynski, Diana Maynard, Niraj Aswani, and Kalina Bontcheva. Microblog-genre noise and impact on semantic annotation accuracy. In *Proceedings of the 24th ACM Conference on Hypertext and Social Media*, pages 21–30. ACM, 2013.

[16] Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical review letters*, 94(16):160202, 2005.

[17] Mark S Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.

[18] The Buntin Group. Social usage involves more platforms, more often. *www.emarketer.com/Article/Social-Usage-Involves-More-Platforms-More-Often/1010019*, Retrieved on February 19 2013.

[19] Willem Van Hage, Antoine Isaac, and Zharko Aleksovski. Sample evaluation of ontology-matching systems. In *Fifth Int. Workshop on Evaluation of Ontologies and Ontology-based Tools, ISWC 2007*.

[20] John Hannon, Mike Bennett, and Barry Smyth. Recommending Twitter users to follow using content and collaborative filtering approaches. In *4th ACM Conference on Recommender Systems (RecSys '10)*, pages 199–206, 2010.

[21] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

[22] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information processing letters*, 76(4):175–181, 2000.

[23] M. Hausenblas and R. Cyganiak. Schema.rdfs.org. *http://schema.rdfs.org/*, Retrieved on April 20 2015.

[24] Tuan-Anh Hoang and Ee-Peng Lim. On joint modeling of topical communities and personal interest in microblogs. In *Social Informatics*, pages 1–16. Springer, 2014.

[25] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[26] Andreas Hotho, Steffen Staab, and Gerd Stumme. Ontologies improve text document clustering. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 541–544. IEEE, 2003.

[27] Anil K Jain, Richard C Dubes, et al. *Algorithms for clustering data*, volume 6. Prentice hall Englewood Cliffs, 1988.

[28] Nilesh Jain, Priyanka Mangal, and Deepak Mehta. Angularjs: A modern mvc framework in javascript. *Journal of Global Research in Computer Science*, 5(12):17–23, 2015.

[29] Kalervo Jarvelin and Jaana Kekalainen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[30] Svetlana Kiritchenko, Fazel Famili, S Matwin, and R Nock. Learning and evaluation in the presence of class hierarchies: Application to text categorization. 2006.

[31] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[32] Su Mon Kywe, Ee-Peng Lim, and Feida Zhu. A survey of recommender systems in twitter. In *Social Informatics*, pages 420–433. Springer, 2012.

[33] Andrea Lancichinetti, M Irmak Sirer, Jane X Wang, Daniel Acuna, Konrad Körding, and Luís A Nunes Amaral. High-reproducibility and high-accuracy method for automated topic classification. *Physical Review X*, 5(1):011007, 2015.

[34] Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th international conference on machine learning*, pages 331–339, 1995.

[35] Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, pages 631–640. ACM, 2010.

[36] Xiaohua Liu, Ming Zhou, Furu Wei, Zhongyang Fu, and Xiangyang Zhou. Joint inference of named entity recognition and normalization for tweets. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 526–535. Association for Computational Linguistics, 2012.

[37] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[38] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[39] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[40] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proc. of the 7th Intl. Conference on Semantic Systems*, 2011.

[41] Matthew Michelson and Sofus A Macskassy. Discovering users' topics of interest on twitter: a first look. In *Proceedings of the fourth workshop on Analytics for noisy unstructured text data*, pages 73–80. ACM, 2010.

[42] George A. Miller. WordNet: a lexical database for English. *Commun. ACM*, 38(11):39–41, 1995.

[43] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[44] Makoto Nakatsuji, Yasuhiro Fujiwara, Toshio Uchiyama, and Hiroyuki Toda. Collaborative filtering by analyzing dynamic user interests modeled by taxonomy. In *11th International Semantic Web Conference*, pages 361–377, 2012.

[45] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[46] Tomoya Noro, Fei Ru, Feng Xiao, and Takehiro Tokuda. Twitter user rank using keyword search. *Information Modelling and Knowledge Bases XXIV. Frontiers in Artificial Intelligence and Applications*, 251:31–48, 2013.

[47] Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 165–176. IEEE, 2006.

[48] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. Community detection in social media. *Data Mining and Knowledge Discovery*, 24(3):515–554, 2012.

[49] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 569–577. ACM, 2008.

[50] Oxford University Press. Rt this: Oup dictionary team monitors twitterer's tweets. http://blog.oup.com/2009/06/oxford-twitter/, 2009.

[51] Minghui Qiu, Feida Zhu, and Jing Jiang. It is not just what we say, but how we say them: Lda-based behavior-topic model. SIAM.

[52] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[53] Jason Rennie. The 20 newsgroups data set. *http://qwone.com/ jason/20Newsgroups/*, Retrieved on April 2 2015.

[54] Alan Ritter, Sam Clark, Oren Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534. Association for Computational Linguistics, 2011.

[55] Jason Ronallo. Html5 microdata and schema. org. *Code4Lib Journal*, 16, 2012.

[56] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494. AUAI Press, 2004.

[57] Naomi Sager. *Natural language information processing*. Addison-Wesley Publishing Company, Advanced Book Program, 1981.

[58] Salvatore Sanfilippo and Pieter Noordhuis. Redis, 2010.

[59] Carlos N Silla Jr and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[60] Kristian Slabbekoorn. Domain-aware ontology matching on the semantic web. Master's thesis, TU Delft, Delft University of Technology, 2012.

[61] Kristian Slabbekoorn, Laura Hollink, and Geert-Jan Houben. Domain-aware ontology matching. In *The Semantic Web–ISWC 2012*, pages 542–558. Springer, 2012.

[62] Kristian Slabbekoorn, Tomoya Noro, and Takehiro Tokuda. Towards twitter user recommendation based on user relations and taxonomical analysis. In *23nd European-Japanese Conference on Information Modelling and Knowledge Bases (EJC), 2013*, 2013.

[63] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[64] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of WWW'07*, pages 697–706, 2007.

[65] Zareen Saba Syed, Tim Finin, and Anupam Joshi. Wikipedia as an ontology for describing documents. In *2nd International Conference on Weblogs and Social Media*, pages 136–144, 2008.

[66] Lei Tang and Huan Liu. Community detection and mining in social media. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2(1):1–137, 2010.

[67] Oren Tsur, Adi Littman, and Ari Rappoport. Efficient clustering of short messages into general domains. In *Proceedings of the 7th International Conference on Weblogs and Social Media*, 2013.

[68] Steven J Vaughan-Nichols. Will html 5 restandardize the web? *Computer*, (4):13–15, 2010.

[69] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. TwitterRank: Finding topic-sensitive influential Twitterers. In *3rd ACM International Conference on Web Search and Data Mining*, pages 261–270, 2010.

[70] Peter Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[71] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrix–vector multiplication on emerging multicore platforms. *Parallel Computing*, 35(3):178–194, 2009.

[72] Shuang-Hong Yang, Alek Kolcz, Andy Schlaikjer, and Pankaj Gupta. Large-scale high-precision topic modeling on twitter. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1907–1916. ACM, 2014.

[73] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms (TALG)*, 1(1):2–13, 2005.

[74] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In *Advances in Information Retrieval*, pages 338–349. Springer, 2011.

[75] Zhongying Zhao, Shengzhong Feng, Qiang Wang, Joshua Zhexue Huang, Graham J Williams, and Jianping Fan. Topic oriented community detection through social objects and link analysis in social networks. *Knowledge-Based Systems*, 26:164–173, 2012.