Universiteit Leiden
Opleiding Informatica

Content-based tag recommendation algorithms

for unstructured data

Name: Bas P. Harenslak

Date: 24/11/2014

1st supervisor: Dr. Michael S. Lew
2nd supervisor: Dr. Erwin M. Bakker

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Content-based tag recommendation algorithms for unstructured data

Bas P. Harenslak
b.p.harenslak@umail.leidenuniv.nl

November 24, 2014

Abstract

Organisations process a huge amount of documents on a daily basis. Such documents are often stored in an unstructured manner, e.g. HTML, Word or PDF files. The term "unstructured data" refers to information that isn't stored in a traditional relational database. Such data can contain valuable information, but it is hard to find and organise since the data does not follow a fixed structure. Tagging resources, annotating them with keywords, is a great way to improve findability of resources in a large collection of resources.

Tagging is however a human task. This research focuses on the task of tag recommendation, which involves finding and suggesting relevant keywords for documents from their content and relations between users, documents and tag-words. We discuss various types of algorithms and test their performance on three real world datasets. Besides some well known algorithms, we implemented a system proposed by Lipczak et al. which won a major scientific event focussed on tag recommendation. His system outperforms the well known algorithms and returns a maximum F1 score of 0.3127 on the DeliciousT140 dataset. We propose two extensions to his system which respectively achieve a maximum F1 score of 0.3167 and 0.3216.

Contents

Chapter 1: Introduction

While the amount of digital information is growing at an enormous speed and the use of *Big Data* technologies allow us to discover information hidden within the data, most of the stored data is not utilized for data mining and analytics tasks. Estimations by IDC, Forrester and Gartner suggest that over 80% of all enterprise data consists of *unstructured data*. Unstructured data, as opposed to *structured data*, does not follow a fixed, pre-defined structure. It is typically human-generated and is not suitable for storage in a traditional database. The content is saved in formats such as Word, PDF or HTML. These are examples of mostly textual documents, but unstructured data also includes o.a. audio and video recordings, PowerPoint-presentations and e-mails.

The massive amount of unstructured data is a great potential source of information, however it asks for different technologies as used on machine-generated data (e.g. sensor information). The field of *text mining* is involved with deriving relevant information from text, e.g., sentiment analysis in which an opinion is extracted from a document. This research covers topics in *information retrieval* and *natural language processing*, both tasks in text mining. We investigate a *tag recommendation* system, a system which reads textual documents and recommends tags based on the document content. A resource can be annotated with words identifying the resource. The words, or tags, are relevant to the resource and can be helpful when searching for a resource in a large collection of resources.

A *folksonomy* is a 'collaborative tagging system'. In a folksonomy, users annotate resources with tags. The tags are not restricted to a limited set of words and thus a folksonomy vocabulary arises from the user's personal vocabularies. Folksonomies became popular with the introduction of web services such as Flickr and Delicious. Such services are very valuable for research on tag recommendation systems since recommendation algorithms can be validated given the user tags connected to resources.

In this research we investigate several tag recommendation algorithms, which are validated using folksonomy datasets made available for research purposes. We develop a system which recommends tags given resources. The goal of annotating resources with tags is to make the resources easier findable in a large collection of resources. In this chapter, we discuss the motivation, challenges and objectives behind this research. We formalize the problem and define three research questions which we intend to answer during this research.

1.1. Tag recommendation

A *tag* is a piece of *metadata* describing a resource. Metadata is additional information about a resource, describing e.g. the time of creation, author, location, size of a photo or content of the resource.

In Figure 1.1 we see a *tag cloud*[1]. A tag cloud is a visualization of words in which the words are formatted such that the most important, frequent or relevant words are displayed. Given a large collection of tagged documents, one can easily see the most important keywords in the collection in the cloud of words.

[1] Generated from this document using Wordle (`http://www.wordle.net`).

Figure 1.1: Tag cloud example

Tag recommendation is the process of finding and recommending the most relevant words for a given document. The process of finding relevant words is used in various other applications:

- Indexing [21]
- Document annotation
- Metadata generation [30]
- Feature selection [6]
- Labelling [28]
- Keyword extraction [27]

1.2. Motivation

The continuously growing amount of information makes it impossible to find the right piece of information without the help of technology. Companies process millions of resources on a daily basis. Annotating resources with tags, which characterize the resource, are extremely helpful in finding and organising resources. Adding tags however is a human task. Processing natural language is a very hard task for computers and remains an active field of research.

This research is a collaboration between Leiden University and Capgemini Netherlands. One of Capgemini's content management products is Nuxeo. Nuxeo is an open source Enterprise Content Management (ECM) platform, which allows document management, case management, digital asset management and other applications[2]. The Nuxeo platform's first release was in February 2007. Up to now, users can manually add tags to resources. There is no way of automatically generating tags from a resource.

Manually adding tags is a time-consuming process and not all users take the time to annotate resources. Annotated resources are easier to find and thus it would be very useful if all resources could be tagged automatically. This research is intended to investigate a 'tag recommendation system', i.e. a system that is able to recommend relevant tags for a given resource.

[2] `http://www.nuxeo.com`

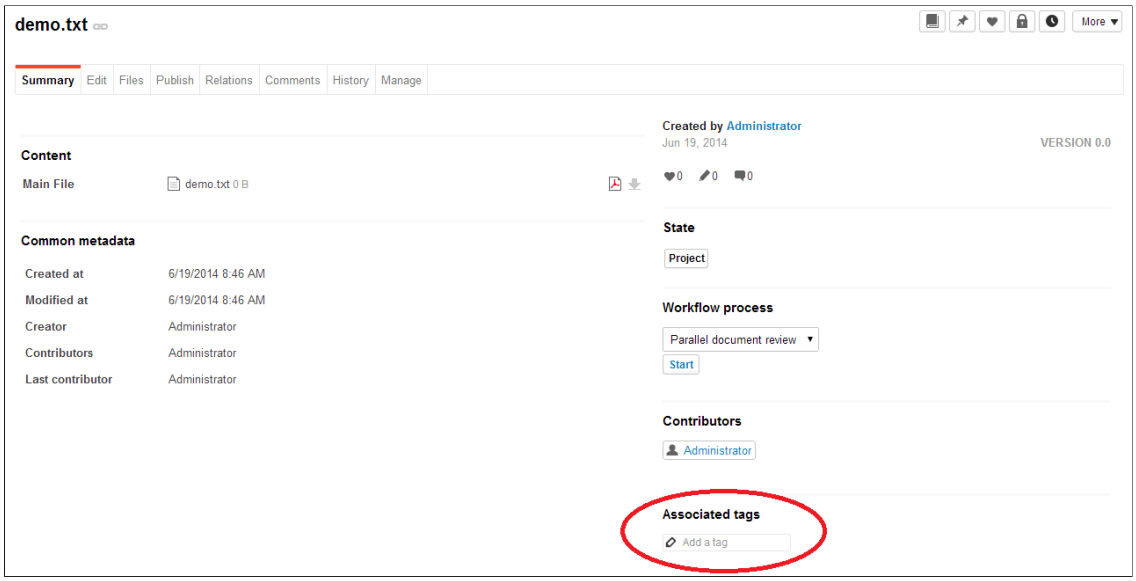


Figure 1.2: Uploading a resource and manually adding tags in Nuxeo Platform.

1.3. Challenges

The recommendation task faces several challenges. The *cold start* problem is a general issue in recommender systems since no information is known about new items, whether it's a new user or a new resource. *Personalisation* remains a challenge, since it is always debatable whether or not tags must include a user's tagging history. The '*curse of dimensionality*' is a large issue for most recommendation systems. A document can contain many unique words. For example, take a collection of 20,000 tagged documents by 5,000 unique tag-words and 100 users. Storing such information in a $d \times t \times u$-matrix results in $20,000 \times 5,000 \times 100 = 10$ billion entries, of which most are empty and thus the resulting matrix is extremely large yet extremely sparse. We require smart techniques for dealing with large amounts of information.

Traditional recommender systems are applied to problems such as product recommendation based on user ratings. Such problems are two-dimensional: the data can be stored in a 2-D matrix in which rows represent users and columns represent items. Each cell is a numerical rating given by the user to the item. Tag recommender systems are more complex since there is an extra dimension in the form of tags. A user can tag a resource using one or more words. Numerical ratings such as '1' and '5' can be easily compared, but it is very hard for a computer to compare e.g. 'train' and 'car'. The fact that words can have multiple meanings adds complexity to the challenge.

1.3.1. Natural Language Processing

The field of *Natural Language Processing* (NLP) focusses on computer systems for understanding and interacting using human language. The language can be either written or spoken. Some research topics in NLP are optical character recognition (OCR, reading handwritten text), automatic summarization, automatic translation, entity recognition, sentiment analysis and speech processing.

1.3.2. Information Retrieval

Information Retrieval (IR) is a research field closely related to NLP in which the goal is to extract relevant pieces of information from a large number of resources. Processing text is a primary

task in IR, in which NLP techniques are useful. The typical IR task is a search task that returns a ranked list of items that are relevant to a search query. In Figure 1.3 we see a search for "computer science books" on Amazon. 148033 results are returned, ordered from most to least relevant. The screenshot shows the top 5 results. None of the titles of these items contain all three words "computer", "science" and "books", yet Amazon's system returns them as relevant to the search query. In tag recommendation, the search query is a resource, and the relevant items are tag-words.

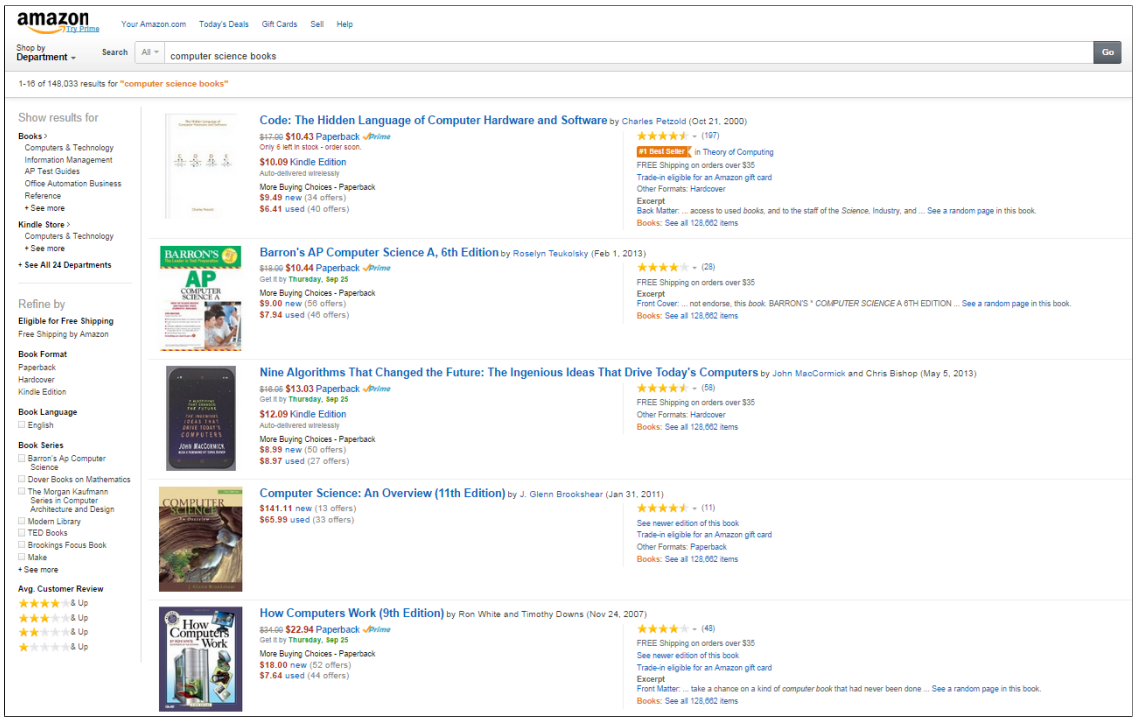


Figure 1.3: Search query for "computer science books" on Amazon

1.3.3. Semantics

Semantics is the study of meaning, a challenge for both NLP and IR. Natural language contains relationships between words, making it hard to interpret by computers. Some concepts in semantics are:

1. **Synonymy** Different words having one meaning, e.g., "buy" and "purchase".
2. **Polysemy** One word having multiple meanings, e.g. "bank" meaning a financial institution or a river bank.
3. **Antonymy** Words having opposite meanings, e.g., "plus" and "minus".
4. **Hyponymy** A relation between an instance and its general term (its hypernym). For example: "red", "green" and "blue" are all hyponyms of "colour". Hierarchical relationships between words are often displayed using a tree structure, called a *taxonomy*.
5. **Word order** For example:
   - Bob walks to the park after dinner.
   - **Only** Bob walks to the park after dinner. (nobody else walks to the park)
   - Bob **only** walks to the park after dinner. (he does nothing else after dinner)

- Bob walks **only** to the park after dinner. (he goes nowhere else after dinner)
- Bob walks to the park **only** after dinner. (he doesn't walk at other times)

1.3.4. Social and enterprise tagging strategies

Most research in the field of tagging strategies focusses on *social tagging*. These systems are characterized by the fact that individual users tag or bookmark items, and share the items publicly. *Enterprise tagging* knows a number of limitations. Social tagging systems don't contain restrictions, i.e., there is no structure or regulation of vocabulary, which is desirable in a corporate environment. Corporate content is used for specific tasks, and everybody in a company benefits from improved document findability. Also, the number of users in an enterprise environment is generally much lower. Millions of people use services such as Facebook or Instagram, while a company with only a thousand employees would already be considered a large company. Privacy and security issues are also important since some information might be kept secret, even to employees within the same company. Lastly, the quality of tags is more important in enterprise tagging. People tend to tag inconsistently and tag items for the best personal findability, while those tags might not be of use to other users. Adding clear and consistent tags will benefit everybody in a company.

While we do not focus specifically on social or enterprise tagging in this research, it is clear that the need for sophisticated tag recommendation techniques is even more present in enterprise tagging.

Social information systems are designed in different ways. *Ontology* is the study concerned with the existence of categories of entities that (may) exist is a domain. An ontology is usually depicted in a mindmap. A *taxonomy* is a hierarchical classification system in which the hierarchy, or categories and sub-categories, is controlled by experts. The word taxonomy derives from the Greek words *taxis* (arrangement or division) and *nomos* (law). *Folksonomies* are also hierarchical classification systems, but differ from taxonomies because the tags or categories are not assigned by experts. All content in folksonomies is collaboratively controlled by the community, i.e., users add their own tags, based on their own vocabulary. Folksonomies are applied on many social media platforms, e.g. Flickr[3], Wordpress[4] and Tumblr[5].

1.4. Objectives

The objective of this work is to investigate and improve existing work in the field of tag recommendation and gain knowledge and experience in the functioning of such algorithms. We want to discover how content from documents can be extracted and handled in order to retrieve relevant tag-words from documents, i.e. we would like to transform unstructured data into some form of structured data in order to retrieve relevant information.

Given the fact that this research is a collaboration between Leiden University and Capgemini Netherlands, we focus on 'as real as possible' challenges, i.e. we include o.a. processing speed of algorithms in the testing of algorithms and are interested in finding an optimal number of tags to recommend.

1.5. Formal problem definition

The tag recommendation problem consists of three components: users $u_i \in U$, resources $r_j \in R$ and tags $t_k \in T$.

[3] `https://www.flickr.com`
[4] `http://wordpress.org`
[5] `https://www.tumblr.com`

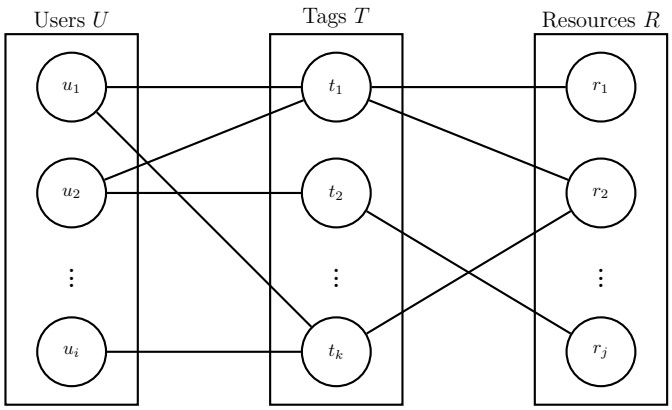


Figure 1.4: Tripartite structure of a tagging system

The users, tags and resources are connected by posts. A post is a triplet $p_{ij} \in (u_i, r_j, T_{ij})$ where $T_{ij} = \{t_k\}$ is a set of tags between a user and a resource. The entire folksonomy is a set of posts. We refer to a triplet as URT.

We formally define a folksonomy as $\mathbb{F} := (U, T, R, Y)$. U, T and R are finite sets whose elements are respectively users $u \in U$, tags $t \in T$ and resources $r \in R$. Y defines a ternary relation between U, T and R: $Y \subseteq U \times T \times R$, which are tag assignments.

From the folksonomy $\mathbb{F}$ we can derive various subsets:

Notation	Description	Definition
Y_u	All tag assignments by user u	$Y_u := Y \cap (\{u\} \times T \times R)$
Y_t	All tag assignments for tag t	$Y_t := Y \cap (U \times \{t\} \times R)$
Y_r	All tag assignments on resource r	$Y_r := Y \cap (U \times T \times \{r\})$
$Y_{u,t}$	All tag assignments by user u using tag t	$Y_{u,t} := Y \cap (\{u\} \times \{t\} \times R)$
T_u	All tags assigned by user u	$T_u := \{t \in T \vert \exists r \in R : (u,t,r) \in Y\}$
$T_{u,r}$	All tags assigned by user u to resource r	$T_{u,r} := \{t \in T \vert (u,t,r) \in Y\}$
P	A set of all posts. A post consists of a user, a resource and all tags that the user assigned to the resource	$P := \{(u,S,r) \vert u \in U, r \in R, S = T_{u,r}, S \neq \emptyset\}$

Table 1.1: Folksonomy subsets

Users are usually represented by a user ID, tags can be any combination of characters, i.e. a string, and the identification of resources depends on the system. Most document management systems create a unique ID and store the resources on a hard drive and save the location of the file in a database.

1.6. Research questions

We formulate three research questions:

RQ1. How can the content of resources be used to recommend accurate tags?

We focus on previously untagged resources, i.e. the resources uploaded by users have not been uploaded before and thus no tags are added to it yet. We want to recommend tags based purely on a resource's content, without any other knowledge from e.g. the folksonomy.

RQ2. How can algorithms be combined in order to produce stronger tag recommendations?

Combining multiple algorithms to produce a single, improved recommendation is a much applied technique in other types of recommendation. In the tag recommendation challenge, there have been only a handful attempts. The most well known one (to the best of our knowledge) was by Lipczak et al. [14] which was submitted to the ECML PKDD 2009 competition and won in the categories 'content-based' and 'online' tag recommendation and placed third in the 'graph-based' category. We would like to investigate how multiple tag recommenders can be combined and optimized.

RQ3. Is there an optimal number of tags to recommend?

Most related research that deals with tag recommendation checks accuracy by testing a range of tags, usually one to twenty tags. A real world system cannot simply return 'a range of tags', but must return a specific number of tags to the user. Since we're dealing with a real world system which isn't used in an experimental setting, we are interested in the optimal number of tags to recommend. This could be a variable number, e.g. depending on a minimum score threshold or user preference.

1.7. Thesis structure

This thesis is organized as follows. Chapter 2 describes several 'building blocks' for a tag recommendation system, e.g. the general functioning of a typical tag recommendation system, open data for testing the performance of tag recommendation algorithms, etc. Before entering resources into a recommendation system, several 'preprocessing' techniques can filter garbage from the resources which helps achieving better tag recommendation results. These techniques are discussed in Chapter 3. Next, we discuss various tag recommendation techniques in Chapter 4. We suggest two new algorithms in Chapter 5. All preprocessing techniques and recommendation algorithms have been implemented and tested. This process has been described in Chapter 6. The experimental settings and results are given in Chapter 7. We finish this thesis with a discussion (Chapter 8) and suggestions for future work (Chapter 9). For readability of the thesis, we discuss the most important results in the chapters mentioned above. The full results are given in the Appendices.

Chapter 2: Building blocks

Several 'building blocks' for creating a tag recommendation system are discussed in this chapter. Existing software is listed in Section 2.1. Data is required for training and testing the performance of recommendation algorithms. Open data, publicly available data useful for training and testing algorithms, is discussed in Section 2.2. Linked datasets, which are huge datasets in which all data is connected to form a network of information, are described in **??**. An overview of the architecture of a tag recommendation system is given in Section 2.3.

2.1. Software solutions

A number of text processing software packages are listed in this section. They offer various tools helpful for the tag recommendation task. All listed packages are open source.

1. **Apache Lucene** (`http://lucene.apache.org`)
   Apache Lucene is an open source library for full text indexing and search. It was first written in Java and has been ported to many other programming languages.
2. **Apache OpenNLP** (`https://opennlp.apache.org`)
   Apache OpenNLP is a Java-based open source library for NLP tasks such as tokenization, sentence segmentation or parsing.
3. **Apache Stanbol** (`https://stanbol.apache.org`)
   Apache Stanbol's goal is to extend traditional content management systems with semantic services.
4. **NLTK** (`http://www.nltk.org`)
   Natural Language Toolkit is a Python library providing text processing tools such as tokenization, stemming or tagging. Tagging in NLTK is not tag suggestion but part-of-speech (POS) labelling, i.e., detecting the category of words in a text. POS labels are linguistic categories such as noun, verb or conjunction.
5. **Language Detection** (`https://code.google.com/p/language-detection`)
   A language detection library made in Java by Google, able to detect 53 languages, trained on Wikipedia articles. The library has been ported to Python (`https://pypi.python.org/pypi/langdetect`).
6. **Topic modelling libraries**
   There are several libraries offering topic modelling tools. These are very helpful since they contain efficient and optimized implementations of topic modelling algorithms.
   (a) **Gensim** (`http://radimrehurek.com/gensim`)
   "Generate similar", or Gensim[22], is a Python library supporting o.a. TF.IDF, LSI and LDA.
   (b) **MALLET** (`http://mallet.cs.umass.edu`)
   "MAchine Learning for LanguagE Toolkit", or MALLET, is a Java library offering several natural language processing tools.

2.2. Open Data

Data from several collaborative tagging services can be used for testing tag recommendation systems. A typical bookmarking or tagging service allows users to bookmark content (e.g. a webpage) and add tags and a description to the content. BibSonomy[1] allows users to tag and share scientific publications. Delicious[2] allows users to tag websites and CiteULike[3] also allows users to save, tag and share academic papers. Stack Exchange[4] is a collection of communities, each covering a specific topic. Users can ask and reply to questions. Each service allows users to add tags from their own vocabulary, thus creating a folksonomy.

Several dumps from tagging services have been made for research purposes. Most of them only consist of the URT triplets and not resource content. For this research, we required at least the resource title. The following datasets were selected:

1. **DeliciousT140**[5]
   DeliciousT140 contains 144574 random documents from the web, most in HTML format, with corresponding tags and users from the Delicious service. It was created June 2008 and published by A. Zubiaga et al. [33].
2. **Wiki10+**[6]
   The Wiki10+ dataset was was created April 2009 and released by the same authors as the DeliciousT140 dataset. It contains 20762 Wikipedia articles and associated users and tags from the Delicious service [32].
3. **MovieLens 10M**[7]
   The GroupLens research group made MovieLens (online movie recommender service) data available in various sizes, containing respectively 100k, 1M and 10M movie ratings. The ratings are expressed using a number between 1 and 5, but tags were also added to some movies. The data is interesting for this research since the type of content is different than the other datasets (movies vs documents). Users, movie titles and tags are given.

Lastly, it is noteworthy to mention Wikipedia serves dumps of all pages. Dumps from the English Wikipedia can be downloaded from `http://dumps.wikimedia.org/enwiki`. Each page including history can be downloaded. The English Wikipedia dump is 10.5GB in .bz2 format. Unpacked its size is almost 50GB and contains a single XML file. Wikipedia articles are written using templates[8]. The template mark-up is contained in the XML file so it has to be stripped before processing the dump. Wikipedia's dumps are valuable since Wikipedia is available in 287 languages[9] and 37 of those languages have over 100,000 articles from which information can be extracted, e.g. topics and entities can be learned from the articles.

2.2.1. Data characteristics

Statistics from all datasets are given in Table 2.1. The densities are expressed in percentages, showing how dense (or sparse) the relations are. For example, if there are 500 users and 1000 resources, and 2000 user-resource relations are given, the density is $\frac{2000}{500 \times 1000} \times 100 = 0.4\%$. The fact that social tagging services are extremely sparse is a large challenge for tag recommendation systems.

[1] `http://www.bibsonomy.org`
[2] `https://delicious.com`
[3] `http://www.citeulike.org`
[4] `http://stackexchange.com`
[5] `http://nlp.uned.es/social-tagging/delicioust140`
[6] `http://nlp.uned.es/social-tagging/wiki10+`
[7] `http://grouplens.org/datasets/movielens`
[8] `http://en.wikipedia.org/wiki/Help:Template`
[9] `http://en.wikipedia.org/wiki/Wikipedia#Language_editions`

	Dataset	DeliciousT140	Wiki10+	MovieLens 10M
	Posts	144574	20762	82103
	URT triplets	2012804	457708	95580
	User/resource density (%)	0.00030	0.00265	0.00013
	User/tag density (%)	0.37895	0.72037	0.06604
	Resource/tag density (%)	0.02071	0.02223	0.05664
	Tag/tag density (%)	0.14251	0.07116	0.02328
Resources	Unique resources	144574	20762	7601
Resources	Avg users per resource	1	1	10.80
Resources	Avg unique users per resource	1	1	7.30
Resources	Avg tags per resource	13.92	22.05	12.57
Resources	Avg unique tags per resource	13.92	22.05	9.36
Users	Unique users	3378	377	4009
Users	Avg resources per user	42.80	55.07	20.48
Users	Avg unique resources per user	42.80	55.07	13.84
Users	Avg tags per user	595.86	1214.08	23.84
Users	Avg unique tags per user	254.72	714.34	10.92
Tags	Unique tags	67217	99162	16529
Tags	Avg tags per post	13.92	22.05	1.16
Tags	Avg users per tag	29.94	4.62	5.78
Tags	Avg unique users per tag	12.80	2.72	2.65
Tags	Avg resources per tag	29.94	4.62	5.78
Tags	Avg unique resources per tag	29.94	4.62	4.30
Tags	Avg related tags per tag	567.68	112.63	9.06
Tags	Avg unique related tags per tag	95.79	70.56	3.85

Table 2.1: Dataset statistics

Most datasets are post-core level 1 datasets (described in Subsection 2.2.2). We see users tagged each resource only once in DeliciousT140 and Wiki10+. This means that tag recommenders are probably more dependant on resource content than graph-based relations given these datasets. We also see large differences between the datasets, e.g. the vocabularies of users in DeliciousT140 and Wiki10+ are quite large, contrary to MovieLens users.

2.2.2. Post-core selection

Tag recommendation algorithms are often evaluated on post-core datasets, which are subsets of complete datasets [12]. Post-core level n of a dataset consists of all posts in which users, resources and tags appear in n or more posts. The goal is to work with dense data to allow algorithms to perform good and reduce the cold start problem.

Over 60% of all users in the DeliciousT140 dataset tagged two or more documents. However, the dataset only contains documents tagged by a single user. Thus, post-core >1 datasets could not be made.

2.3. IR system architecture

A typical information retrieval system consists of two parts: an *offline* and an *online* part. The offline part consists of training a model on a large amount of data, which takes long and is thus impractical to do real-time. During this stage model parameters are learned and indexes are created. The online part is performed real-time. After a user makes a request, a recommender service is expected to return results immediately and it should thus be very efficient. Generally it consists of a simple calculation or lookup in the pre-computed index.

In Figure 2.1 we see a simplified model of such a system: training data is processed in the offline stage and its results are stored in memory or a database. In the online stage, a user submits a document to a recommender service, which processes only the single document. The recommender service fetches the precomputed data, inserts the document into the learned model, and returns the tags.

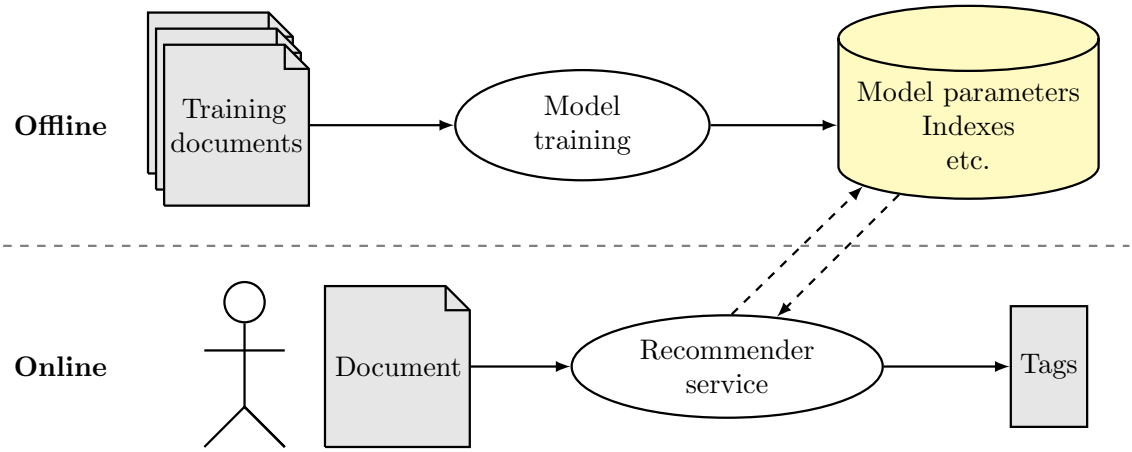


Figure 2.1: IR system architecture

Chapter 3: Text preprocessing

Text preprocessing is intended to transform raw textual data into usable data by removing and/or converting unnecessary data. The expression "garbage in, garbage out" is used in the computer science field to indicate that no matter how good an information processing system is, when one starts with garbage, one always ends with garbage.

This section describes various steps in a text preprocessing-process. Techniques described in this chapter aren't 'holy grails' and are active fields of research. They all have strengths and weaknesses and it is questionable whether the techniques must be used.

3.1. Text clean-up

Documents come from different sources and need to be handled differently in order to extract usable textual data. Many documents from the web come in HTML format, which contain elements such as `<img src="smiley.gif">`, which can be stripped. A text extraction algorithm is required for each format, e.g., HTML, XML, PDF or Word-documents.

Also, a filter for the word length can be applied. Text preprocessing tools generally remove one- or two-letter words such as "a" and "of". A maximum word length can also be applied.

Infrequent words may be erased by setting a minimum frequency threshold. Rare words are unlikely to identify documents and over 50% of distinct words may occur only a single time, so removing infrequent words will yield large savings in document size [6].

Lastly, text can be converted to lowercase characters. The meaning of words will not change, but the number of unique words in a text can be reduced by converting uppercase characters to lowercase characters.

3.2. Stop word removal

Tag words should describe and identify a document. Words such as 'the' or 'and', which typically occur in almost every document are not descriptive and can therefore be filtered out. Highly frequent words can be identified by setting a maximum frequency threshold, e.g. by counting the number of documents a word occurs in, or by maintaining a list of *stop words*, and checking every word in a document against the list of stop words. A word is removed if its frequency appears above the threshold or if it is present in the stop word list. Stop words are language-specific and lists are publicly available for most languages, although domain-specific lists might be used. For example, the word 'can' appears in most public stop word lists, but might be distinctive in the document collection of a recycling company [6].

Stop word removal is an easy method for reducing file size (documents in the DeliciousT140 dataset consist of 20% – 30% stop words) and improving efficiency of the system since stop words aren't indexed.

3.3. Stemming

Stemming is the process of 'linguistic normalisation', i.e., a technique for transforming derivations of a word to the stem of the word. The stem (also called root) of a word is the most basic form of a word. For example, the stem of the words 'driving', 'drove' and 'driven' is 'drive'.

Stemming programs are called stemmers, and unfortunately they are not perfect. Combining words by stemming could reduce the file size of most documents up to 50% [25]. However, some of problems are computational time (stemming algorithms are computationally expensive), accuracy and languages. Most stemming techniques are applied to English language, although stemmers for other languages do exist. The three major stemming algorithms are Porter, Snowball (Porter2) and Lancaster (Paice/Husk), ordered from least to most aggressive.

3.4. Tokenization

Tokenization is the process of splitting text into tokens (chunks of text). A tokenizer can split texts into words, sentences or other units. Generally a rule based system using regular expressions is employed. Western European languages are handled relatively well. Tokenizers are unfortunately not perfect, e.g. a classic example where naive tokenizers fail is "New York-based", where the tokenizer splits at the whitespace. Advanced tokenizers use rules for handling i.a. e-mail addresses, phone numbers, city names and dates.

Multilingual tokenization is complicated. Languages such as Dutch tend to concatenate most compound words (e.g. "huizenprijzen"), while English tends to keep such words separated by dashes or spaces (e.g. "housing prices"). Compound word rules are complex and change over time, making it hard to build perfect tokenizers.

3.5. Part-of-speech tagging

Part-of-speech tagging, or POS tagging, is the process of identifying the lexical class of words. For example the sentence "this research is very interesting":

This	**research**	**is**	**very**	**interesting**
DT (determiner)	NN (singular noun)	VBZ (verb, 3rd person singular present)	RB (adverb)	JJ (adjective)

Table 3.1: POS tagging example

The English language is divided into eight lexical categories. By filtering certain categories the number of words in a document can be vastly reduced. For example, filtering only nouns in the example would result in only "research" and removing 4 out of 5 words.

3.6. Document representation models

After documents are stripped, cleaned and filtered, the last step is to represent the data in a way that is usable by recommender models, also called mid-level representation. One way to store URT triplets is to create a three-dimensional matrix and store the number of times a user u tagged a resource r with tag t (usually once). This requires a $u \times r \times t$-matrix. Most entries will be zero, and given the smallest dataset in this paper (Wiki10+), this results in $20762 \times 377 \times 99162 = 776$ billion entries. If each entry takes one byte, this would result in storing 776 gigabytes of data. This is obviously a very inefficient way of storing the data since non-existing URT triplets also take up one element in the matrix. More efficient mid-level representation models are discussed in this subsection.

3.6.1. Bag of words (BOW)

A common way to represent documents in the form of a matrix is 'bag-of-words' (BOW). From a document, or a collection of documents, an $n \times t$ document-term matrix is constructed, where n is the number of documents and t is the number of distinct terms. For example:

Document 1 Bob walks to the park only after dinner.

Document 2 Bob never walks to the park by himself.

Document 3 Bob always walks with the dog to the park.

From these documents, a dictionary or vocabulary is created where each term is given a unique id:

1. Bob
2. walks
3. to
4. the
5. park
6. only
7. after
8. dinner
9. never
10. by
11. himself
12. always
13. with
14. dog

Next, a document-term matrix is created in which each document is represented by a vector of word-counts. Word order is lost using this model.

Word / Document	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Document 1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
Document 2	1	1	1	1	1	0	0	0	1	1	1	0	0	0
Document 3	1	1	1	2	1	0	0	0	0	0	0	1	1	1

Table 3.2: Documents represented by word count vectors.

This representation faces the same issue as shown in the example at the start of this subsection: the matrix grows very fast as more documents are added (it is in fact the same, but without the third dimension of users). Therefore, a BOW model is stored more efficient using two steps:

1. Create dictionary of words
2. Create sparse BOW vectors

First, all documents in a dataset are read and all distinct words are stored. This forms the dictionary. All words are given a unique identifier. After that, all documents are read a second time, but now a BOW vector is created for each document. Given the example above, documents are represented as follows:

Document 1 [(1,1), (2,1), (3,1), (4,1), (5,1), (6,1), (7,1), (8,1)]

Document 2 [(1,1), (2,1), (3,1), (4,1), (5,1), (9,1), (10,1), (11,1)]

Document 3 [(1,1), (2,1), (3,1), (4,2), (5,1), (12,1), (13,1), (14,1)]

User-tag relations can be stored in similar fashion. This greatly reduces the number of stored elements since non-existing tag assignments are not stored. Since documents and folksonomies are extremely sparse, this is a much needed feature. This model is also abbreviated as $sBoW$ (sparse Bag-of-Words) [31].

3.6.2. TF.IDF

From the BOW model a TF.IDF model (described in Subsection 4.2.2) can be derived in which the IDF scores can be pre-computed for each term. When a new document is processed, only the

BOW-vector for the new document has to be generated and inserted into the TF.IDF model. For each term in a corpus, its IDF score is calculated as:

$$IDF_t = log\frac{D}{df_t} \tag{3.1}$$

Where D is the number of documents in the corpus and df_t is the document frequency of term t (the number of documents in which term t appears). The base of the logarithm is typically set to 2 or 10. Given the three documents above, this results in 14 IDF-scores.

1. 0
2. 0
3. 0
4. 0
5. 0
6. $log(\frac{3}{1})$
7. $log(\frac{3}{1})$
8. $log(\frac{3}{1})$
9. $log(\frac{3}{1})$
10. $log(\frac{3}{1})$
11. $log(\frac{3}{1})$
12. $log(\frac{3}{1})$
13. $log(\frac{3}{1})$
14. $log(\frac{3}{1})$

Since a dictionary can be very large (100000+ terms), the size of the dictionary is often limited to e.g. 50,000 or 100,000 terms. The dictionary of single documents is often limited to a small number of terms such as 100 or 1000. Limiting the dictionary size does have one disadvantage: if a new document is inserted into a TF.IDF model and a term is unknown in the pre-computed IDF scores, a TF.IDF score cannot be computed for the term. However, given a large enough dictionary this should not be a problem since most other terms will be present in the dictionary.

3.6.3. Other representation models

An often discussed limitation of the BOW and TF.IDF model is the lack of ability to handle sentiment [31]. A dictionary can contain 100,000+ terms but a single document contains a much smaller dictionary, e.g. 100 – 1000 terms. Each word is considered to be unique in the BOW and TF.IDF model. However, natural language contains synonyms and thus many terms can be closely related. For example, the English word with the most synonyms is “good” (300+ synonyms). Good can also be described as e.g. “excellent”, “wonderful” or “splendid”. However, without semantics these three words have no relation.

Two representation models without this limitation are Dense Cohort of Terms (dCoT) [31] and Latent Dirichlet Allocation (LDA) [2]. dCoT maps high-dimensional sparse vectors into low-dimensional dense representations. The mapping is trained to reconstruct frequent words from infrequent words. When removing common words from a text, the model should reconstruct the removed words from the remaining words. LDA (described in more detail in Subsubsection 4.2.3) is a topic modelling framework. It models co-occurring words into topics. For example, one could have a topic “computers” with words ‘network’, ‘hardware’, ‘networks’, ‘keyboard’ and ‘mouse’. Another topic could be “animals” containing the words ‘cat’, ‘cats’, ‘dog’, ‘mouse’ and ‘pets’. Since words can have multiple meanings, they can belong to multiple topics.

Chapter 4: Tag recommendation techniques

Tag recommendation techniques are based on different assumptions and are applied to different perspectives of data, e.g. a user's tag history or a document's title. The techniques described in this section are applied in order to predict a set of tags that a user will most likely add to a resource.

Several techniques for tag recommendation have been proposed over the years. Most are variations on one of the few major algorithms. An overview of different types of algorithms and their characteristics is given in this chapter. The types are divided over sections, where each section discusses one type of algorithms. Section 4.1 describes popularity based methods. Content based and graph based methods are respectively addressed in Section 4.2 and Section 4.3 and hybrid approaches, in which models are combined, are discussed in Section 4.4. Lastly, measurements for measuring the performance of algorithms are discussed in Section 4.5.

4.1. Popularity based methods

Popularity-based recommenders are naive methods but very simple and fast to apply since they simply sort tags by frequency.

4.1.1. Global popular tags

The simplest solution is to recommend the top tags in a dataset sorted by frequency. The most popular tags for any user $u \in U$ and resource $r \in R$ are calculated from the set Y_t (where $|Y|$ denotes the cardinality of set Y):

$$\hat{T}_{u,r} \coloneqq \underset{t \in T}{\operatorname{argmax}^{n}}(|Y_t|) \tag{4.1}$$

4.1.2. User popular tags

Variations shown in [26] select the most popular tags by (a combination of) user, resource or item. For example, the following recommends the most popular tags from a user's set of tags and is thus presumably better at recommending tags (if the user has previously tagged resources):

$$\hat{T}_{u,r} \coloneqq \underset{t \in T}{\operatorname{argmax}^{n}}(|Y_{u,t}|) \tag{4.2}$$

These recommenders are very simple and fast to use, but have several drawbacks. First, they suffer from the cold start problem. The global popular tag recommender suffers only minimally from this problem, but more specific recommenders require more training in order to produce relevant tags. When a user processes his first document, he won't have a tag history and the global tag recommender can be used as a backup.

4.2. Content based methods

Content based recommendation techniques focus on extracting features and structure from resources themselves, such as the title, headings or anchor tags.

4.2.1. Content sources

Different sources of content can be URL, resource content, resource title, etc. File types such as PDF store data unstructured making it difficult to extract text according to any structure from the document. Other file types such as XML and HTML store the data structured, allowing to extract separate pieces of documents. These types are referred to as *semi-structured*, since they don't reside in a database but they do contain elements which follow a pre-defined structure. The impact of document titles was researched by Lipczak et al. in [15]. Findings showed "a strong relation between the resource title and the choice of tags".

We checked this claim on the DeliciousT140 dataset. Titles, h1 to h6 tags, anchor tags and remaining content were separately extracted from HTML documents and the ratio between words and user-tags was averaged over all documents. The user-tag-density of each source, ordered from most to least dense, showed to be (1) title, (2) h1, (3) anchor tags, (4) h2, (5) h3, (6) remaining content, (7) h4, (8) h5 and (9) h6. The experiments and results are discussed in Section 7.1.

4.2.2. TF.IDF

Term Frequency Inverse Document Frequency (TF.IDF) is a term-weighting algorithm which shows the importance of a term t within a document d within a corpus D. The IDF function was proposed in 1972 [23].

Term Frequency

The *term frequency* $TF_{t,d}$ defines the frequency of term t in document d. The TF value is typically normalized in some way to prevent a bias towards longer documents. For example:

$$TF_{t,d} = n_{t,d} \tag{4.3}$$

$$TF_{t,d} = \frac{n_{t,d}}{\sum_k n_{k,d}} \tag{4.4}$$

$$TF_{t,d} = \frac{n_{t,d}}{\max_k n_{k,d}} \tag{4.5}$$

The variable $n_{t,d}$ is the number of occurrences of term t in document d. The denominator in Equation 4.4 denotes the sum of all occurrences of all terms in a document d, which is the number of words in the document. The denominator in Equation 4.5 gives the maximum number of occurrences of any term in a document.

Inverse Document Frequency

The *inverse document frequency* IDF_t shows the rarity of a term in a document collection:

$$IDF_{t,D} = log\frac{D}{df_t} \tag{4.6}$$

Where D is the number of documents in the collection and df_t is the document frequency of term t, the number of documents containing the term. The IDF scores can be pre-computed from a

corpus, after which only the TF scores (small computation) will need to be computed and IDF scores fetched for calculating the complete TF.IDF score.

TF.IDF

The TF.IDF statistic is calculated as $TFIDF_{t,d,D} = TF_{t,d} \times IDF_{t,D}$. The idea is that words appearing in all documents become less important because of the IDF measure, and words appearing a lot in a single document become important as a result of the TF measure. When processing a previously unseen document, the TF scores for the given document are calculated and multiplied by the pre-computed IDF scores.

There are several drawbacks. State of the art research focuses on semantics, i.e., identification of topics, since language is complex and contains synonyms. The vocabulary of TF.IDF is limited to the words within a document, which are often not sufficient as tags. There's no way to deal with synonyms or polysemes. From a performance viewpoint, pre-processing the corpus takes a large amount of memory and requires a lot of computational time. To train the model, more documents have to be added, increasing the required memory to store the processed corpus.

Also, normalization of the TF vector does not benefit the tag recommendation task in which the most important terms for individual documents are determined. For example, given a document with three terms with respective TF scores 5, 10 and 15. Normalization will not change the proportion between the scores, e.g. maximum frequency normalization will result in $\frac{1}{6}$, $\frac{1}{3}$ and $\frac{1}{2}$. If the document contents would be duplicated the resulting TF scores will be 10, 20 and 30. If these scores were to be compared to the original document's TF scores, the duplicated document's TF scores are higher since there are more words in the document. Normalization will balance these scores and benefit applications such as document similarity measures.

Variations

Variations on TF.IDF have been proposed in order to emphasize different characteristics of documents. Basili et al. [4] proposed IWF, *Inverse Word Frequency*, as a variation on IDF:

$$IWF_{t,D} = log\frac{N}{f_t} \tag{4.7}$$

Where N is the sum of all frequencies of words, i.e. the total number of words in the corpus. The variable f_t is the overall frequency of term t, instead of the the document frequency used in the IDF measure. The IWF measure is also referred to as *Inverse Term Frequency* (ITF) in related literature. Also, to compute the final importance of a term, Basili squared the IWF measure since it is too biased by the term frequency [4].

$$TF.IWF_{t,d} = TF \times IWF \times IWF = n_{t,d} \times \left(log\frac{N}{f_t}\right)^2 \tag{4.8}$$

Other variations are included in the SMART (System for the Mechanical Analysis and Retrieval of Text) Information Retrieval System, developed at Cornell University [24]. Some search engines allow different weighting schemes for queries vs. documents. The SMART model defines several weighting combinations.

The SMART notation is written in the format "*ddd.qqq*", where the *ddd* specifies a document vector weighting scheme and *qqq* specifies a query vector weighting scheme, e.g. *lnc.ltc*. We can apply the term and document frequency variants to the tag recommendation task.

Term Frequency	Document Frequency	Normalization
n (natural): $tf_{t,d}$	n (no): 1	n (none): 1
l (logarithm): $1 + log(tf_{t,d})$	t (idf): $log(\frac{N}{df_t})$	c (cosine): $\frac{1}{\sqrt{w_1^2+w_2^2+...+w_M^2}}$
a (augmented): $0.5 + \frac{0.5 \times tf_{t,d}}{max(tf_{t,d})}$	p (prob idf): $max\left(0, log\frac{N-df_t}{df_t}\right)$	b (byte size): $1/CharLength^{\alpha}, \alpha < 1$
b (boolean): $\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		
L (log average): $\frac{1+log(tf_{t,d})}{1+log(ave_{t \in d}(tf_{t,d}))}$		

Table 4.1: SMART system

4.2.3. Topic modelling

The concept of *topic modelling* was first addressed by Papadimitriou et al. in 1998 [20]. Major publications were made by T. Hofmann with the introduction of Probabilistic Latent Semantic Indexing (PLSI) in 1999 [8] and D. Blei with the introduction of Latent Dirichlet Allocation (LDA) in 2003 [2].

Most search engines such as Google Search use algorithms based on keywords, number of connecting pages, etc. to find webpages related to a query. Since natural language contains synonymy, polysemy and other word relationships, simple search engines are claimed to be inaccurate. For example, a document could contain words such as "auto", "bonnet" and "boot", while a user searches for "car", "hood" or "trunk". Semantic search engines (e.g. DuckDuckGo[1]) take word semantics into account. E.g., a query for "set" on DuckDuckGo first shows multiple meanings of the word 'set' in the fields of 'Mathematics and Programming', 'Chemistry', 'Psychology', 'Technology', etc.

Topic models are mathematical frameworks for discovering semantically coherent clusters of words, known as topics. The topics are called *latent* topics, which are present but hidden topics. Figure 4.1 shows the challenge of topic modelling: there are various topics and words can be mapped to one or more topics.

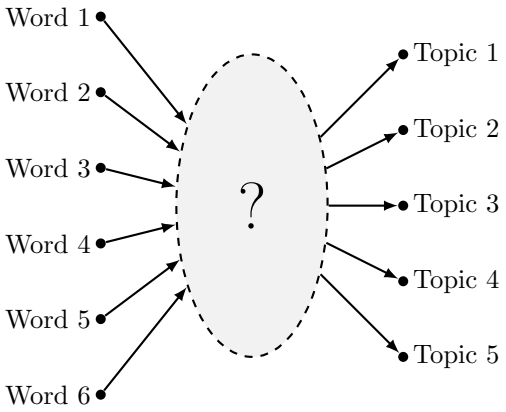


Figure 4.1: Problem of mapping topics to words

The idea behind topic modelling is that documents consist of topics, which are distributions over words. In the language modelling domain, topics are considered as 'a probability distribution over

[1] `https://duckduckgo.com`

a vocabulary of words'. Each word in a vocabulary has a probability between 0 and 1 to belong to each topic.

LDA

Latent Dirichlet Allocation (LDA) is a topic modelling framework. It cannot be applied directly to the tag recommendation challenge as it finds topics that belong to a document, but it is unclear how tags can be suggested from those topics. LDA was applied to tag recommendation by Krestel et al. [11], but they trained an LDA model on tag-words from already tagged resources with the goal of extending the set of tags for each resource. This research focusses on recommending tags for previously untagged resources. However, their method for suggesting tags from learnt topics is applicable. Per word, they select the topic with the highest probability for that word, and select all words from the topic with a probability above a certain threshold.

Diaz-Aviles et al. [5] applied LDA to tag recommendation for previously untagged documents. They extract a document's content and combine this with the titles and summaries from the top ten search results of BOSS (Yahoo!'s open search platform). They train an LDA model 'on-the-fly' on the combined corpus and select the first word from all topics (the number of topics was varied).

An extension to the LDA model was published by Si et al. [27] which they called 'Tag-LDA'. Standard LDA models documents, topics and words. Tag-LDA also models tags. After training their model, recommendations are made for new documents by inferring the probability for tags given the document and the top n tags are selected.

We see three different approaches for recommending tags for a document: (1) select top n words from topic with highest probability, (2) select first word from top n topics sorted by probability and (3) infer tag probabilities from Tag-LDA model. Unfortunately all three methods were applied on different datasets.

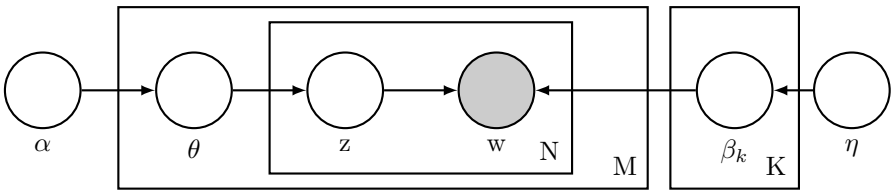


Figure 4.2: Smoothed LDA model

Due to complexity and availability of great topic modelling libraries, we use the gensim Python library for testing with LDA. An older topic modelling framework called LSI is also provided. We implemented a tag recommendation function as suggested in (1) ourself.

4.3. Graph based methods

Graph based tag recommendation focusses on learning relations between users, resources and tags from previous posts. These algorithms are typically tested on post core level > 1 datasets. Well-known algorithms are collaborative filtering (applied to the tag recommendation task in i.a. [18, 10, 29]) and FolkRank [10]. In collaborative filtering, the idea is to recommend tags based on the recommendations on similar resources. Similar resources are determined by calculating a similarity measure such as Cosine similarity between all resources, after which a set of tags is selected from those resources.

In this research we focus on content-based tag recommendation, the suggestion of tags on previously unseen resources in post core level 1 datasets. Therefore we do no include graph-based techniques in the experiments.

4.4. Hybrid approaches

Various techniques has been proposed for tag recommendation. However, the number of recommenders in which techniques are combined to 'join forces' appears to be limited. The goal of a hybrid approach is to combine multiple 'weak' recommenders into a single 'strong' recommender, as shown in Figure 4.3.

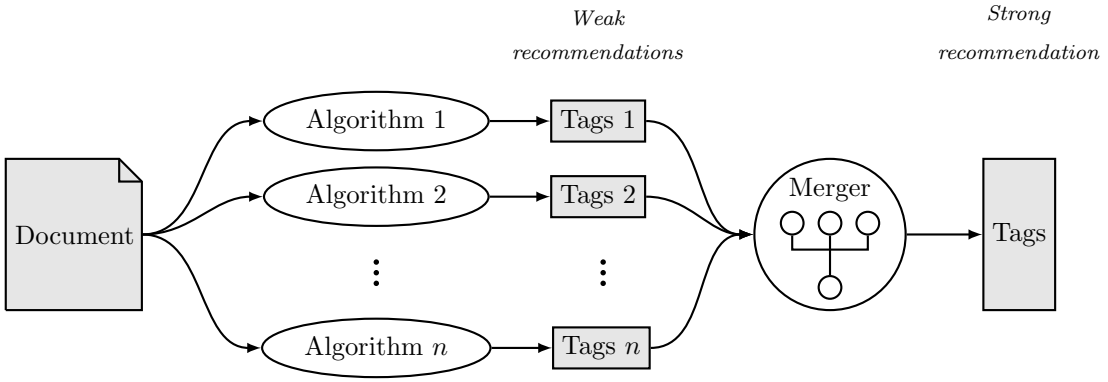


Figure 4.3: Hybrid tag recommendation system setup

Lipczak et al. published two papers in which multiple recommenders are combined [14, 16]. All recommenders produce a set of tags with scores $s \in [0, 1]$. Each recommender's set of tags is combined by a merger which takes two recommendation sets and re-scores the tags given a merge coefficient $p_{merge} \in [0, 1]$ which represents the importance of both recommendations. The merge value determines the accuracy of the system and is learned from the training data.

Burke [3] lists seven types of methods for combining multiple recommenders:

Type	Description
Weighted	The scores (or votes) of several recommendation techniques are combined together to produce a single recommendation.
Switching	The system switches between recommendation techniques depending on the current situation.
Mixed	Recommendations from several different recommenders are presented at the same time.
Feature combination	Features from different recommendation data sources are thrown together into a single recommendation algorithm.
Cascade	One recommender refines the recommendations given by another.
Feature augmentation	Output from one technique is used as an input feature to another.
Meta-level	The model learned by one recommender is used as input to another.

Table 4.2: Hybridization method types as listed by Burke

In other types of recommender systems combining models is very common to achieve better accuracy. In the Netflix challenge, a million-dollar prize competition where the goal was to predict user ratings as accurately as possible, the winning team used gradient boosted decision trees to combine over 500 models [9]. Using a large amount of models might improve accuracy by minimal differences but will not benefit efficiency in a real-time system. However, using a combination of just three models an RMSE close to the winning RMSE was be achieved[2].

4.4.1. Ensemble types

Ensemble learning is the field of research concerned with combining multiple models in order to solve computational challenges [17]. *Data fusion* is a particular application of ensemble learning. A tag recommender system will output a set of tags and multiple recommenders will output multiple sets of tags. Combining those sets can result in improved accuracy and is known as data fusion.

[2] `http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary`

Bagging

Bagging (Bootstrap Aggregation) is designed to reduce the variance of predictions. Each model in the ensemble has equal weight. In order to reduce model variance, each model in the ensemble is trained using a randomly drawn subset of the training set.

Boosting

Boosting algorithms try to combine methods using a weighted average approach.

Stacking

Stacking consists of two phases: first individual models are learned. Next, the output of all models is given to a second learning algorithm producing the final results.

4.4.2. Ensembles for tag recommendation

Fox and Shaw defined several popular combination methods [7] which have become the standard methods for data fusion. Before combining, the scores in each list must be normalized within that list. The most used normalization was proposed by Lee [13]:

$$score_{normalized} = \frac{score_{original} - score_{min}}{score_{max} - score_{min}} \tag{4.9}$$

After normalization, Fox's and Shaw's methods shown in Table 4.3 can be applied. There are k recommendation models and each model returns tags t_j with an associated score s_{kj} [19], e.g.:

$$\begin{gathered}
[(t_1, s_{11}), (t_2, s_{12}), (t_3, s_{13}), (t_j, s_{1j})] \\
[(t_1, s_{21}), (t_4, s_{24}), (t_5, s_{25}), (t_j, s_{2j})] \\
[(t_3, s_{33}), (t_4, s_{34}), (t_6, s_{36}), (t_j, s_{3j})] \\
\vdots \\
[(t_j, s_{kj}), (t_j, s_{kj}), (t_j, s_{kj}), (t_j, s_{kj})]
\end{gathered}$$

Where $j = \{1, 2, \dots\}$ (j word ids in dictionary) and $k = \{1, 2, \dots\}$ (k recommenders). Each recommender returns a (different) set of words. As we see in the example, word t_1 is recommended by recommenders k_1 and k_2, but not by k_3.

The methods listed above are unweighted, i.e., they all have equal influence in the final recommendation. Intuitively this is odd: each recommendation engine's performance is differently, so they should influence the final result differently. Bartell et al. [1] applied a linear combination to combine multiple recommenders:

$$combined = \sum_{n=1}^{N} w_n * s_{kj} \tag{4.15}$$

Where weight w in range [0,1]. The next subsection describes Lipczak's system in which a combination of weighted recommendations are applied, and weights are learned on a part of the data.

Method	Description
CombMIN	Minimum of all scores for a tag. $$CombMIN(t_j) = min(s_{1j}, s_{2j}, \ldots, s_{kj}) \quad (4.10)$$
CombMAX	Maximum of all scores for a tag. $$CombMAX(t_j) = max(s_{1j}, s_{2j}, \ldots, s_{kj}) \quad (4.11)$$
CombSUM	The sum of all scores for a tag. $$CombSUM(t_j) = \sum_{k}^{i=1} s_{ij} \quad (4.12)$$
CombANZ	The CombSUM score divided by number of times a tag was recommended. $$CombANZ(t_j) = \frac{CombSUM(t_j)}{\lvert\{s_{1j}, s_{2j}, \ldots, s_{kj}\}\rvert} \quad (4.13)$$
CombMNZ	The CombSUM score times the number of times a tag was recommended. $$CombANZ(t_j) = CombSUM(t_j) \times \lvert\{s_{1j}, s_{2j}, \ldots, s_{kj}\}\rvert \quad (4.14)$$

Table 4.3: Combination methods proposed by Fox and Shaw

4.4.3. Lipczak's tag recommendation system

Lipczak et al. presented a system [16, 14] consisting of multiple tag recommenders which are combined to output a single recommendation. His system won 2 out of 3 tasks at the ECML PKDD 2009 challenge. The system was applied to BibSonomy data. His system consists of six basic recommenders:

1. URL recommender
2. Title recommender
3. TitleToTag recommender
4. TagToTag recommender
5. Resource recommender
6. User recommender

The URL and title recommenders are trained on words found in respectively the URL and title. Each word's score was calculated as the number of times the word was used in a title/URL and as a tag, divided by the number of times the word was used in the title/URL, as in Equation 4.16.

$$score_{word} = \frac{|frequency_{(title \cap tag)}|}{|frequency_{title}|} \quad (4.16)$$

Words with a score below 0.05 were removed, these are assumed to be of too low quality. The resulting word scores are rescored using a 'leading precision rescorer' as in Equation 4.17 since they're the result of independent recommenders. This ensures that tags from more accurate

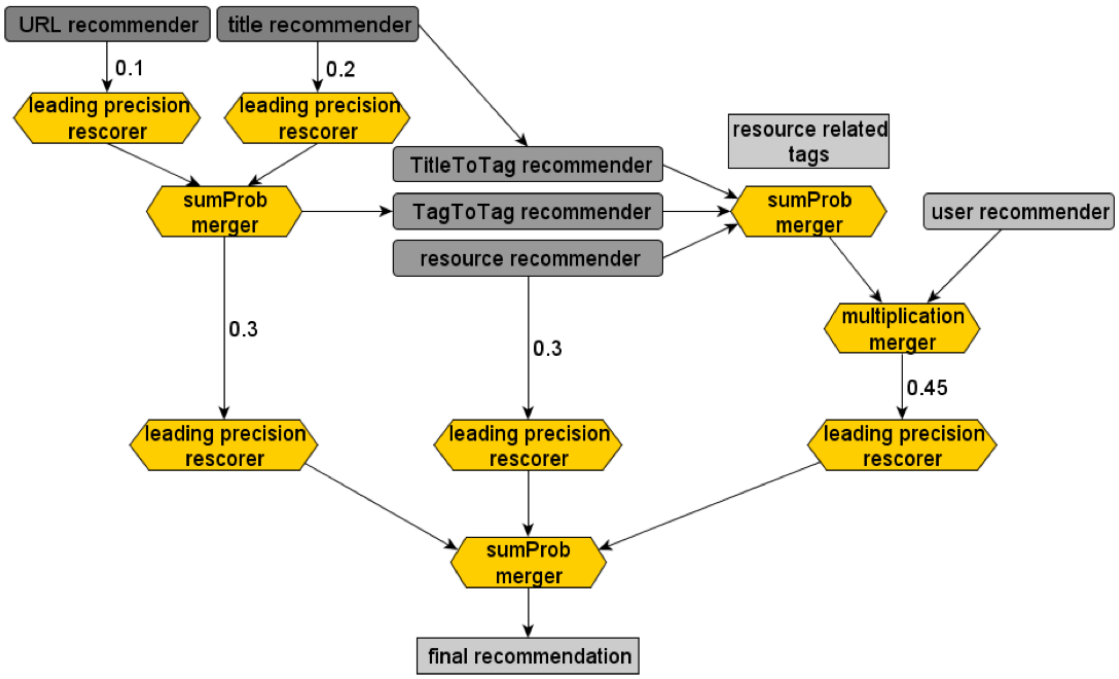


Figure 4.4: Lipczak's tag recommendation system

recommenders will have higher scores in the final tag recommendation. The weights are displayed next to the arrows in the Figure 4.4.

$$s_i^{'} = \frac{avgPrecisionAt1 \times s_i}{s_1} \tag{4.17}$$

Each recommender returns a set of tags t with scores s: (t_j, s_{kj}). The Precision $\left(\frac{|correct\ tags|}{|recommended\ tags|}\right)$ for each recommendation was calculated and the highest score s_1 was given the average precision score. Following scores s_i were re-calculated using the leading precision rescorer.

The result of the title recommender is used as input for the TitleToTag recommender. For each word in each post title, the related tags are stored. The related tag scores are calculated as the title recommender's score times the *TitleToTag* confidence, which is the co-occurrence frequency of the titleword and the tagword, divided by the frequency of the titleword:

$$score_{TitleToTag} = score_{TitleRecommender} \times \frac{frequency_{(titleword \cap tagword)}}{frequency_{titleword}} \tag{4.18}$$

In similar fashion, the combined result of the URL and title recommenders is used as input for the TagToTag recommender. This recommender captures tag-to-tag relations and the word scores are calculated as the combined tag scores times the *TagToTag* confidence, which is the co-occurrence frequency of two words t_1 and t_2, divided by the frequency of t_1. The 'sumProb merger' shown in Figure 4.4 merges two probabilities, i.e. tag scores, using the equation in Equation 4.19:

$$score_{merged} = 1 - \prod_i (1 - score_i) \tag{4.19}$$

Lastly, the resource and user recommenders base their recommendations on respectively the resource tag history and the user tag history. The resource tag scores are calculated as number of

occurrences of the tag divided by the resource frequency. The user recommender is a combination of tag frequency and tag recency in the user's set of tags. The two sets are merged by testing for the best Precision and combined into a single set of user related tags.

The 'multiplication merger' merges two sets of tag recommendations by multiplying the scores of tags occurring in both sets.

Lipczak's system uses a combination of ensemble methods listed in the previous subsection. The URL and title recommender's input is merged using a stacking technique. The (merged) result is then passed the TitleToTag and TagToTag recommenders, called meta-level hybridization by Burke. The scores are normalized by the leading precision rescorer and the resource and user related tags are merged using a multiplication merger which is similar to the methods proposed by Fox and Shaw.

Data flow

To demonstrate the functioning of Lipczak's system, we show the performed calculations by processing a post with the title "Content-based tag recommendation algorithms for unstructured data". The results from each step are shown in Table 4.4. Step (1) is preprocessing of the title. We skip some preprocessing methods for this example and apply tokenisation, conversion to lowercase and stop word removal, which results in ["content-based", "tag", "recommendation", "algorithms", "unstructured", "data"]. In step (2), the tokens are used as input for the TitleRecommender. This returns the same tokens, paired with a score. In step (3), the TitleRecommender's result is used as input for the TitleToTag and TagToTag recommenders. These recommenders retrieve a list of related words and scores for each word. In step (4), the related scores are multiplied by the scores from the TitleRecommender. Next, in step (5), all scores are merged using the SumProb merger in Equation 4.19. Thereafter, in step (6), all scores are rescored by a pre-computed merge coefficient. The value of the merge coefficient is determined by rescoring using a range of values and testing the Recall@5. Let's assume the merge coefficient between the TitleToTagRecommender and the TagToTagRecommender is 0.3, meaning TitleToTagRecommender*0.3 and TagToTagRecommender*(1-0.3). Finally, in step (7), we merge the results of both recommendations using, again, the SumProb merger.

For illustration, we created the network in Figure 4.5. Each node represents a word. A post containing three words is inserted, activating three nodes in the Title recommender. The title recommender result is forwarded to the TitleToTag and TagToTag recommender. In the TitleToTag recommender, each word contains a set of related words. Once a word is activated, the set of related words (red nodes in the TitleToTag recommender) is forwarded for the final result. In the TagToTag recommender, the words form a different network structure. Each word can be connected to other words. When words from the Title recommender are passed to the TagToTag recommender, related words are selected for recommendation from the TagToTag network (red nodes). We apply a single pass for selecting related words.

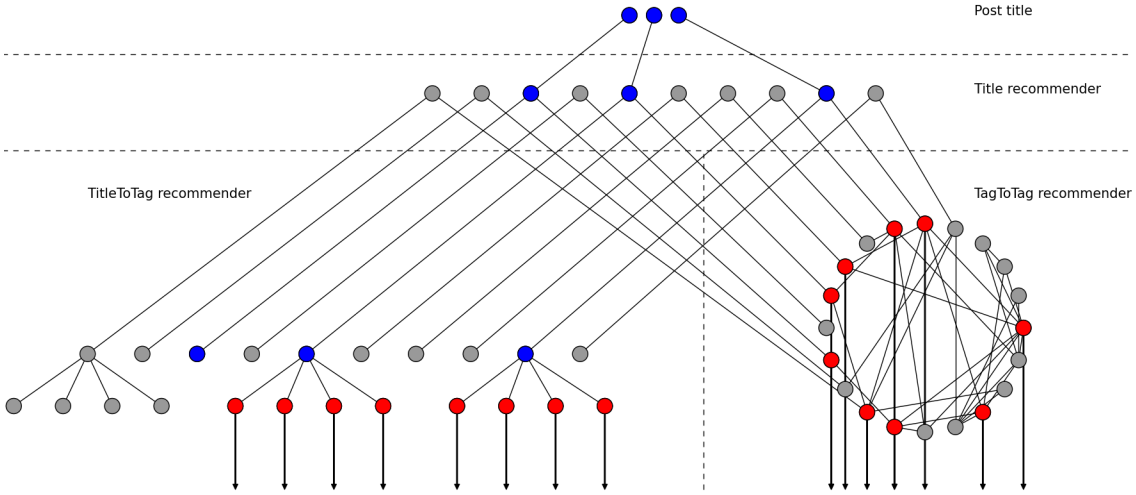


Figure 4.5: Network of words in Lipczak's system. Blue nodes are selected from the post's title but not recommended. Red nodes are recommended.

Raw title	"Content-based tag recommendation algorithms for unstructured data"
(1) Preprocessed title	["content-based", "tag", "recommendation", "algorithms", "unstructured", "data"]
(2) TitleRecommender	[("content-based", 0.7), ("tag", 0.8), ("recommendation", 0.6), ("algorithms", 0.4), ("unstructured", 0.2), ("data", 0.3)]
(3) TitleToTagRecommender result	("content-based", 0.7): [("recommendation", 0.8), ("content", 0.4), ("instruction", 0.3)] ("tag", 0.8): [("tags", 0.9), ("metadata", 0.6), ("documents", 0.5)] ("recommendation", 0.6): [("recommendation", 0.9), ("collaborative", 0.4), ("system", 0.3)] ("algorithms", 0.4): [("model", 0.9), ("technique", 0.8), ("recommendation", 0.7)] ("unstructured", 0.2): [("data", 0.9), ("documents", 0.6), ("database", 0.5)] ("data", 0.3): [("data", 1), ("mining", 0.8), ("processing", 0.7)]
(3) TagToTagRecommender result	("content-based", 0.7): [("recommendation", 0.9), ("data", 0.8), ("algorithms", 0.7)] ("tag", 0.8): [("tags", 0.9), ("documents", 0.6), ("content", 0.5)] ("recommendation", 0.6): [("collaborative", 0.9), ("documents", 0.6), ("system", 0.5)] ("algorithms", 0.4): [("math", 0.9), ("technique", 0.8), ("system", 0.7)] ("unstructured", 0.2): [("data", 0.9), ("documents", 0.8), ("pdf", 0.4)] ("data", 0.3): [("science", 0.9), ("mining", 0.9), ("text", 0.7)]
(4) TitleToTagRecommender score multiplication	[("recommendation", 0.7*0.8=0.56), ("content", 0.7*0.4=0.28), ("instruction", 0.7*0.3=0.21)] [("tags", 0.8*0.9=0.72), ("metadata", 0.8*0.6=0.48), ("documents", 0.8*0.5=0.4)] [("recommendation", 0.6*0.9=0.54), ("collaborative", 0.6*0.4=0.24), ("system", 0.6*0.3=0.18)] [("model", 0.4*0.9=0.36), ("technique", 0.4*0.8=0.32), ("recommendation", 0.4*0.7=0.28)] [("data", 0.2*0.9=0.18), ("documents", 0.2*0.6=0.12), ("database", 0.2*0.5=0.1)] [("data", 0.3*1=0.3), ("mining", 0.3*0.8=0.24), ("processing", 0.3*0.7=0.21)]
(4) TagToTagRecommender score multiplication	[("recommendation", 0.7*0.9=0.63), ("data", 0.7*0.8=0.56), ("algorithms", 0.7*0.7=0.49)] [("tags", 0.8*0.9=0.72), ("documents", 0.8*0.6=0.48), ("content", 0.8*0.5=0.4)] [("collaborative", 0.6*0.9=0.54), ("documents", 0.6*0.6=0.36), ("system", 0.6*0.5=0.3)] [("math", 0.4*0.9=0.36), ("technique", 0.4*0.8=0.32), ("system", 0.4*0.7=0.28)] [("data", 0.2*0.9=0.18), ("documents", 0.2*0.8=0.16), ("pdf", 0.2*0.4=0.08)] [("science", 0.3*0.9=0.27), ("mining", 0.3*0.9=0.27), ("text", 0.3*0.7=0.21)]
(5) TitleToTagRecommender result merge	[("recommendation", 0.854272), ("tags", 0.72), ("metadata", 0.48), ("documents", 0.472), ("data", 0.426), ("model", 0.36), ("technique", 0.32), ("content", 0.28), ("mining", 0.24), ("collaborative", 0.24), ("instruction", 0.21), ("processing", 0.21), ("system", 0.18), ("database", 0.1)]
(5) TagToTagRecommender result merge	[("documents", 0.720448), ("tags", 0.72), ("data", 0.6392), ("recommendation", 0.63), ("collaborative", 0.54), ("system", 0.496), ("algorithms", 0.49), ("content", 0.4), ("math", 0.36), ("technique", 0.32), ("mining", 0.27), ("science", 0.27), ("text", 0.21), ("pdf", 0.08)]
(6) TitleToTagRecommender result rescoring	[("recommendation", 0.3000), ("tags", 0.2528), ("metadata", 0.1686), ("documents", 0.1658), ("data", 0.1496), ("model", 0.1264), ("technique", 0.1124), ("content", 0.0983), ("mining", 0.0843), ("collaborative", 0.0843), ("instruction", 0.0737), ("processing", 0.0737), ("system", 0.0632), ("database", 0.0351)]
(6) TagToTagRecommender result rescoring	[("documents", 0.7000), ("tags", 0.6996), ("data", 0.6211), ("recommendation", 0.6121), ("collaborative", 0.5247), ("system", 0.4819), ("algorithms", 0.4761), ("content", 0.3886), ("math", 0.3498), ("technique", 0.3109), ("mining", 0.2623), ("science", 0.2623), ("text", 0.2040), ("pdf", 0.0777)]
(7) Merged result	[("tags", 0.7755), ("documents", 0.7497), ("recommendation", 0.7285), ("data", 0.6778), ("collaborative", 0.5648), ("system", 0.5146), ("algorithms", 0.4761), ("content", 0.4487), ("technique", 0.3884), ("math", 0.3498), ("mining", 0.3245), ("science", 0.2623), ("text", 0.2040), ("metadata", 0.1686), ("model", 0.1264), ("pdf", 0.0777), ("processing", 0.0737), ("instruction", 0.0737), ("database", 0.0351)]

Table 4.4: Data flow in Lipczak's system

4.5. Evaluation

The performance of a tag recommendation algorithm can be measured by comparing the recommended tags with tags added by users. The user supplied tags are seen as 'true' tags.

4.5.1. Cross-validation

Cross-validation (CV) is a method for checking the performance of predictive systems. Data is first split into n folds. In Figure 4.6 we see an example of 5-fold cross validation. The data is tested n times, where each testcase consists of one fold as testset while the other folds serve as the training set. The data serving as testset and training set alternates in all tests and results are averaged.

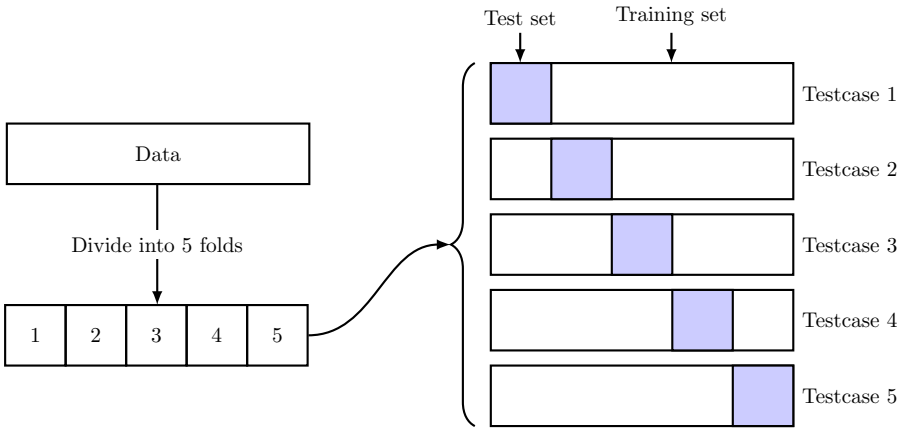


Figure 4.6: 5-fold cross-validation setup

4.5.2. Performance metrics

In order to evaluate the accuracy of tag recommender systems, three performance metrics are applied. We first identify four types of results:

From the confusion matrix in Figure 4.7 we can derive the following performance metrics:

1. **Precision**
   Precision measures the ratio between the correctly recommended tags and the total recommended tags. It returns a value between 0 and 1, where 0 means none of the recommended tags match the user tags and 1 means that all recommended tags are also user tags, although user tags might be missed.

$$precision = \frac{|TP|}{|TP+FP|} = \frac{|\text{correct tags}|}{|\text{recommended tags}|}$$

2. **Recall**
   Recall measures the ratio between the correctly recommended tags and the number of user tags. It also returns a value between 0 and 1, where 0 means none of the recommended tags match the user tags and 1 means that all user tags were recommended. However, recommended tags which are not user tags (FP) are not taken into account and thus a lot of false tags could be included.

$$recall = \frac{|TP|}{|TP+FN|} = \frac{|\text{correct tags}|}{|\text{user tags}|}$$

3. **F1**
   Using either Precision or Recall we cannot determine the optimal performance of an algorithm since they both lack information. The F_1 measure combines both:

		User	
		User tags (positive)	Not user tags (negative)
Recommended	Recommended tags (positive)	User & recommended tags **True positive (TP)**	Not user & recommended tags **False positive (FP)**
	Not recommended tags (negative)	User & not recommended tags **False negative (FN)**	Not user & not recommended tags **True negative (TN)**

Figure 4.7: Recommendation confusion matrix

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

In Figure 4.8[3] we see Precision versus Recall, and the blue lines show the respective F1 scores. The top left, where precision = 1 and recall = 0, means only matching user tags were recommended but doesn't tell how many, i.e. a lot of user tags might not be recommended. The bottom right, where precision = 0 and recall = 1, means all user tags were recommended, but doesn't tell how many non user tags were recommended, i.e. false tags might also be recommended.

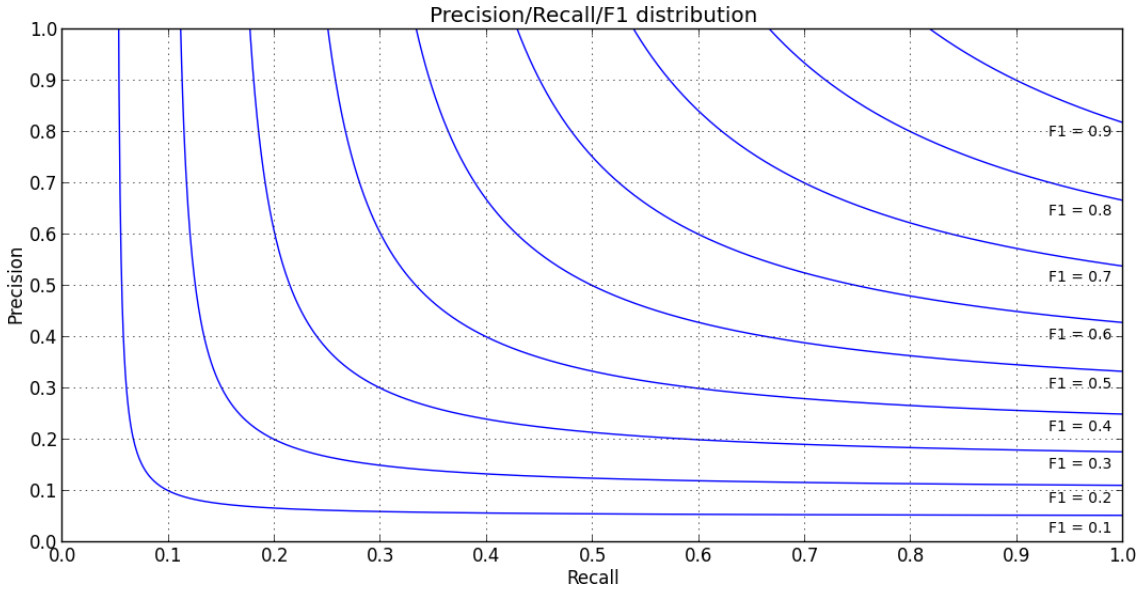


Figure 4.8: Distribution of F1 scores given Precision and Recall

By combining Precision and Recall into the F1 measure a ratio is obtained between the recommended matching user tags (TP), recommended but not matching user tags (FP) and

[3]Source code used from `https://github.com/marcelcaraciolo/pyrecommender`

not recommended user tags (FN). An F1 score of 1 (precision and recall both 1) means that all recommended tags are also user tags (TP), and no other tags were recommended, i.e. the recommended tags perfectly match the user tags.

Chapter 5: Proposed techniques

We discussed various preprocessing techniques, document sources for tags and tag recommendation techniques. During this research we implemented and tested all techniques and developed ideas for improvement of the discussed theories. We found that Lipczak's algorithm worked best on content-based tag recommendation. His recommendation engine combines very basic recommendation models to produce a single, very precise recommendation. We believe this is since his model is the only model that learns relationships between titlewords and tagwords. In fact, while we managed to improve the F1 scores on all algorithms by applying preprocessing techniques, we did not manage to do so on Lipczak's model (F1 actually decreased very slightly) since he learns this title-tag relationship. All preprocessing techniques filter out mostly garbage, but also filter out some relevant words. When applying models based on words frequencies, preprocessing will improve the F1 results since garbage is filtered out. When applying Lipczak's model, the garbage-words will never appear in the final recommendation since only words used as tags by users are recommended (which aren't considered as garbage). Preprocessing will filter out a couple of those words, leading to a lower F1 score.

Based on Lipczak's model, we developed two ideas:

1. Expansion of the TitleRecommender's space by adding a title-to-title recommender.
2. Weighting of the recommendation based on tag popularity.

5.1. Expansion of title search space

A document's title is used as input for the system. When the title is very short, e.g. 1 word, its tag search space is quite small. To expand the search space for relevant tags, we suggest adding a 'title-to-title' recommender. This works in similar fashion as the TitleToTag and TagToTag recommenders. When training the model on a set of documents, words appearing together in document titles are stored as shown in Figure 5.1. For each word in the document's title in the training set, we produce a set of related words. The related word scores are calculated as:

$$s_{t_1 \to t_2} = \frac{frequency_{(t_1 \cap t_2)}}{frequency_{t_1}} \tag{5.1}$$

Where the variable $s_{t_1 \to t_2}$ represents the relation score from term t_1 to t_2. When a document is inserted, the title-to-title recommender will produce a set of related words for each word in the title. The sets are combined using the sumProb merger. Next, the TitleRecommender and TitleToTitle recommender's merge coefficient is learned by testing the Recall@5 on a subset of the training data. The scores are rescored by the leading precision rescorer and merged using the sumProb merger. After this, the resulting set of tags will continue the flow as in the original Lipczak model.

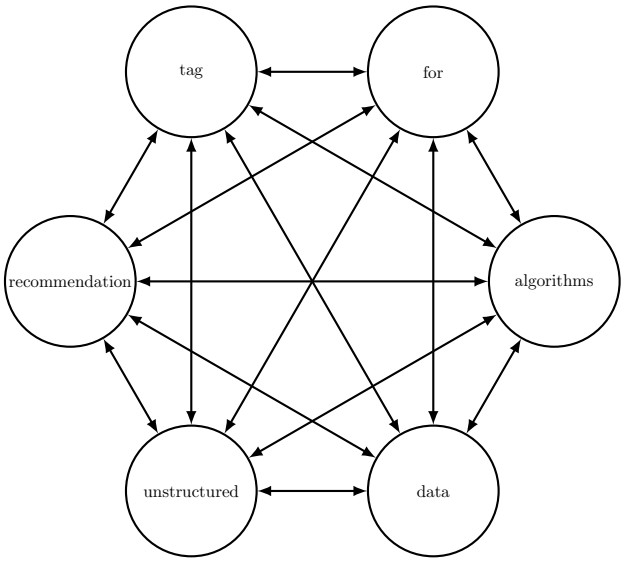


Figure 5.1: Co-occurring words in titles are connected by the title-to-title recommender. Here we see connected words given the sentence 'tag recommendation algorithms for unstructured data'. All edges have a score in both directions. A short sentence, e.g. 'tag recommendation' can be expanded by searching in the network of related title-to-title words.

The graph in Figure 5.2 illustrates the flow of data through the modified network. Each node represents a word. Blue nodes are activated and red nodes indicate recommended words. The TitleToTitle recommender expands the number of nodes forwarded to the TitleToTag and TagToTag recommenders.

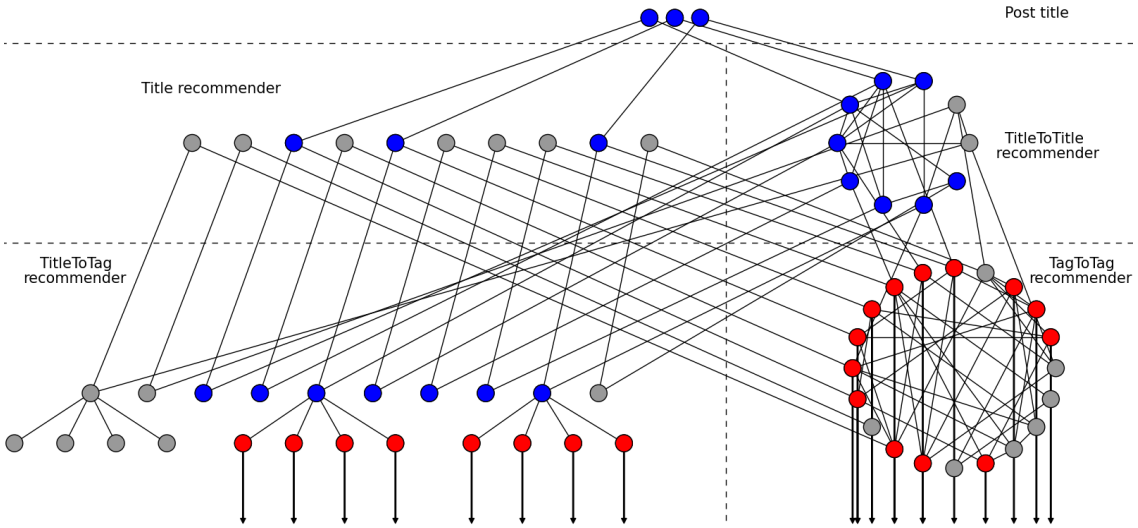


Figure 5.2: Network of words in Lipczak's system. Each node represents a word, blue nodes are selected from the post's title but not recommended and red nodes are recommended.

5.2. Weighted recommendation based on tag popularity

Lipczak's system does not take tag popularity into account but we think it could benefit from incorporating a weighted function based on tag popularity. The system produces various sets of

tags in several stages. We add a popularity weighted function after the TitleRecommender. We first rescore the Title recommender's recommendation by multiplying each score by the normalized tag popularity:

$$s_t^{'} = s_t * \frac{f_t}{f_{\max}} \tag{5.2}$$

Each score s_t is multiplied by the tag frequency of t divided by the maximum tag frequency in the training data $f_{\max}$. However, we figured that the new scores were too biased by the tag frequency multiplier. A smoothing term α was added to reduce the effect of the multiplier:

$$s_t^{'} = s_t * \left(1 + (1 - \alpha) * \frac{f_t}{f_{\max}}\right) \tag{5.3}$$

However, this appeared to have the same effect as normalization discussed earlier in the TF.IDF section (Subsection 4.2.2): tags were rescored but the ordering between them did not change by varying the value of α. We finally implemented the rescoring function as:

$$s_t^{'} = \begin{cases} s_t * \left(1 + \frac{f_t}{f_{max}}\right) & \text{if } f_t \text{ exists} \\ s_t & \text{otherwise} \end{cases} \tag{5.4}$$

Where the value of s_t is multiplied by a value between 1 (infrequent tag) and 2 (frequent tag). The value of s_t remains the same if t is unknown in the set of trained tags.

Chapter 6: Implementation

In this chapter, we discuss how raw data was transformed in order to use it in the experiments (Section 6.1) and discuss how all algorithms were implemented/applied in this research (Section 6.2).

6.1. Data extraction

Each dataset is published in a different format. All datasets were first transformed into a generic format in order to use them in the experiments. Two files were created from each dataset:

1. **posts**
   Each line in this file corresponds to a post in a folksonomy. Data is stored in the format: `(post, user, resource, [tags])`. *Post* is a unique identifier. Some datasets don't supply post ids so the id numbers were automatically generated. *User* corresponds to a unique user id, *resource* to a unique resource id and *tags* is an array of tag-words used to annotate the resource.

2. **resources**
   Each line in this file contains: `(resource, title)`. The resource is a unique id which relates to the resource ids in *posts*, and the title is the title of the resource. The titles are kept seperate from *posts* since resources can be used multiple times and storing the titles in *posts* would result in duplicate data.

During the transformation to generic format, some documents were lost. For example, the DeliciousT140 dataset contained mostly HTML files (99.4%), but also some files in other formats such as PDF, plain text or XML. The extraction of titles from HTML files is simple since it contains title elements (`<title>Document name</title>`). Extraction from other file formats is less straight-forward so those files were dropped from the experiments. Also, some HTML files appeared to be empty and thus no titles could be extracted. Some resources could not be matched with a title so those resources were also removed. The number of remaining posts after extraction for each dataset is given in Table 6.1.

Dataset	Posts	Extracted posts
DeliciousT140	144574	121886
Wiki10+	20762	20762
MovieLens 10M	82103	81167

Table 6.1: Number of remaining posts after data extraction.

A summary of the extraction process per dataset:

1. **DeliciousT140**
   Data consisted of an XML file (188MB) containing users/resources/tags and a folder containing complete HTML documents (6.92GB). Python was used for extracting *posts* from

the XML file and extraction of titles from each individual HTML file for *resources* (using beautifulsoup4 library).

2. **Wiki10+**
   The dataset contained an XML file (38MB) containing users/resources/tags/titles. Complete documents were also supplied (1.73GB). Python was used to parse the XML and store *posts* and *resources*. Complete documents were not used since the XML file contained resource titles.

3. **MovieLens 10M**
   Data was supplied in multiple .dat files, each containing several attributes separated by two colons '::'. Python was used for extracting *posts* and *resources* from the .dat files.

6.2. Code implementation

All coding was done using Python 2.7. Programming was done on two systems: (1) Windows 7 Professional SP1 using an Intel Core i7-2600 CPU @ 3.40GHz and 8GB of RAM and (2) OS X 10.9.5 using an Intel Core i7-2677M CPU @ 1.8GHz and 4GB of RAM. All experiments were run only on the Windows system since it was faster and to assure equal processing speeds on all algorithms.

All algorithms were tested using a self-written framework in Python which selected data, created folds from the data, trained and tested one or more algorithms on the data and averaged the results (as described in Subsection 4.5.1). All algorithms consist of a training stage and a testing stage. We give a summary of the implementation of each algorithm. Pseudocode is given in Appendix A.

The NLTK library was applied for some preprocessing tasks. It was applied to strip HTML, tokenize text, remove stopwords and tag POS categories. Other tasks such as word length filters and non-alphanumeric filters were self implemented. The NLTK tokenizer and stopword remover appeared to miss a lot of 'garbage', mostly punctuation and HTML markup. Therefore additional stopwords were added[1]:

"?", "!", ";", "$", "%", "&", "-", "+", "=", "|", "`", "_", ".", "{", "}", ",", "/", "[", "]", "#", "@", ":", "'", "(", ")", "^", "∼", "*", "http", "gt", "lt", "′t", "la", "′s", "′′", "amp", "n′t", "′m", "...", "de", ""

Algorithm	Implementation method
Global popular tags	Implemented in Python
User popular tags	Implemented in Python
TF.IDF & variants	Implemented in Python
LSI and LDA	Used gensim [22] implementation for training and implemented tag recommendation function
Lipczak & variants	Implemented in Python

Table 6.2: Implementation of algorithms

Each algorithm was implemented as a Python Class with (at least) two functions: `train` and `test`. This ensured every algorithm could be addressed in the same generic way. The `train` function takes a collection of `posts` and the `test` function takes a single `post` and parameter `n_tags`, which is the number of desired tags. During the training stage, the posts contain the tags assigned by a users. During the testing stage, the tags have been removed and tags are recommended based on the remaining post content.

[1] From: `https://github.com/ewencp/twitter-sirikata-vis/blob/master/baseline.py`

Global popular tags

During training, a dictionary of tags was created with their respective frequencies. The tags in the dictionary were sorted by frequency (decreasingly). During testing, the top n tags were returned.

User popular tags

During training, a list of users with their tags sorted by frequency was collected. During testing, the user was selected and his top n most frequent tags were returned. In case the test-post was made by an unknown user or the user's set of tags was too small, tags were added from the global popular tag recommender.

TF.IDF

A TF.IDF implementation is available in gensim, however this appeared to be not flexible enough to support all variants and thus an own implementation was written in Python. In the training stage, all IDF values were precomputed. First all document frequencies of tags were counted. Next, for each tag in the dictionary its IDF score was calculated as $IDF_t = log10\frac{D}{df_t}$.

In the testing stage, for each post a term-frequency (TF) vector was created based on the words in the title. Each term's TF.IDF score was then calculated as $TF_t \times IDF_t$. The resulting tags and scores were sorted by TF.IDF score. If the resulting number of tags was lower than the requested number of tags, the global popular tag recommender was used to add remaining tags.

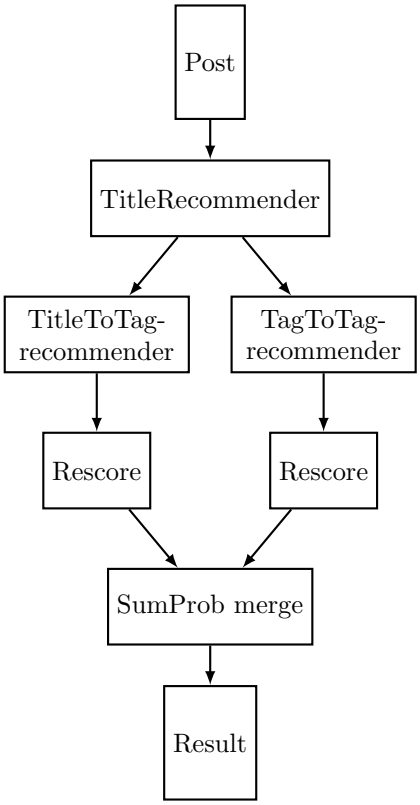
LSI and LDA

We started implementing LDA in Python but quickly discovered that the gensim implementation would be a better option to use since the algorithms were more complex to implement. The gensim model allows to train a model on given documents, but does not have a recommendation option. This was implemented ourself.

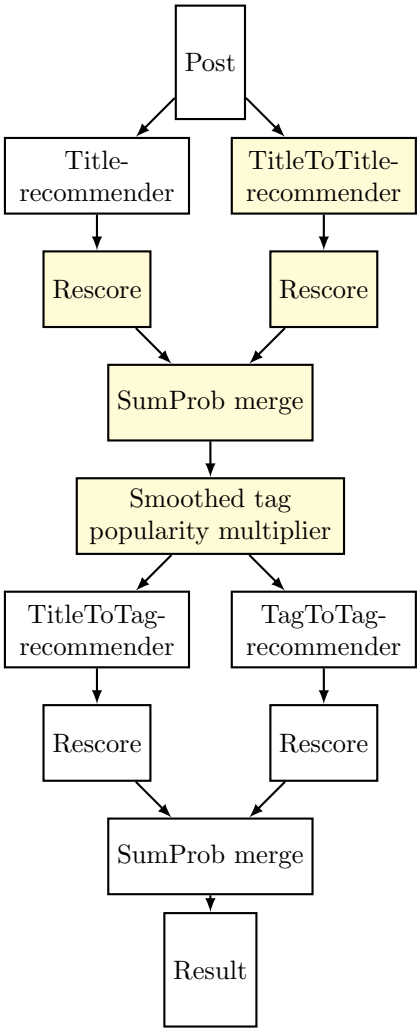
Lipczak

Lipczak's system was described in two papers and source code was available to his website[2]. The source code was undocumented and written in Java. Since we implemented everything in Python, and in order to understand the functioning of the system, we implemented the system based on his papers. Since we focus on content-based tag recommendation in this research we decided to only implement the content-based part of his system. In Figure 6.1 we see two flowcharts: Figure 6.1a shows the part of Lipczak's system that we implemented (the content-based part). Figure 6.1b shows the same system, but with additions as described in Chapter 5.

[2] `https://web.cs.dal.ca/~lipczak/old/tr09/src.php`



(a) Implemented part of Lipczak's system.



(b) Yellow blocks indicate additions to the system.

Figure 6.1: Implemented recommendation system based on Lipczak's papers.

Chapter 7: Experiments

In this chapter, results of experiments in different settings are listed. Experiments are conducted to measure various algorithms on speed and accuracy and to discover which parameters produce optimal results. First, data is preprocessed in order to use it in the experiments. Words are filtered during the preprocessing stage and thus the results are influenced by it.

Extraction of data and transformation to a usable format is described in Section 6.1. Experiments on different sources of words and preprocessing techniques are conducted in respectively Section 7.1 and Section 7.2. Experiments on tag recommendation techniques are carried out in Section 7.3.

Since the number of possible experimental settings is unlimited, a number of experiments were defined to give a comprehensive, yet clear and concise understanding of the performance of algorithms. Most research was done using the DeliciousT140 dataset and thus the results on DeliciousT140 are described comprehensively. Results on other datasets are described in a summarized manner, but complete results are given in the Appendices.

7.1. Tag sources

The DeliciousT140 dataset provides most documents in HTML format. This format allows to select different elements from the document and test them individually to check their performance or 'tag-density'. From each HTML document, the `<title>`, `<h1>` to `<h6>`, `<a>` and remaining content in the `<body>` elements were selected. Basic preprocessing steps were applied to the text from all extracted elements:

1. Conversion to lowercase
2. HTML code stripped
3. Words tokenized
4. Removed words of length 1
5. Removed stopwords

Next, the remaining words were recommended as tags and the Precision, Recall and F1-scores were calculated. This shows which elements are the most tag-dense and provide the best source for possible tags. This process was performed for each document in the DeliciousT140 dataset and results were averaged over all documents. Figure 7.1 shows four plots, respectively the Precision, Recall, F1 and average number of tokens per element in the document (with and without duplicates).

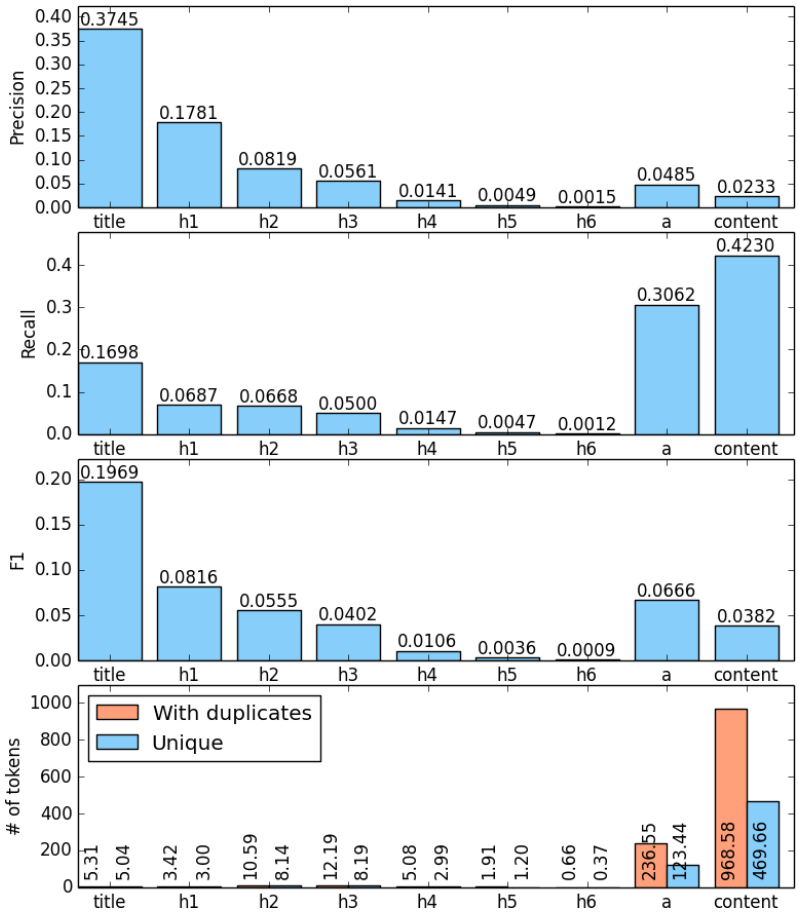


Figure 7.1: Tag density of HTML elements from DeliciousT140 documents after basic preprocessing.

7.2. Preprocessing

Given the results from the previous section, we tested the performance of various preprocessing techniques on the titles of documents. Three necessary preprocessing steps were performed: (1) HTML was stripped, (2) text was converted to lowercase and (3) titles were tokenized. After these steps, each preprocessing step was individually executed and the F1 score was calculated given the remaining tokens.

7.2.1. Word length thresholds

The word length filter is two-fold: it consists of a minimum and maximum word length threshold. A minimum threshold range of $[2, 5]$ combined with a maximum threshold range of $[6, 20]$ was tested. The F1 score for the remaining tags was calculated and results were averaged over all documents. Table 7.1 shows the results for all combinations of the two ranges.

	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	0.1184	0.1399	0.1561	0.1689	0.1755	0.1800	0.1816	0.1824	0.1826	0.1826	0.1826	0.1825	0.1824	0.1824	0.1824
3	0.1204	0.1426	0.1591	0.1722	0.1789	0.1835	0.1851	0.1860	0.1862	0.1861	0.1861	0.1860	0.1859	0.1859	0.1859
4	0.1079	0.1321	0.1499	0.1639	0.1711	0.1761	0.1778	0.1787	0.1789	0.1789	0.1789	0.1788	0.1787	0.1787	0.1787
5	0.0801	0.1075	0.1274	0.1430	0.1510	0.1565	0.1584	0.1594	0.1597	0.1597	0.1597	0.1596	0.1595	0.1595	0.1595

Table 7.1: DeliciousT140 F1-scores after word length filtering.
Rows = minimum word length, columns = maximum word length.

The same experiments were run on other datasets. The complete results are given in Appendix B. Table 7.2 below gives a summary of the optimal thresholds found for each dataset.

Dataset	Optimal minimum length	Optimal maximum length
DeliciousT140	3	14
Wiki10+	2	20
MovieLens 10M	4	12

Table 7.2: Optimal word length thresholds for all datasets.

7.2.2. Part-of-speech filtering

POS tagging and filtering (Section 3.5) was tested by plotting the distribution of POS categories for both title-words and tag-words (Figure 7.2). The bar-plots show the percentage of each POS category[1], averaged over all posts.

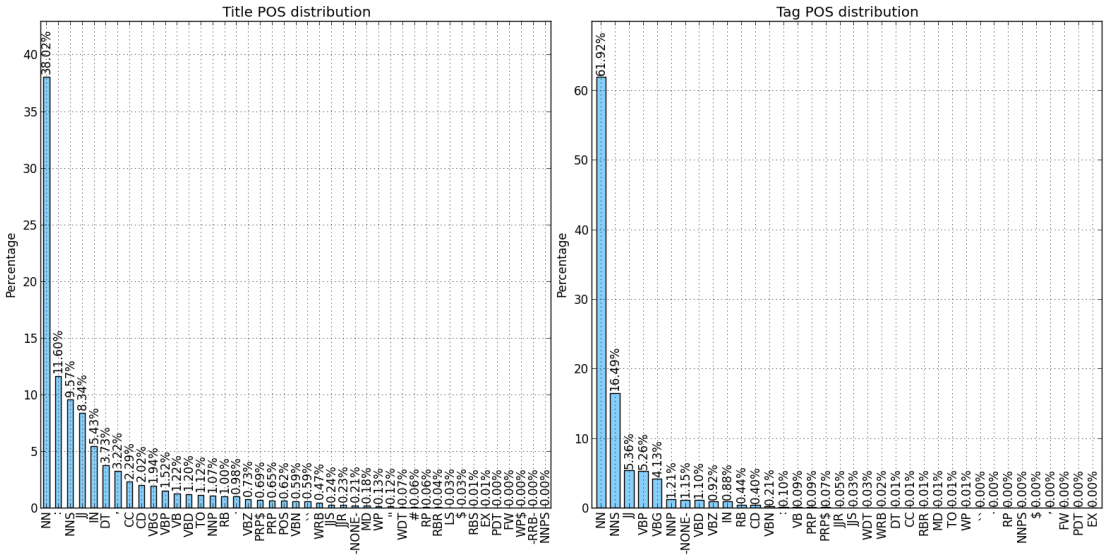


Figure 7.2: Distribution of DeliciousT140 POS categories.
Left: title-word POS distribution, right: tag-word POS distribution.

Given the top 20 POS categories of the tag-words, a combination of POS filters on the title-words was tested by filtering only the top n POS categories. For example: Filter #1: only NN, Filter #2: only NN & NNS, Filter #3: only NN, NNS & JJ, etc. The results for all datasets are given in Table 7.3.

[1]The POS categories displayed in the plots are the POS categories as defined in the Penn Treebank Project: `http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html`

Filter top # of POS	Dataset		
	DeliciousT140	**Wiki10+**	**MovieLens 10M**
1	0.1510	0.0919	0.0068
2	0.1747	0.1022	0.0072
3	0.1849	0.1133	0.0085
4	0.1861	0.1138	0.00896
5	0.1883	0.1156	0.00903
6	0.1892	0.1184	0.0066
7	0.1891	0.1193	0.0067
8	0.1906	0.1238	0.0067
9	0.1911	0.1239	0.0067
10	0.1873	0.1243	0.0067
11	0.1875	0.1264	0.0065
12	0.1869	0.1272	0.0064
13	0.1871	0.1267	0.0064
14	0.1770	0.1273	0.0064
15	0.1779	0.1276	0.0064
16	0.1778	0.1277	0.0064
17	0.1775	0.1279	0.0064
18	0.1775	0.1283	0.0064
19	0.1775	0.12850	0.0064
20	0.1775	0.12854	0.0064

Table 7.3: F1-scores after POS filtering. The top POS categories are determined from the user supplied tags. Best result per dataset is highlighted orange (results for each dataset are independent).

7.2.3. Other preprocessing

Other preprocessing techniques (those that don't require any parameters) are tested in this subsection. The top results from the word length filter and POS filter are included in the results for reference. For each preprocessing technique, the average number of remaining words, the average F1-score and average processing time are listed. The results on DeliciousT140 are given in Table 7.4, the results on other datasets are given in Appendix C.

Technique	Number of tokens	Number of unique tokens	F1	Processing time (seconds)
Raw data	7.86	7.24	0.1689	n.a.
Remove non-alphabetic	6.31	6.01	0.1828	0.000016
Remove non-alphanumeric	6.43	6.13	0.1824	0.000016
Remove stopwords	5.34	5.08	0.1933	0.000253
Word length $\geq$3 filter	5.80	5.53	0.1857	0.000003
Word length $\leq$14 filter	7.77	7.15	0.1691	0.000003
Word length $\geq$3 and $\leq$14 filter	5.71	5.44	0.1862	0.000003
POS top 9 usertag categories filter	4.92	4.70	0.1911	0.005877
Stemming Porter	7.86	7.21	0.1056	0.000104

Table 7.4: DeliciousT140 preprocessing techniques results.

Dataset	Remove non-alphabetic/ non-alphanumeric	Remove stopwords	Word length filter	POS filter	Stemming
DeliciousT140	Remove non-alphabetic	Apply	$\geq$3 and $\leq$14	Top 9 usertag categories	Don't apply
Wiki10+	Remove non-alphanumeric	Apply	$\geq$2	Don't apply	Don't apply
MovieLens 10M	Remove non-alphabetic	Apply	$\geq$4 and $\leq$12	Top 5 usertag categories	Don't apply

Table 7.5: Summary of optimal preprocessing settings for each dataset.

7.3. Tag recommendation algorithms

Several algorithms are implemented and compared:

1. Global popular tags (Subsection 4.1.1)
2. User popular tags (Subsection 4.1.2)
3. TF.IDF & variants (Subsection 4.2.2)
4. LSI (Subsubsection 4.2.3)
5. LDA (Subsubsection 4.2.3)
6. Lipczak & variants (Subsection 4.4.3)

The performance of all algorithms is tested on various datasets (Section 2.2) and using various parameters. All experiments are defined in the next subsection.

7.3.1. Experiment settings

A very large number of experiments can be performed: various algorithms, each with various parameters, on various datasets with a varying number of documents and multiple sources of data. Therefore the number of experiments was limited in order to provide the most important information. All experiments produce a range of 1 - 20 tags and all results are based on three evaluation measures: Precision, Recall and F1.

The following experiments have been carried out. All results displayed in this section are based on DeliciousT140 due to readability. Results on other datasets are given in Appendix D.

#	Setting	Description
1	Popular global tags	Subsection 4.1.1
2	Popular user tags	Subsection 4.1.2
3	TF.IDF	Subsection 4.2.2
4	TF.IWF	Subsection 4.2.2
5	$TF.IWF^2$	Subsection 4.2.2
6	WF.IDF	Logarithmic TF.IDF (Subsection 4.2.2)
7	BTF.IDF	Boolean TF.IDF (Subsection 4.2.2)
8-18	NTF.IDF(0, 0.1, 0.2, ..., 1)	Augmented TF.IDF (Subsection 4.2.2)
19	LSI (5 topics)	Subsection 4.2.3
20-29	LSI (10, 20, 30, ..., 100 topics)	Subsection 4.2.3
30	LDA (5 topics)	Subsection 4.2.3
31-40	LDA (10, 20, 30, ..., 100 topics)	Subsection 4.2.3
41	LDA trained on Wikipedia (50 topics)	LDA trained on English Wikipedia (Subsection 4.2.3)
42-46	LDA trained on Wikipedia (100, 200, ..., 500 topics)	LDA trained on English Wikipedia (Subsection 4.2.3)
47	Content-based Lipczak	Standard Lipczak content-based system (Subsection 4.4.3)
48	TitleToTitle	Section 5.1
49	Tag popularity multiplier	Section 5.2
50	TitleToTitle & Tag popularity multiplier	TitleToTitle and tag popularity multiplier combined

Table 7.6: Experiment settings

Memory usage

Measuring the memory usage of objects types other than built-in types (e.g. int and float) appeared to be not possible. We decided to apply the following approach to get an indication of the required memory. Each recommender was serialized to a string after training of the recommender and written to disk. The size of the file is used as an approximation of the memory usage of the recommender. There is some overhead, e.g. each recommender contains a "name" field which is

also serialized. However, these attributes take very little memory and are thus negligible. The objects were serialized using the Python pickle library[2].

Processing time

The processing time was measured using the Python time library[3]. We measured the training stage and testing stage separately. Due to deviations when measuring very small processing times, we measured the execution time of the full training and testing stage and divided the result by the number of processed posts.

7.3.2. Results

The results consist of three measurements: Precision, Recall and F1. The F1 measure is a combination of Precision and Recall and is the most used measure in the field of information retrieval. The performance of algorithms is ultimately based on the F1 score, so only the F1 results are given in this subsection. The full results including Precision and Recall scores are given in Appendix D. The algorithms shown in the legends are ordered by the summed value of all F1 scores.

[2] `https://docs.python.org/2/library/pickle.html`
[3] `https://docs.python.org/2/library/time.html`

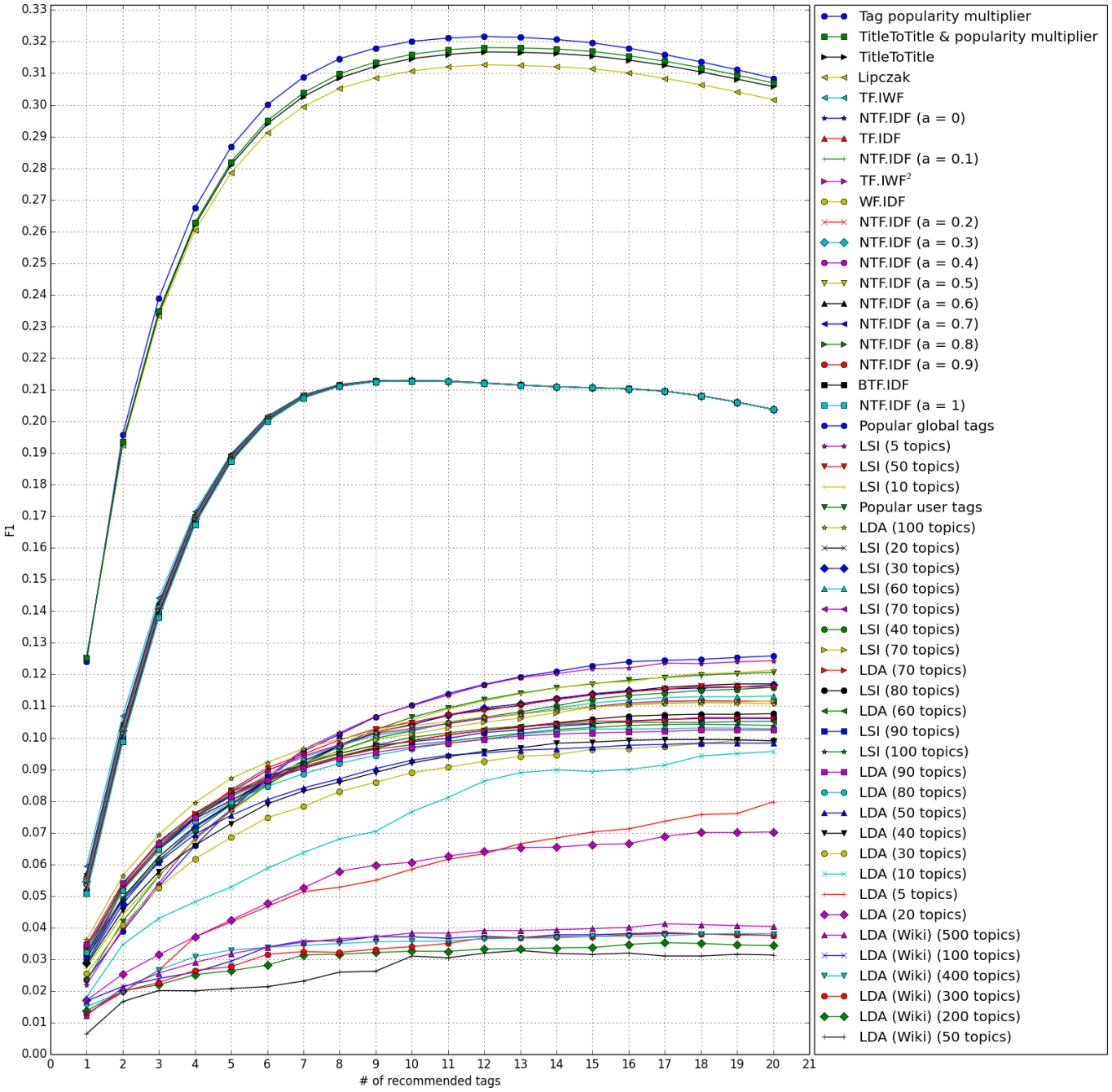


Figure 7.3: DeliciousT140 F1 results for all experiments

First, we show the complete F1 results for all experiments in Figure 7.3. We see three separate sections in the F1 scores: at the top (near 0.32) we see four scores which are Lipczak & contributions. The second section (around 0.21) consists of TF.IDF and variations and we group the other results into a third section.

We first further investigate the TF.IDF results. We see variance between the TF.IDF results, mostly in the range of 1-10 tags (Figure 7.4):

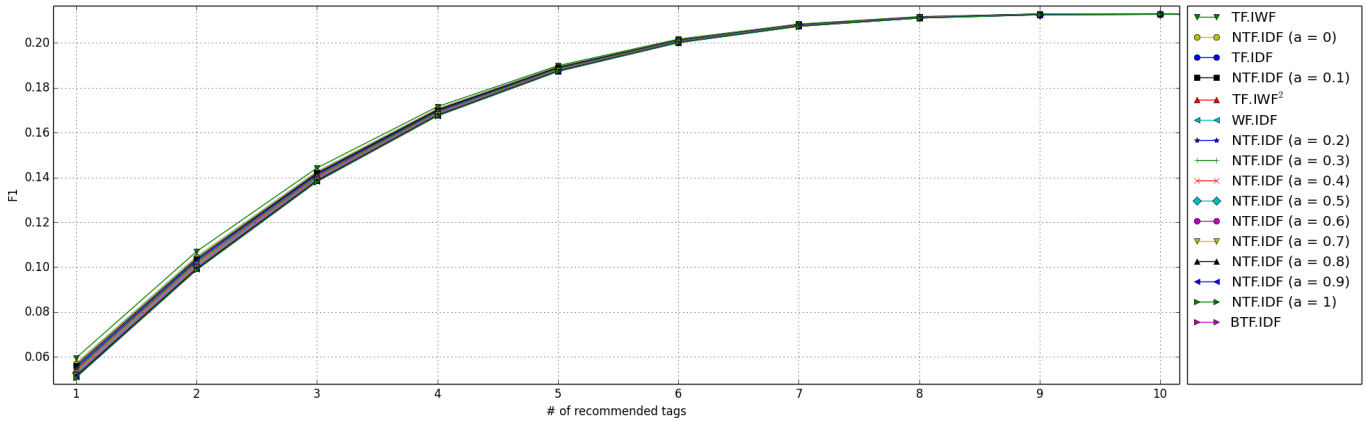


Figure 7.4: DeliciousT140 F1 results for TF.IDF & variants

The variation in results is quite small so we zoom even closer to the F1 scores for recommending 3 tags:

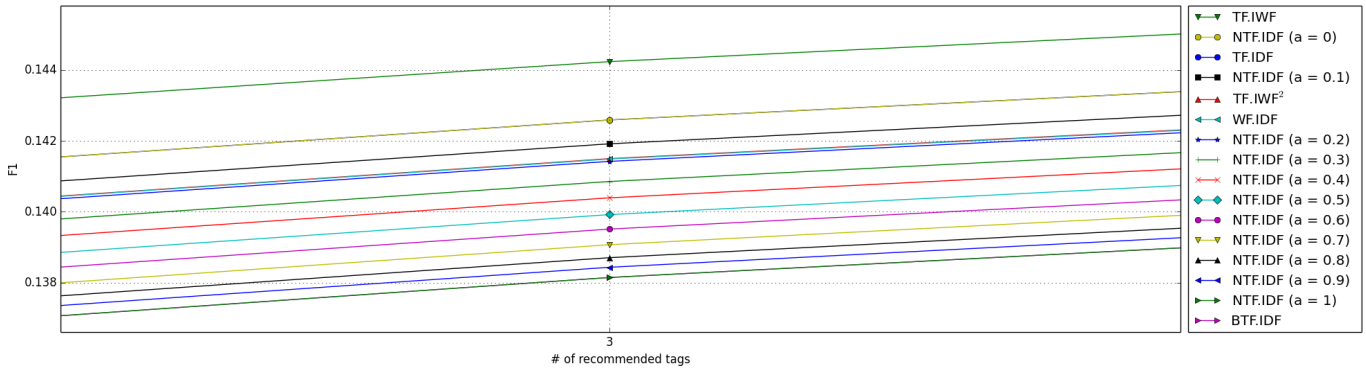


Figure 7.5: DeliciousT140 F1 results for TF.IDF & variants

The TF.IWF scores are on top. Second and third place is taken by NTF.IDF ($\alpha = 0$) and regular TF.IDF. The yellow plot of NTF.IDF overlaps the blue TF.IDF since the results score exactly the same. For $\alpha = 0$ we get: $\alpha + (1 - \alpha)\frac{tf_{t,d}}{tf_{max}(d)} \times IDF_t = \frac{tf_{t,d}}{tf_{max}(d)} \times IDF_t$. As discussed in the TF.IDF chapter, normalisation does not affect the results in the tag recommendation task. Therefore, we can remove the normalisation which results in $tf_{t,d} \times IDF_t$, which equals the regular TF.IDF measure.

Next, we investigate the functioning of Lipczak's system and the novel contributions. The system consists of several stages: training, learning of optimal parameters and testing. We first examine the size of the network of words.

During the experiments, we train on 80% of the data and test on the remaining 20% of the data. In Lipczak's system, we use 80% of the training data to fill the network with words and relations, and the remaining 20% of the training data to learn the optimal merge coefficients. The DeliciousT140 dataset consists of 121886 posts. Each fold contains 97509 training posts of which 78008 are used for filling the network and 19502 posts are used for learning of the merge coefficients. 24380 posts serve as testing data. The individual recommenders produced the following statistics after training:

Title recommender Learned on 404863 titlewords of which 61602 unique, 5.19 average titlewords per post (4.95 unique)

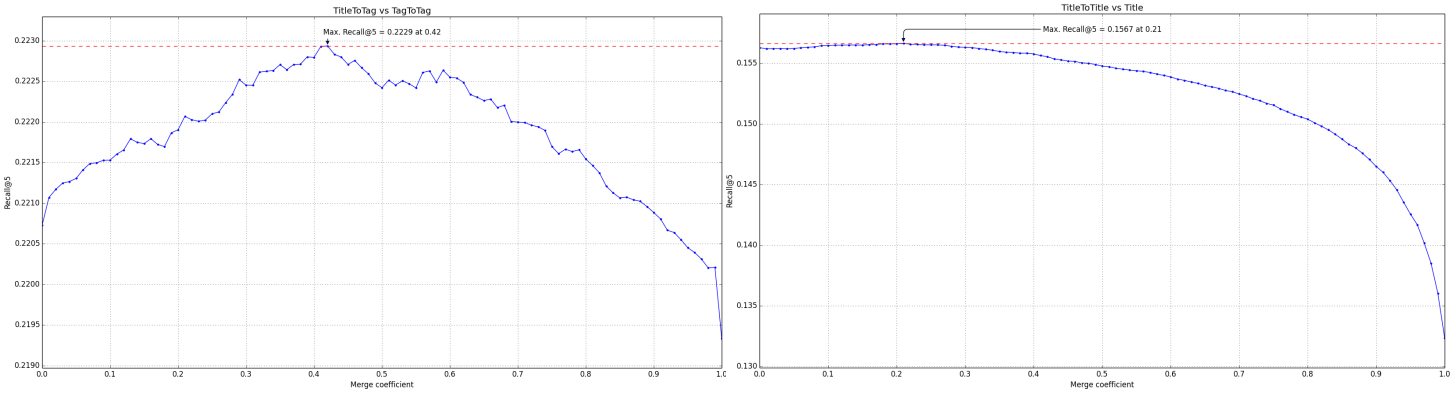
TitleToTag recommender Learned on 61602 titlewords and 1549063 related tagwords = 25.15 related tagwords per titleword

TagToTag recommender Learned on 1081620 tagwords, of which 46708 unique and 4195484 relations = 89.82 related tagwords per tagword

TitleToTitle recommender Learned on 404863 titlewords of which 61602 unique and 1526972 relations = 24.79 related titlewords per titleword

Next, optimal merge coefficients are learned which took 306.20 seconds on average on the original Lipczak system and 411.13 seconds on average with the addition of the TitleToTitle recommender. Both recommenders were serialized. The original Lipczak system took 193.28MB on average on disk and with the additional TitleToTitle recommender it took 251.75MB on average on disk.

The merge coefficients on Lipczak's system were learned by testing the Recall@5 (Recall given top 5 tags) after rescoring two recommenders given merge coefficient values ranging between 0 and 1 with steps of 0.01. His original system learns optimal values between the TitleToTag recommmender and the TagToTag recommender (Figure 7.6a). The addition of the TitleToTitle recommender requires a merge with the Title recommender. We plotted the results of this merger in Figure 7.6b.



(a) TitleToTag (0.42) vs TagToTag (1-0.42=0.58)

(b) TitleToTitle (0.21) vs Title (1-0.21=0.79)

Figure 7.6: Optimal merge coefficients on DeliciousT140 based on Recall@5

The merge coefficient plots shown above are the result of a single fold. The optimal values for all folds are shown below in Table 7.7.

Fold	TitleToTag-TagToTag
1	0.50
2	0.59
3	0.42
4	0.45
5	0.50

(a) Lipczak's content-based merge coefficients

Fold	Title-TitleToTitle	TitleToTag-TagToTag
1	0.21	0.59
2	0.16	0.60
3	0.18	0.59
4	0.17	0.60
5	0.18	0.48

(b) Merge coefficients with TitleToTitle

Fold	TitleToTag-TagToTag (with $\alpha = 1.4$)
1	0.50
2	0.58
3	0.44
4	0.54
5	0.51

(c) Merge coefficients with popularity multiplication

Fold	Title-TitleToTitle	TitleToTag-TagToTag with popularity multiplier
1	0.21	0.51
2	0.16	0.59
3	0.18	0.56
4	0.17	0.60
5	0.18	0.40

(d) Merge coefficients with TitleToTitle and popularity multiplication

Table 7.7: Merge coefficients of different systems on DeliciousT140

Lastly, we zoom in on the results on Lipczak's system for 7 to 17 tags in Figure 7.7. We see a clear peak at 12 tags.

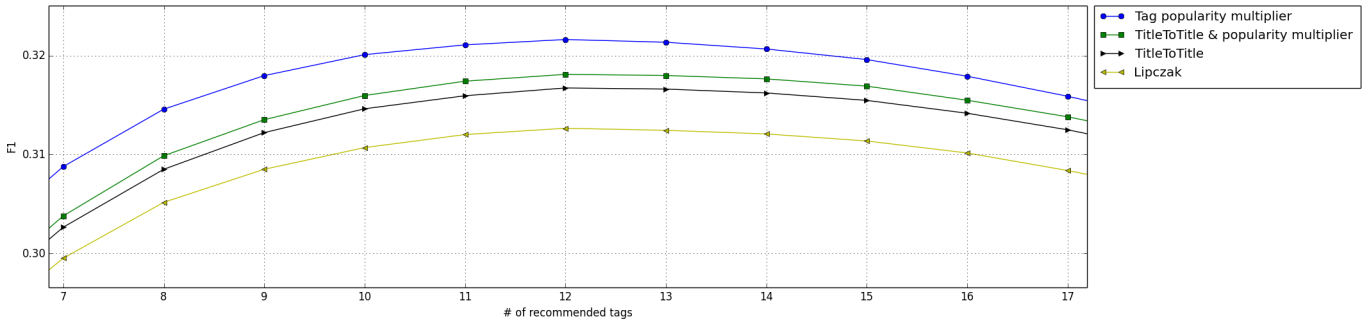


Figure 7.7: DeliciousT140 F1 results for Lipczak, TitleToTitle and popularity multiplier

To see if the number of trained posts influenced the performance of the system, we also trained on a varying number of posts:

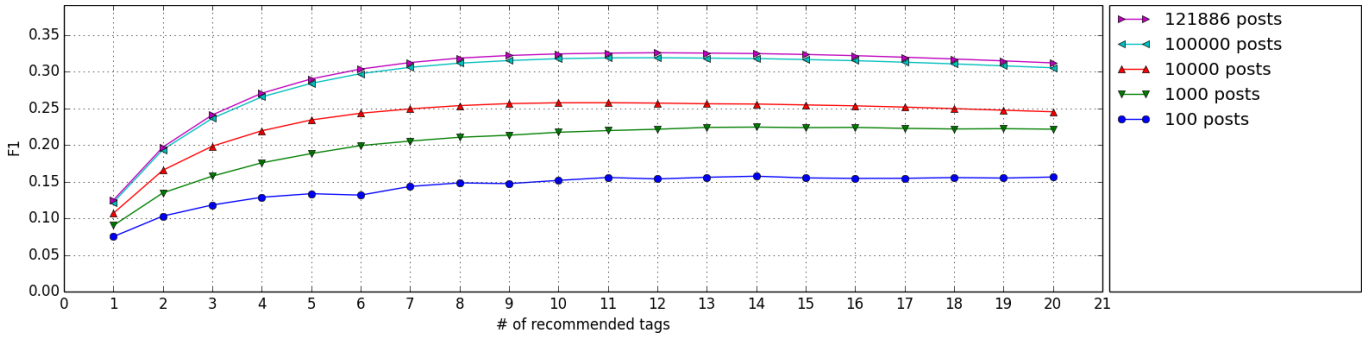


Figure 7.8: DeliciousT140 TitleToTitle with popularity multiplier tested on varying number of posts

The F1 score increases with an increase in the number of posts. However, we see only a small improvement after 100000 posts. Results on other datasets are given in Appendix D.

Chapter 8: Discussion

We started this research with the goal of improving content-based tag recommendation given a collection of posts containing users, resources, tags and resource titles.

Tag sources

We start the experiments by testing various content sources for their tag-density. We tested on all HTML documents in the DeliciousT140 dataset by calculating Precision, Recall and F1 scores after extraction of content from various document elements and basic preprocessing. We see that the body (text within `<body><\body>` tags and words from `<h1>` to `<h6>` and `<a>` removed) contains the most words also seen in the set of user tags (highest Recall score). However, it is also the most imprecise source of tag-words since it contains over 1000 words on average. Words in the `title` prove to be the most accurate source of tags (highest F1 score). The average `title` contains about five words, giving the highest ratio of words also found in the set of user tags. After the `title`, words in respectively `h1`, `a`, `h2` and `h3` elements gave the highest F1 score. In 6th place came content, finally followed by `h4`, `h5` and `h6` elements.

Preprocessing

Next, we compare the performance of various preprocessing methods. We applied all preprocessing techniques on the titles in the DeliciousT140 dataset, since this proved to be the most precise source of tags. The remaining tokens after preprocessing were checked against user tags. The POS tagging took longest to perform, although the processing time is negligible (0.006 seconds). However, we found that the POS tagger did significantly influence the processing time during training of the system. We often trained on 100,000+ documents and found that the POS tagger substantially delayed the system when preprocessing so many documents. During the testing stage, POS tagging was applied on single documents and did not noticeably delay the system. Accuracy-wise, we found that all techniques except stemming had a positive influence on the results. A number of 'garbage' tokens was removed and the resulting F1 scores increased after preprocessing. Stemming also altered many relevant words and thus worsened the results.

Tag recommendation techniques

After preprocessing, we applied a number of recommendation algorithms to three real world datasets. DeliciousT140 and Wiki10+ are post-core level 1 datasets. All resources were tagged by a single user and thus a 'resource profile' containing previously added tags to a resource was of no use. Recommending global popular tags and user popular tags was possible since there were enough posts to train on. When a user profile was not available for a given post, we applied the global popular tag recommender as a backup. The popularity based recommenders scored similar, i.e. the difference in results was small.

TF.IDF & variants

Next, we applied TF.IDF and several variations. We implemented it ourself since the gensim implementation did not offer enough flexibility to apply the variants. We implemented it so that TF.IDF would recommend tags given a resource's title, and extend the set of recommended tags using the set of global popular tags in case not enough tags could be recommended. This was often the case since we tested up to 20 tags and the average title in DeliciousT140 was 7.24 words long. The result of this is noticeable: TF.IDF calculates the significance of each word in a set of words, but does not extend this set of words. So when given the titles, which are about 7.24 words long, TF.IDF will only return that number of tags. The extended tags added from the set of global popular tags will always be the same, regardless of the TF.IDF variant. The effect is shown in Figure 8.1, where we see the results converging. The differences in TF.IDF variants is clearer in Figure 8.2 in which we zoomed to the F1 scores given 3 tags. We see interesting results in the low range of tags: the TF.IWF variant outperforms TF.IDF. Other variants such as WF.IDF, normalized/augmented TF.IDF (NTF.IDF) and boolean TF.IDF (BTF.IDF) perform worse.

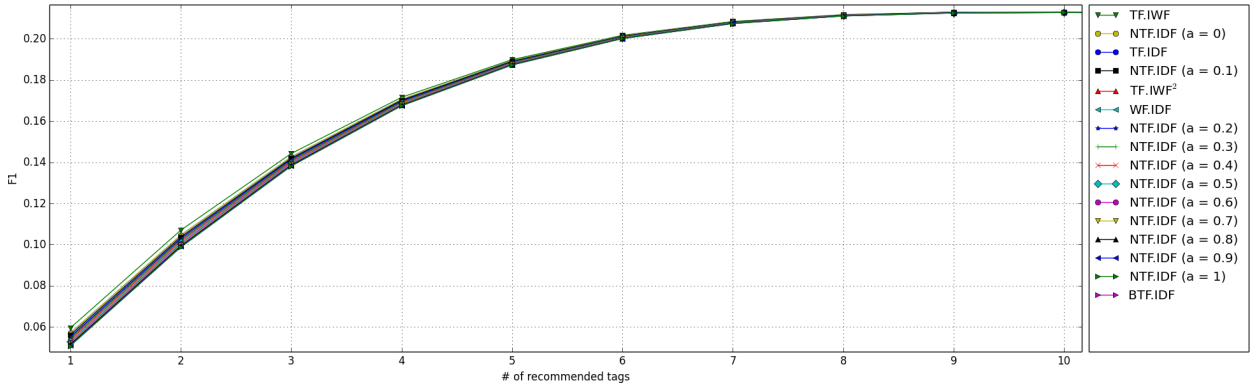


Figure 8.1: DeliciousT140 TF.IDF F1 results zoomed to 1-10 tags. Algorithms in the legend are sorted by summed F1 scores. We see the results converse, which is because of the length of the titles (7.24 words on average)

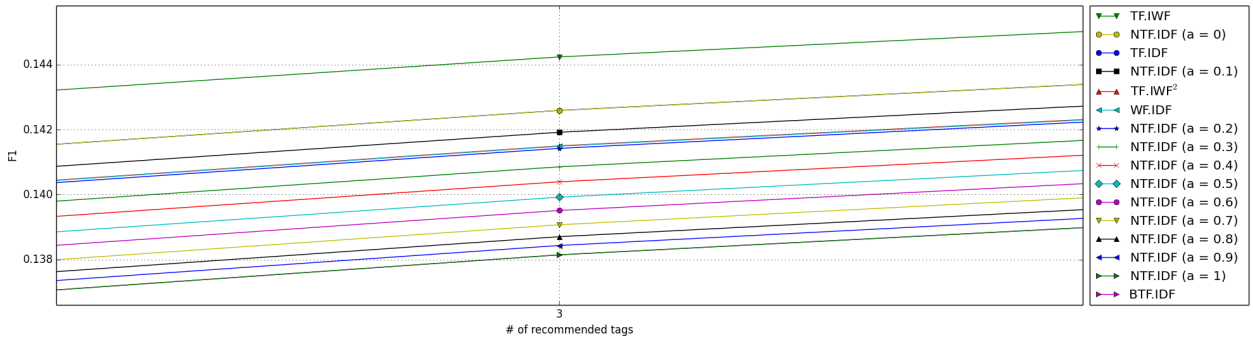


Figure 8.2: DeliciousT140 TF.IDF F1 results zoomed to 3 tags.

Topic modelling techniques

Next, we focussed on topic modelling techniques. We applied the gensim implementation of the LSI and LDA algorithms. The idea is that co-occurring words are clustered into topics. Given a new document, the most related topics are determined and tags can be recommended from those topics. The learning of topic clusters was handled by the gensim library, the tag recommendation was self implemented. We learned the topics from the user tags and tested on resource titles.

However, the clustering seemed inaccurate for the tag recommendation task. The algorithms returned the lowest scores over all experiments over all datasets. We varied the number of topics but did not manage to achieve significantly better results.

The low scores can be explained by the fact that topic models are very similar to popularity based methods in the tag recommendation task (given our implementation). We determine the topic with the highest probability given a post, and select the top n words from that topic as tags. This returns the most frequent words per topic, but those words are often very generic and not able to represent individual documents.

We also trained an LDA model on the English Wikipedia with varying number of topics, but these resulted in the lowest results over all datasets. This is presumably because each dataset is retrieved from a certain domain, containing certain domain-specific or domain-popular words. Also, the tagging process using LSI and LDA models was a time-consuming process (>0.1 second per post = over 3 hours to tag the entire DeliciousT140 dataset). The experiments had to be run overnight, although this could be solved since the last gensim update offers a multicore implementation: `http://radimrehurek.com/gensim/models/ldamulticore.html`.

Lipczak's system

The fact that TF.IDF (or TF.IWF) remained the best-scoring algorithm felt counter-intuitive. We discovered the ECML 2009 challenge in which a winning system was created by Lipczak et al. His system consists of several basic recommenders which are rescored and merged. We implemented the content-based part of his system using Python based on two of his papers. The system learns relations between title-words, tag-words and co-occurring tag-words. The system outperformed TF.IDF significantly.

The testing of merge coefficients was the most time-consuming process. An optimal merge coefficient between the TitleToTag and TagToTag recommender was empirically found by testing 101 values between 0 and 1. When training on a large dataset, the number of related words per word can grow quickly. For example, on DeliciousT140 we learned 46708 unique tagwords with 90 related tagwords on average. Thus, every combination of two recommenders involved merging hundreds of values, sorting these, and selecting only a top number of tags ($\leq$20). This is unnecessary and a limit to the number of processed words could be set.

Novel contributions

We proposed two additions to Lipczak's system. First a TitleToTitle recommender which creates a network of related title-words. This is helpful when given a post with a short title since the post's title can be extended using the TitleToTitle graph. The TitleToTitle recommender was placed parallel with the Title recommender and the scores of the two were merged by testing 101 merge coefficient values between 0 and 1. On the DeliciousT140 dataset, we found optimal merge coefficients around 0.20, where the TitleToTitle recommender is given a weight of 0.20 and the Title recommender a weight of 1-0.20=0.80. This indicates that the Title recommender remains the primary source of tags, but the TitleToTitle recommender is a good addition. This is confirmed given the results: the F1 scores improved on all three datasets with the addition of the TitleToTitle recommender:

Dataset	Lipczak	TitleToTitle
DeliciousT140	0.3127 (12 tags)	0.3167 (12 tags)
Wiki10+	0.2189 (13 tags)	0.2193 (13 tags)
MovieLens 10M	0.0370 (6 tags)	0.0909 (3 tags)

Table 8.1: Best F1 scores original Lipczak versus Lipczak with TitleToTitle

Second, we propose a popularity-based rescoring function. Suggested tags are rescored based on their popularity in the dataset. The results are varying. We measured an improvement on the DeliciousT140 dataset, no change on the Wiki10+ dataset and a slight decrease on the MovieLens 10M dataset.

The DeliciousT140 and Wiki10+ datasets are typical social tagging datasets: they consist of documents which were annotated with tags by users. The MovieLens 10M dataset differs since it is focussed on movie ratings and non-descriptive tags seem to be added. I.e., the tags were not added with the goal of findability. Also, the tags are not supplied as an array of tag-words, but as one sentence, e.g.: "Hunting Party, The (2007)". The tag-sentences were split at whitespaces. Most tags only contained a movie year and name. All these facts clearly translated into poor results: the best algorithm was the popular user tag recommender scoring 0.1736 at 2 tags. Second came Lipczak's system with the additional TitleToTitle recommender. On Wiki10+, the range of results was wider. LSI and LDA models performed poor, next came popularity based methods, followed by TF.IDF and variants. Lipczak's system outperformed TF.IDF and our TitleToTitle extension combined with the popularity rescorer gave the best F1 score of 0.2193 at 13 tags. On DeliciousT140, LSI and LDA also performed poorest, followed by popularity based methods and again TF.IDF and variants. Lipczak's system outperforms TF.IDF and our additions again outperform the original Lipczak system. The best F1 score was 0.3216 for 12 tags. An optimal number of tags differs per dataset. MovieLens 10M contained 1.16 tags per post, DeliciousT140 13.92 tags and Wiki10+ 22.05 tags per post. We see this in our results: we see peaks in the results at or near those numbers.

Chapter 9: Future work

We investigated various tag recommendation techniques on various datasets. During this research, we found that Lipczak's system performed the best. Our additional popularity rescorer showed varying results and the TitleToTitle recommender outperformed the original system on all three datasets. We are pleased with the results but see several possibilities for future work.

One issue we faced during the research was the performance. Lipczak's model grows fast in memory. Although the TitleToTitle recommender improves results, it also consumes additional memory. A more efficient implementation of the system could greatly improve the usability of the system, e.g. by setting a score threshold before processing a word.

In other research, Lipczak applied post date in order to build a user profile recommender. Not all datasets provide a post date and thus we decided not to apply this method. However, additional data can easily be fitted into the system. For example, we can incorporate movie ratings between 1 and 5 from MovieLens 10M in the recommendation task. Or, we could add weights to tags based on the number of views of a resource in a system. This however depends on the dataset. Lipczak's method of learning optimal merge coefficients for Recall@5 proves to work accurate and is thus very helpful in determining an optimal ratio between multiple recommenders.

For a generic improvement, we could imagine handling multiword concepts, personalisation and selecting other sources than the title from the content. We did a thorough analysis of the tag-density of various document sources in DeliciousT140 and found that `h1` tags were the second best source of tags. This could provide additional information in the tag recommendation task.

Chapter 10: Bibliography

[1] Brian T Bartell, Garrison W Cottrell, and Richard K Belew. Automatic combination of multiple ranked retrieval systems. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 173–181. Springer-Verlag New York, Inc., 1994.

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

[3] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[4] Keli Chen and Chengqing Zong. A new weighting algorithm for linear classifier. In *Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003 International Conference on*, pages 650–655. IEEE, 2003.

[5] Ernesto Diaz-Aviles, Mihai Georgescu, Avaré Stewart, and Wolfgang Nejdl. Lda for on-the-fly auto tagging. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 309–312. ACM, 2010.

[6] George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[7] Edward A Fox and Joseph A Shaw. Combination of multiple searches. *NIST Special Publications SP*, pages 243–243, 1994.

[8] 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.

[9] Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 693–702. ACM, 2010.

[10] Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *Knowledge Discovery in Databases: PKDD 2007*, pages 506–514. Springer, 2007.

[11] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 61–68. ACM, 2009.

[12] Nikolas Landia, Stephan Doerfel, Robert Jäschke, Sarabjot Singh Anand, Andreas Hotho, and Nathan Griffiths. Deeper into the folksonomy graph: Folkrank adaptations and extensions for improved tag recommendations. *arXiv preprint arXiv:1310.1498*, 2013.

[13] Joon Ho Lee. Analyses of multiple evidence combination. In *ACM SIGIR Forum*, volume 31, pages 267–276. ACM, 1997.

[14] Marek Lipczak, Yeming Hu, Yael Kollet, and Evangelos Milios. Tag sources for recommendation in collaborative tagging systems. *ECML PKDD discovery challenge*, 497:157–172, 2009.

[15] Marek Lipczak and Evangelos Milios. The impact of resource title on tags in collaborative tagging systems. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, pages 179–188. ACM, 2010.

[16] Marek Lipczak and Evangelos Milios. Learning in efficient tag recommendation. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 167–174. ACM, 2010.

[17] Richard Maclin and David Opitz. Popular ensemble methods: An empirical study. *arXiv preprint arXiv:1106.0257*, 2011.

[18] Leandro Balby Marinho and Lars Schmidt-Thieme. Collaborative tag recommendations. In *Data Analysis, Machine Learning and Applications*, pages 533–540. Springer, 2008.

[19] Olfa Nasraoui. Web data mining: Exploring hyperlinks, contents, and usage data. *ACM SIGKDD Explorations Newsletter*, 10(2):23–25, 2008.

[20] Christos H Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168. ACM, 1998.

[21] Maria T Pazienza. *Information Extraction*. Springer, 1999.

[22] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[23] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60:2004, 2004.

[24] G. Salton. *The SMART retrieval system: experiments in automatic document processing*. Prentice-Hall series in automatic computation. Prentice-Hall, 1971.

[25] Deepika Sharma. Stemming algorithms: A comparative study and their analysis. *International Journal of Applied Information Systems*, 4(3):7–12, 2012.

[26] Xiance Si, Zhiyuan Liu, Peng Li, Qixia Jiang, and Maosong Sun. Content-based and graph-based tag suggestion. *Proceedings of the ECML PKDD Discovery Challenge*, pages 243–260, 2009.

[27] Xiance Si and Maosong Sun. Tag-lda for scalable real-time tag recommendation. *Journal of Computational Information Systems*, 6(1):23–31, 2009.

[28] Marta Tatu, Munirathnam Srikanth, and Thomas DSilva. Rsdc08: Tag recommendations using bookmark content. *ECML PKDD discovery challenge*, 2008:96–107, 2008.

[29] Karen HL Tso-Sutter, Leandro Balby Marinho, and Lars Schmidt-Thieme. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999. ACM, 2008.

[30] Lei Wu, Linjun Yang, Nenghai Yu, and Xian-Sheng Hua. Learning to tag. In *Proceedings of the 18th international conference on World wide web*, pages 361–370. ACM, 2009.

[31] Zhixiang Eddie Xu, Minmin Chen, Kilian Q Weinberger, and Fei Sha. From sbow to dcot marginalized encoders for text representation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1879–1884. ACM, 2012.

[32] Arkaitz Zubiaga. Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469*, 2012.

[33] Arkaitz Zubiaga, Alberto Pérez Garcia-Plaza, Víctor Fresno, and Raquel Martínez. Content-based clustering for tag cloud visualization. In *Social Network Analysis and Mining, 2009. ASONAM'09. International Conference on Advances in*, pages 316–319. IEEE, 2009.

Appendix A: Algorithm pseudocode

The pseudocode is written in a slightly Pythonic way. A `dict` is a container consisting of (key,value) pairs. `dict[key]` will return a key's value from the dict. `dict.get(key,0)` will also return a key's value, but return 0 if the key doesn't exist.

Algorithm 1 Global popular tag recommender

```
1:  function TRAIN(posts)
2:      global_tags = dict()
3:      for each p in posts do
4:          for each word in p.tags do
5:              global_tags[word] = global_tags.get(word,0) + 1      ▷ Add +1 for each tag occurence
6:          end for
7:      end for
8:      global_tags = sorted(global_tags)
9:  end function
10:
11: function TEST(post, n_tags)
12:     return global_tags[:n_tags]                                  ▷ Return top n_tags global tags
13: end function
```

Algorithm 2 User popular tag recommender

```
1:  function TRAIN(posts)
2:      user_tags = dict()
3:      global_tags = dict()
4:      for each p in posts do
5:          user_dict = user_tags.get(p.user,dict())                 ▷ Get user specific tags
6:          for each word in p.tags do
7:              user_dict[word] = user_dict.get(word,0) + 1          ▷ Count user tag
8:              global_tags[word] = global_tags.get(word,0) + 1      ▷ Count global tag
9:          end for
10:         user_tags[p.user] = sorted(user_dict)
11:     end for
12:     global_tags = sorted(global_tags)
13: end function
14:
15: function TEST(post, n_tags)
16:     result = user_tags[p.user][:n_tags]
17:     if length(result) < n_tags then
18:         result += global_tags[:n_tags - length(result)]          ▷ Add global popular tags
19:     end if
20:     return result
21: end function
```

Algorithm 3 TF.IDF recommender

```
1: function TRAIN(posts)
2:     for each p in posts do
3:         for each word in set(p.tags) do
4:             term_doccount[word] = term_doccount.get(word,0) + 1        ▷ Count term-document
   frequency
5:         end for
6:     end for
7:     for each term, doccount in term_doccount do
8:         idf[term] = log10(length(posts)/doccount)                      ▷ Compute IDF for each term
9:     end for
10: end function
11:
12: function TEST(post, n_tags)
13:     for each word in p.title do
14:         tf[word] = tf.get(word,0) + 1                                  ▷ Count terms in post (=TF)
15:     end for
16:     for each word, tf_score in tf do
17:         tfidf[word] = tf_score * idf[word]                             ▷ Calculate TF*IDF for each term
18:     end for
19:     return tfidf[:n_tags]
20: end function
```

Algorithm 4 LSI and LDA recommender

```
1: function TRAIN(posts, k)
2:     if LSI then
3:         model = gensim.LSI(posts, k)                                   ▷ Train LSI
4:     else if LDA then
5:         model = gensim.LDA(posts, k)                                   ▷ Train LDA
6:     end if
7: end function
8:
9: function TEST(post, n_tags, strategy)
10:     topics = InferTopicDistribution(post.title)
11:     if strategy == 1 then                                              ▷ Top 1 from all topics
12:         return t[0] for t in topics[:n_tags]
13:     else if strategy == 2 then                                         ▷ Top n_tags from top topic
14:         return topics[0][:n_tags]
15:     end if
16: end function
```

Algorithm 5 Lipczak recommender

```
function TRAIN(posts)
    trainPosts = posts[:len(posts)*0.8]                    ▷ Use 80% of posts for training
    learnPosts = posts[len(posts)*0.2:]        ▷ Use 20% of posts for learning merge coefficients
    titleRecommender = TitleRecommender.train(trainPosts)
    titleToTitleRecommender = titleToTitleRecommender.train(trainPosts)
    titleToTagRecommender = titleToTagRecommender.train(trainPosts)
    learnMergeCoefficients(learnPosts)
end function

function TEST(post, n_tags)
    ▷ Get recommendations
    titleRec = TitleRecommender.test(post)
    titleToTagRec = TitleToTagRecommender.test(titleRec)
    tagToTagRec = TagToTagRecommender.test(titleRec)
    ▷ Rescore recommendations
    titleToTagRec = rescore(titleToTagRec, titleToTag_mergeCoefficient)
    tagToTagRec = rescore(tagToTagRec, tagToTag_mergeCoefficient)
    ▷ Merge recommendations
    contentRec = SumProbMerge([titleToTagRec, tagToTagRec])
    return contentRec[:n_tags]
end function
```

Algorithm 6 Title recommender

```
function TRAIN(posts, low_freq, low_freq_score, low_score_threshold)
    titleWords = dict()
    titleTagWords = dict()
    for each p in posts do
        for each word in p.title do
            titleWords[word] = titleWords.get(word,0)+1                    ▷ Count titlewords
            if word in p.tags then
                titleTagWords[word] = titleTagWords.get(word,0)+1      ▷ Count title&tag-words
            end if
        end for
    end for
    titleWordScores = dict()
    for each titleword,titleword_freq in titleWords do
        if titleword_freq < low_freq then
            ▷ Set default score for low frequency words
            titleWordScores[titleword] = low_freq_score
        else
            score = titleTagWords[titleword]/titleword_freq
            if score ≥ low_score_threshold then                ▷ Drop words with low score
                titleWordScores[titleword] = score
            end if
        end if
    end for
end function

function TEST(post)
    result = titleWordScore[w] for w in post.title
    ▷ result = [(word1,score1),(word2,score2)....]
    return sorted(result)
end function
```

Algorithm 7 TitleToTag recommender

```
1: function TRAIN(posts)
2:     titleWordCount = dict()
3:     titleTagGraph = dict()
4:     for each for p in posts do
5:         for each titleword in p.title do
6:             titleWordCount[titleword] = titleWordCount.get(titleword,0)+1
7:             titleTagGraph[titleword] += p.tags                ▷ Add tags to related tags list
8:         end for
9:     end for
10:    titleTagScores = dict()
11:    for each titleword,titletags in titleTagGraph do
12:        titletag_scores = dict()
13:        for each tag in titletags do
14:            titletag_scores[tag] = titletag_scores.get(tag,0)+1          ▷ Count tag frequency
15:        end for
16:        for each tag,tag_freq in titletag_scores do
17:            ▷ Calculate title to tag scores
18:            titletag_scores[tag] = tag_freq / titleWordCount[titleword]
19:        end for
20:    end for
21: end function
22:
23: function TEST(wordscores)
24:    ▷ wordscores = [(word1,score1),(word2,score2)....]  from TitleRecommender
25:    for each word,score in wordscore do
26:        relatedTags = titleTagScores[word]
27:        result += relatedTags * score
28:    end for
29:    ▷ result = [[(word1,score1),(word_n,score_n)][(word1,score1),(word_n,score_n)],....]
30:    ▷ (one array of related tagwords/scores for each word from TitleRecommender)
31:    result = SumProbMerge(result)
32:    return result
33: end function
```

Algorithm 8 SumProb merge

```
1: function SUMPROBMERGE(recommendations)
2:     ▷ recommendations = [[(word1,score1),(word2,score2)][(word1,score1)(word2,score2)]....]
3:     products = dict()
4:     for each rec in recommendations do
5:         for each word,score in rec do
6:             products[word] = products.get(word,1) * (1-score)
7:         end for
8:     end for
9:     for each word,score in products do
10:        products[word] = 1-score
11:    end for
12:    ▷ products = [(word1,score1),(word2,score2), ...,(word_n,score_n)]
13:    return products
14: end function
```

Algorithm 9 TagToTag recommender

```
function TRAIN(posts)
    tagFrequencies = dict()
    tagGraph = dict()
    for each p in posts do
        for each tag in p.tags do
            tagFrequencies[tag] = tagFrequencies.get(tag,0)+1        ▷ Store tag frequency
            tagGraph[tag] += p.tags                                  ▷ Store co-occuring tags
        end for
    end for
    for each t1, t1related in tagGraph do                            ▷ Calculate tag relation scores
        t1related_frequencies = count(t1related)
        for each t2, t2frequency in t1related_frequencies do
            t1related_scores[t2] = t2frequency / tagFrequencies[t1]
        end for
        tagGraph[t1] = t1related_scores
    end for
end function

function TEST(wordscores)
    ▷ wordscores = [(word1,score1),(word2,score2)....]  from TitleRecommender
    for each word,score in wordscore do
        relatedTags = tagGraph[word]
        result += relatedTags * score
    end for
    ▷ result = [[(word1,score1),(word_n,score_n)][(word1,score1),(word_n,score_n)],....]
    ▷ (one array of related tagwords/scores for each word from TitleRecommender)
    result = SumProbMerge(result)
    return result
end function
```

Algorithm 10 Learn merge coefficients

```
function LEARNMERGECOEFFICIENTS(posts)
    mergeRange = [0, 0.01, 0.02, ..., 1]                             ▷ Merge values to test
    for each p in posts do
        titleRec = TitleRecommender.test(p)                          ▷ Get title recommendation
        titleToTagRec = titleToTagRecommender.test(titleRec)         ▷ Title-to-tag recommendation
        tagToTagRec = tagToTagRecommender.test(titleRec)             ▷ Tag-to-tag recommendation
        for each m in mergeRange do
            titleToTagRec = rescore(titleToTagRec, m)                ▷ Rescore
            tagToTagRec = rescore(tagToTagRec, 1-m)                  ▷ Rescore
            result = SumProbMerge(titleToTagRec, tagToTagRec)        ▷ Merge
            recall = Recall@5(result)                                ▷ Check result
        end for
    end for
    setMergeCoefficients(max(recall))
end function
```

Algorithm 11 Rescoring function

```
function RESCORE(recommendation, mergeCoefficient)
    ▷ recommendation = [(word1,score1),(word2,score2), ...]
    ▷ mergeCoefficient = value between 0 and 1
    scoreFactor = mergeCoefficient/recommendation[0]                 ▷ Divide by first score
    rescoredRecommendation = r*scoreFactor for r in recommendation
    return rescoredRecommendation
end function
```

Algorithm 12 TitleToTitle recommender

```
1: function TRAIN(posts)
2:     titleWordCount = dict()                                  ▷ For counting title-words
3:     titleGraph = dict()                                      ▷ For storing related title-words
4:     for each p in posts do
5:         for each word in p.title do
6:             titleWordCount[word] = titleWordCount.get(word,0)+1          ▷ Count word
7:             titleGraph[word] = count([word for word in p.title])         ▷ Count related words
8:         end for
9:     end for
10:    for each titleword, related_words in titleGraph do       ▷ Calculate tag relation scores
11:        for each related_titleword,count in related_words do
12:            scores[titleword → related_titleword] = count/titleWordCount[titleword]
13:        end for
14:    end for
15: end function
16:
17: function TEST(post)
18:    for each word in post.title do
19:        for each related_word,related_score in scores[word] do
20:            result += (related_word,related_score)           ▷ Add related words & scores to result
21:        end for
22:    end for
23:    ▷ result = [[(w1,s1),(w2,s2)],[(w1,s1),(w2,s2)],...]
24:    ▷ (one array of related titlewords/scores for each word in title)
25:    result = SumProbMerge(result)
26:    return result
27: end function
```

Algorithm 13 Smoothed tag frequency multiplier

```
1: function RESCOREBYPOPULARITY(recommendation,alpha)
2:     ▷ Assume variable ''frequencies'' contains tag frequencies
3:     for each word,score in recommendation do
4:         score = score*(1+(1-alpha))*(frequencies[word]/max(frequencies))
5:     end for
6:     return recommendation
7: end function
```

Appendix B: Word length filter results

The tables in this chapter show the F1 scores after recommending title-words which have been filtered by word length as described in the experiments in Subsection 7.2.1. Each table contains the results for one dataset.

All results are displayed using four decimals. In case multiple settings scored best due to the rounding, the number of decimals was increased until a single best score was found.

	6	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
2	0.1184	0.1399	0.1561	0.1689	0.1755	0.1800	0.1816	0.1824	0.1826	0.1826	0.1826	0.1825	0.1824	0.1824	0.1824
3	0.1204	0.1426	0.1591	0.1722	0.1789	0.1835	0.1851	0.1860	0.1862	0.1861	0.1861	0.1860	0.1859	0.1859	0.1859
4	0.1079	0.1321	0.1499	0.1639	0.1711	0.1761	0.1778	0.1787	0.1789	0.1789	0.1789	0.1788	0.1787	0.1787	0.1787
5	0.0801	0.1075	0.1274	0.1430	0.1510	0.1565	0.1584	0.1594	0.1597	0.1597	0.1597	0.1596	0.1595	0.1595	0.1595

Table B.1: DeliciousT140 F1-scores after word length filtering.
Rows = minimum word length, columns = maximum word length.

	6	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
2	0.0636	0.0820	0.0981	0.1091	0.1169	0.1225	0.1253	0.1269	0.1279	0.1284	0.1286	0.1288	0.12885	0.12888	0.12890
3	0.0615	0.0801	0.0963	0.1074	0.1153	0.1209	0.1237	0.1254	0.1263	0.1268	0.1271	0.1272	0.1273	0.1273	0.1274
4	0.0547	0.0735	0.0898	0.1010	0.1090	0.1145	0.1174	0.1191	0.1200	0.1205	0.1208	0.1209	0.1210	0.1210	0.1211
5	0.0387	0.0577	0.0743	0.0857	0.0938	0.0994	0.1023	0.1040	0.1050	0.1055	0.1057	0.1059	0.1060	0.1060	0.1060

Table B.2: Wiki10+ F1-scores after word length filtering.
Rows = minimum word length, columns = maximum word length.

	6	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
2	0.0060	0.0066	0.0070	0.0074	0.0074	0.0074	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075
3	0.0063	0.0069	0.0073	0.0077	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078
4	0.0064	0.0072	0.0076	0.0080	0.00808	0.00812	0.00815	0.00814	0.00814	0.00814	0.00814	0.00814	0.00814	0.00814	0.00814
5	0.0053	0.0065	0.0071	0.0076	0.0077	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078	0.0078

Table B.3: MovieLens 10M F1-scores after word length filtering.
Rows = minimum word length, columns = maximum word length.

Appendix C: Preprocessing results

The performance of various preprocessing techniques is tested on all datasets. Only titles were used in these tests. The titles were preprocessed and recommended as tags. The resulting F1 scores are given in the tables below.

All titles are initially stripped from HTML, tokenized and converted to lowercase. After that, each preprocessing technique is applied and measured. Grey cells show the initial F1 score (only initial preprocessing), green cells show an increase in F1 score and red cells denote a decrease in F1 score.

Technique	Number of tokens	Number of unique tokens	F1	Processing time (seconds)
Raw data	7.86	7.24	0.1689	n.a.
Remove non-alphabetic	6.31	6.01	0.1828	0.000016
Remove non-alphanumeric	6.43	6.13	0.1824	0.000016
Remove stopwords	5.34	5.08	0.1933	0.000253
Word length ≥3 filter	5.80	5.53	0.1857	0.000003
Word length ≤14 filter	7.77	7.15	0.1691	0.000003
Word length ≥3 and ≤14 filter	5.71	5.44	0.1862	0.000003
POS top 9 usertag categories filter	4.92	4.70	0.1911	0.005877
Stemming Porter	7.86	7.21	0.1056	0.000104

Table C.1: DeliciousT140 preprocessing techniques results.

Technique	Number of tokens	Number of unique tokens	F1	Processing time (seconds)
Raw data	2.29	2.28	0.12834	n.a.
Remove non-alphabetic	2.17	2.16	0.1286	0.000007
Remove non-alphanumeric	2.18	2.18	0.1292	0.000006
Remove stopwords	2.00	2.00	0.12831	0.000216
Word length ≥2 filter	2.17	2.17	0.1289	0.000002
Word length ≤20 filter	2.29	2.28	0.12831	0.000002
Word length ≥2 and ≤20 filter	2.17	2.16	0.1289	0.000002
POS top 20 usertag categories filter	2.24	2.23	0.1285	0.001654
Stemming Porter	2.29	2.28	0.0770	0.000042

Table C.2: Wiki10+ preprocessing techniques results.

Technique	Number of tokens	Number of unique tokens	F1	Processing time (seconds)
Raw data	6.99	6.57	0.00517	n.a.
Remove non-alphabetic	3.28	3.15	0.0081	0.000015
Remove non-alphanumeric	4.33	4.20	0.0072	0.000015
Remove stopwords	3.45	3.39	0.0083	0.000426
Word length ≥4 filter	3.16	3.11	0.0080	0.000003
Word length ≤12 filter	6.98	6.55	0.00518	0.000003
Word length ≥4 and ≤12 filter	3.14	3.10	0.0080	0.000003
POS top 5 usertag categories filter	2.19	2.14	0.0090	0.005579
Stemming Porter	6.99	6.56	0.0035	0.000099

Table C.3: MovieLens 10M preprocessing techniques results.

Appendix D: Experiment results

The results are organized per dataset. Each section contains the results of one dataset and consists of (1) processing time and memory usage results, (2) a plot of the Precision/Recall/F1 results and (3) all F1 scores. Other statistics and plots are given at points of interesting behaviour.

D.1. DeliciousT140

	DeliciousT140		
Technique	Training stage (seconds per post)	Testing stage (seconds per post)	Memory usage (MB)
Popular global tags	0.000003	0.00022	1.65
Popular user tags	0.00001	0.00027	14.25
TF.IDF	0.00001	0.00026	3.01
TF.IWF	0.00001	0.00026	3.00
TF.IWF^2	0.00001	0.00026	3.00
WF.IDF	0.00001	0.00026	3.01
BTF.IDF	0.00001	0.00025	3.01
NTF.IDF (a = 0)	0.00001	0.00025	3.01
NTF.IDF (a = 0.1)	0.00001	0.00025	3.01
NTF.IDF (a = 0.2)	0.00001	0.00025	3.01
NTF.IDF (a = 0.3)	0.00001	0.00025	3.01
NTF.IDF (a = 0.4)	0.00001	0.00025	3.01
NTF.IDF (a = 0.5)	0.00001	0.00024	3.01
NTF.IDF (a = 0.6)	0.00001	0.00025	3.01
NTF.IDF (a = 0.7)	0.00001	0.00024	3.01
NTF.IDF (a = 0.8)	0.00001	0.00025	3.01
NTF.IDF (a = 0.9)	0.00001	0.00024	3.01
NTF.IDF (a = 1)	0.00001	0.00025	3.01
LSI (5 topics)	0.00026	0.00413	9.99
LSI (10 topics)	0.00026	0.00422	15.82
LSI (20 topics)	0.00028	0.00449	27.53
LSI (30 topics)	0.00024	0.00478	39.19
LSI (40 topics)	0.00024	0.00486	50.86
LSI (50 topics)	0.00026	0.00487	62.45
LSI (60 topics)	0.00030	0.00481	74.07
LSI (70 topics)	0.00029	0.00489	85.60
LSI (80 topics)	0.00032	0.00484	97.11
LSI (90 topics)	0.00034	0.00489	108.66
LSI (100 topics)	0.00035	0.00485	120.16
LDA (5 topics)	0.00135	0.00535	14.66
LDA (10 topics)	0.00119	0.00578	26.35
LDA (20 topics)	0.00113	0.00773	48.25
LDA (30 topics)	0.00112	0.00875	66.47
LDA (40 topics)	0.00116	0.00884	87.43
LDA (50 topics)	0.00118	0.01011	107.06
LDA (60 topics)	0.00119	0.01176	131.90
LDA (70 topics)	0.00123	0.01324	154.95
LDA (80 topics)	0.00125	0.01490	178.07
LDA (90 topics)	0.00127	0.01627	201.04
LDA (100 topics)	0.00128	0.01664	220.89
LDA (Wikipedia) (50 topics)	n.a.	0.02094	204.76
LDA (Wikipedia) (100 topics)	n.a.	0.03113	426.59
LDA (Wikipedia) (200 topics)	n.a.	0.05737	842.72
LDA (Wikipedia) (300 topics)	n.a.	0.08221	1258.31
LDA (Wikipedia) (400 topics)	n.a.	0.11109	1837.82
LDA (Wikipedia) (500 topics)	n.a.	0.13807	2330.83
Lipczak	0.00328	0.02710	193.28
TitleToTitle	0.00399	0.02729	251.75
Tag popularity multiplier	0.00312	0.02590	193.28
TitleToTitle & popularity multiplier	0.00398	0.02739	251.75

Table D.1: Processing time per post and memory usage results on DeliciousT140. Times are measured over the entire training/testing stage and divided by the number of posts.

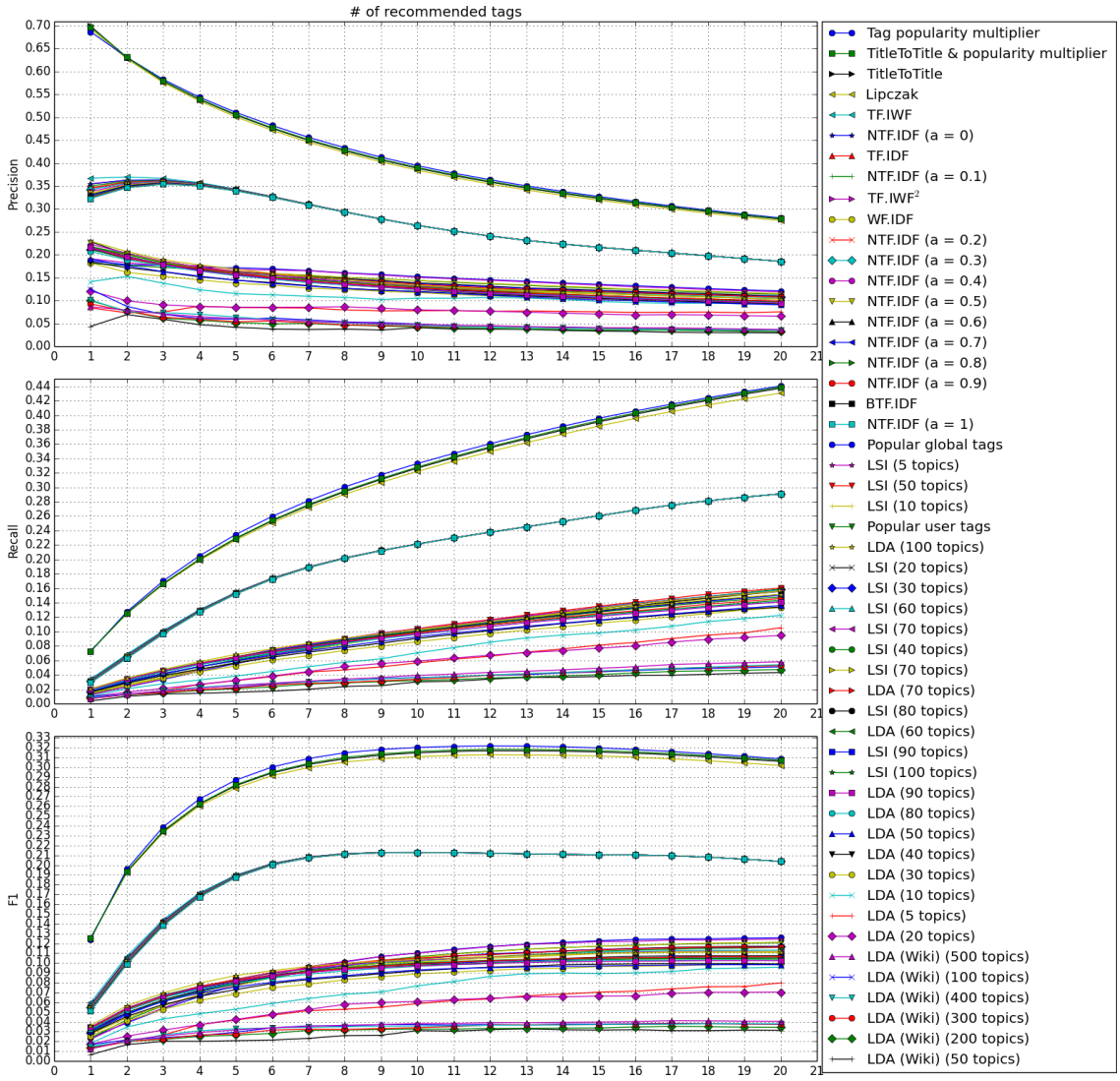


Figure D.1: Precision, Recall and F1 results on DeliciousT140 dataset.
Algorithms are sorted by summed F1 scores.

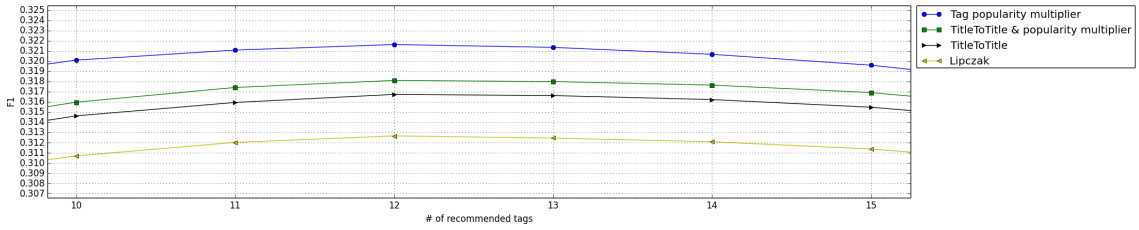


Figure D.2: Top F1 results between 10-15 tags on DeliciousT140 dataset.
Algorithms are sorted by summed F1 scores.

	# of tags																			
Technique	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Popular global tags	0.0236	0.0389	0.0531	0.0659	0.0773	0.0871	0.0957	0.1011	0.1066	0.1103	0.1140	0.1169	0.1193	0.1210	0.1228	0.1240	0.1245	0.1248	0.1254	0.1258
Popular user tags	0.0234	0.0419	0.0564	0.0678	0.0772	0.0851	0.0918	0.0977	0.1025	0.1065	0.1094	0.1120	0.1141	0.1158	0.1171	0.1182	0.1191	0.1198	0.1203	0.1206
TF.IDF	0.0570	0.1046	0.1426	0.1706	0.1893	0.2014	0.2082	0.2116	0.2129	0.2130	0.2127	0.2121	0.2115	0.2110	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
TF.IWF	0.0595	0.1070	0.1442	0.1716	0.1898	0.2017	0.2084	0.2117	0.2129	0.2130	0.2127	0.2122	0.2115	0.2110	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
TF.IWF^2	0.0551	0.1030	0.1415	0.1699	0.1889	0.2012	0.2081	0.2115	0.2128	0.2129	0.2127	0.2121	0.2115	0.2110	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
WF.IDF	0.0553	0.1030	0.1415	0.1698	0.1888	0.2011	0.2081	0.2115	0.2128	0.2129	0.2127	0.2121	0.2115	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
BTF.IDF	0.0509	0.0988	0.1381	0.1674	0.1873	0.2001	0.2074	0.2111	0.2125	0.2127	0.2126	0.2120	0.2114	0.2109	0.2106	0.2103	0.2095	0.2081	0.2060	0.2038
NTF.IDF (a = 0)	0.0570	0.1046	0.1426	0.1706	0.1893	0.2014	0.2082	0.2116	0.2129	0.2130	0.2127	0.2121	0.2115	0.2110	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.1)	0.0562	0.1037	0.1419	0.1701	0.1890	0.2012	0.2081	0.2115	0.2128	0.2129	0.2127	0.2121	0.2115	0.2110	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.2)	0.0553	0.1030	0.1414	0.1697	0.1888	0.2011	0.2080	0.2115	0.2128	0.2129	0.2127	0.2121	0.2115	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.3)	0.0545	0.1023	0.1409	0.1693	0.1886	0.2009	0.2080	0.2114	0.2128	0.2129	0.2127	0.2121	0.2115	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.4)	0.0538	0.1016	0.1404	0.1690	0.1883	0.2008	0.2078	0.2114	0.2127	0.2129	0.2127	0.2121	0.2115	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.5)	0.0531	0.1010	0.1399	0.1687	0.1881	0.2006	0.2078	0.2113	0.2127	0.2128	0.2126	0.2121	0.2115	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.6)	0.0524	0.1004	0.1395	0.1684	0.1879	0.2005	0.2077	0.2113	0.2126	0.2128	0.2126	0.2121	0.2114	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.7)	0.0520	0.0999	0.1391	0.1681	0.1877	0.2004	0.2076	0.2112	0.2126	0.2128	0.2126	0.2120	0.2114	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.8)	0.0516	0.0994	0.1387	0.1678	0.1875	0.2003	0.2075	0.2112	0.2126	0.2128	0.2126	0.2120	0.2114	0.2109	0.2106	0.2103	0.2095	0.2081	0.2061	0.2038
NTF.IDF (a = 0.9)	0.0512	0.0991	0.1384	0.1676	0.1874	0.2002	0.2074	0.2111	0.2126	0.2128	0.2126	0.2120	0.2114	0.2109	0.2106	0.2103	0.2095	0.2081	0.2060	0.2038
NTF.IDF (a = 1)	0.0509	0.0988	0.1381	0.1674	0.1873	0.2001	0.2074	0.2111	0.2125	0.2127	0.2126	0.2120	0.2114	0.2109	0.2106	0.2103	0.2095	0.2081	0.2060	0.2038
LSI (5 topics)	0.0219	0.0397	0.0540	0.0684	0.0773	0.0875	0.0963	0.1016	0.1067	0.1101	0.1135	0.1167	0.1190	0.1203	0.1218	0.1221	0.1236	0.1234	0.1240	0.1244
LSI (10 topics)	0.0273	0.0439	0.0565	0.0680	0.0766	0.0849	0.0917	0.0970	0.1018	0.1056	0.1091	0.1117	0.1139	0.1157	0.1173	0.1178	0.1193	0.1202	0.1206	0.1213
LSI (20 topics)	0.0311	0.0495	0.0624	0.0721	0.0792	0.0869	0.0927	0.0975	0.1018	0.1042	0.1071	0.1089	0.1101	0.1125	0.1137	0.1148	0.1159	0.1166	0.1170	0.1170
LSI (30 topics)	0.0288	0.0474	0.0611	0.0716	0.0790	0.0861	0.0918	0.0974	0.1015	0.1042	0.1072	0.1094	0.1108	0.1123	0.1139	0.1149	0.1154	0.1157	0.1160	0.1167
LSI (40 topics)	0.0289	0.0491	0.0615	0.0706	0.0788	0.0854	0.0914	0.0960	0.0999	0.1021	0.1047	0.1065	0.1082	0.1101	0.1122	0.1134	0.1141	0.1150	0.1153	0.1159
LSI (50 topics)	0.0299	0.0519	0.0658	0.0749	0.0836	0.0904	0.0948	0.0993	0.1029	0.1049	0.1072	0.1085	0.1104	0.1120	0.1135	0.1145	0.1153	0.1163	0.1161	0.1160
LSI (60 topics)	0.0305	0.0530	0.0657	0.0752	0.0832	0.0897	0.0937	0.0976	0.1007	0.1025	0.1044	0.1062	0.1075	0.1093	0.1109	0.1120	0.1127	0.1130	0.1129	0.1132
LSI (70 topics)	0.0321	0.0536	0.0668	0.0762	0.0827	0.0897	0.0942	0.0978	0.1013	0.1029	0.1044	0.1061	0.1076	0.1087	0.1099	0.1110	0.1116	0.1117	0.1116	0.1115
LSI (70 topics)	0.0308	0.0528	0.0661	0.0746	0.0820	0.0884	0.0920	0.0958	0.0995	0.1011	0.1032	0.1049	0.1062	0.1079	0.1096	0.1105	0.1112	0.1114	0.1113	0.1116
LSI (80 topics)	0.0314	0.0519	0.0645	0.0744	0.0802	0.0865	0.0905	0.0940	0.0968	0.0991	0.1009	0.1023	0.1035	0.1047	0.1058	0.1068	0.1072	0.1074	0.1074	0.1076
LSI (90 topics)	0.0315	0.0517	0.0649	0.0748	0.0818	0.0878	0.0916	0.0951	0.0976	0.0988	0.0998	0.1016	0.1025	0.1036	0.1044	0.1053	0.1058	0.1061	0.1061	0.1060
LSI (100 topics)	0.0323	0.0538	0.0672	0.0760	0.0821	0.0868	0.0903	0.0939	0.0964	0.0978	0.0988	0.1002	0.1014	0.1026	0.1032	0.1039	0.1042	0.1043	0.1041	0.1039
LDA (5 topics)	0.0129	0.0196	0.0266	0.0371	0.0418	0.0467	0.0514	0.0528	0.0550	0.0585	0.0616	0.0634	0.0665	0.0683	0.0702	0.0712	0.0736	0.0757	0.0761	0.0797
LDA (10 topics)	0.0181	0.0346	0.0429	0.0481	0.0529	0.0587	0.0637	0.0680	0.0704	0.0766	0.0811	0.0863	0.0890	0.0899	0.0893	0.0900	0.0914	0.0943	0.0950	0.0957
LDA (20 topics)	0.0170	0.0253	0.0315	0.0371	0.0424	0.0477	0.0526	0.0578	0.0597	0.0607	0.0627	0.0641	0.0653	0.0654	0.0663	0.0665	0.0689	0.0701	0.0700	0.0702
LDA (30 topics)	0.0255	0.0409	0.0527	0.0617	0.0685	0.0748	0.0783	0.0830	0.0859	0.0890	0.0907	0.0926	0.0941	0.0947	0.0964	0.0966	0.0973	0.0981	0.0992	0.0991
LDA (40 topics)	0.0283	0.0455	0.0577	0.0660	0.0729	0.0791	0.0831	0.0860	0.0890	0.0921	0.0940	0.0957	0.0968	0.0983	0.0987	0.0992	0.0994	0.0995	0.0994	0.0990
LDA (50 topics)	0.0304	0.0486	0.0607	0.0695	0.0755	0.0804	0.0842	0.0870	0.0903	0.0929	0.0945	0.0952	0.0961	0.0965	0.0970	0.0976	0.0979	0.0983	0.0983	0.0983
LDA (60 topics)	0.0339	0.0530	0.0654	0.0753	0.0821	0.0866	0.0917	0.0950	0.0977	0.1001	0.1015	0.1028	0.1037	0.1041	0.1046	0.1049	0.1049	0.1049	0.1051	0.1052
LDA (70 topics)	0.0337	0.0525	0.0655	0.0762	0.0829	0.0871	0.0910	0.0942	0.0970	0.0990	0.1009	0.1023	0.1032	0.1045	0.1053	0.1052	0.1057	0.1064	0.1063	0.1063
LDA (80 topics)	0.0321	0.0517	0.0645	0.0735	0.0796	0.0847	0.0886	0.0919	0.0944	0.0966	0.0982	0.0995	0.1012	0.1021	0.1027	0.1028	0.1028	0.1031	0.1030	0.1028
LDA (90 topics)	0.0345	0.0542	0.0667	0.0749	0.0814	0.0866	0.0903	0.0934	0.0954	0.0971	0.0983	0.0995	0.1006	0.1012	0.1015	0.1018	0.1020	0.1025	0.1025	0.1025
LDA (100 topics)	0.0361	0.0564	0.0694	0.0795	0.0872	0.0922	0.0965	0.0997	0.1017	0.1038	0.1051	0.1066	0.1074	0.1088	0.1098	0.1104	0.1108	0.1110	0.1109	0.1107
LDA (Wiki) (50 topics)	0.0064	0.0167	0.0201	0.0200	0.0208	0.0214	0.0231	0.0259	0.0263	0.0310	0.0305	0.0319	0.0328	0.0319	0.0316	0.0319	0.0310	0.0310	0.0316	0.0313
LDA (Wiki) (100 topics)	0.0168	0.0214	0.0239	0.0259	0.0294	0.0339	0.0358	0.0358	0.0372	0.0371	0.0366	0.0373	0.0368	0.0378	0.0377	0.0381	0.0384	0.0380	0.0378	0.0376
LDA (Wiki) (200 topics)	0.0137	0.0200	0.0219	0.0251	0.0264	0.0282	0.0314	0.0316	0.0321	0.0326	0.0325	0.0333	0.0334	0.0336	0.0337	0.0347	0.0352	0.0350	0.0345	0.0343
LDA (Wiki) (300 topics)	0.0127	0.0199	0.0226	0.0265	0.0277	0.0315	0.0324	0.0322	0.0332	0.0340	0.0350	0.0368	0.0369	0.0372	0.0372	0.0378	0.0381	0.0380	0.0377	0.0376
LDA (Wiki) (400 topics)	0.0150	0.0198	0.0266	0.0308	0.0329	0.0338	0.0344	0.0350	0.0355	0.0357	0.0358	0.0365	0.0367	0.0368	0.0373	0.0373	0.0375	0.0379	0.0382	0.0380
LDA (Wiki) (500 topics)	0.0123	0.0211	0.0256	0.0290	0.0317	0.0339	0.0354	0.0365	0.0371	0.0383	0.0383	0.0392	0.0390	0.0393	0.0397	0.0401	0.0412	0.0410	0.0406	0.0404
Lipczak	0.1249	0.1924	0.2332	0.2604	0.2786	0.2912	0.2995	0.3052	0.3085	0.3107	0.3120	0.3127	0.3124	0.3121	0.3114	0.3102	0.3084	0.3064	0.3041	0.3017
TitleToTitle	0.1254	0.1931	0.2343	0.2621	0.2811	0.2940	0.3027	0.3085	0.3122	0.3146	0.3159	0.3167	0.3166	0.3162	0.3155	0.3142	0.3125	0.3104	0.3082	0.3058
Tag popularity multiplier	0.1241	0.1958	0.2389	0.2674	0.2868	0.3000	0.3088	0.3146	0.3180	0.3201	0.3211	0.3216	0.3214	0.3207	0.3196	0.3179	0.3159	0.3136	0.3111	0.3084
TitleToTitle & popularity multiplier	0.1251	0.1935	0.2349	0.2628	0.2820	0.2950	0.3038	0.3099	0.3135	0.3160	0.3174	0.3181	0.3180	0.3176	0.3169	0.3155	0.3138	0.3118	0.3094	0.3069

Table D.2: F1 results on DeliciousT140 dataset.
Green = best result per algorithm, orange = best overall result.

D.2. Wiki10+

	Wiki10+		
Technique	Training stage (seconds per post)	Testing stage (seconds per post)	Memory usage (MB)
Popular global tags	0.00001	0.00023	2.84
Popular user tags	0.00003	0.00024	7.46
TF.IDF	0.00002	0.00024	5.03
TF.IWF	0.00002	0.00024	5.02
TF.IWF^2	0.00002	0.00025	5.02
WF.IDF	0.00002	0.00024	5.03
BTF.IDF	0.00002	0.00024	5.03
NTF.IDF (a = 0)	0.00002	0.00025	5.03
NTF.IDF (a = 0.1)	0.00002	0.00025	5.03
NTF.IDF (a = 0.2)	0.00002	0.00026	5.03
NTF.IDF (a = 0.3)	0.00002	0.00025	5.03
NTF.IDF (a = 0.4)	0.00002	0.00025	5.03
NTF.IDF (a = 0.5)	0.00002	0.00026	5.03
NTF.IDF (a = 0.6)	0.00002	0.00025	5.03
NTF.IDF (a = 0.7)	0.00002	0.00026	5.03
NTF.IDF (a = 0.8)	0.00002	0.00026	5.03
NTF.IDF (a = 0.9)	0.00002	0.00026	5.03
NTF.IDF (a = 1)	0.00002	0.00025	5.03
LSI (5 topics)	0.00049	0.00593	16.14
LSI (10 topics)	0.00051	0.00629	25.10
LSI (20 topics)	0.00054	0.00624	42.94
LSI (30 topics)	0.00041	0.00678	60.55
LSI (40 topics)	0.00043	0.00676	78.23
LSI (50 topics)	0.00045	0.00651	95.85
LSI (60 topics)	0.00048	0.00648	113.50
LSI (70 topics)	0.00049	0.00648	130.90
LSI (80 topics)	0.00053	0.00674	148.31
LSI (90 topics)	0.00056	0.00662	165.74
LSI (100 topics)	0.00061	0.00670	183.10
LDA (5 topics)	0.00353	0.00820	24.13
LDA (10 topics)	0.00330	0.00948	42.96
LDA (20 topics)	0.00310	0.01166	74.07
LDA (30 topics)	0.00303	0.01312	108.09
LDA (40 topics)	0.00303	0.01293	143.11
LDA (50 topics)	0.00307	0.01574	183.17
LDA (60 topics)	0.00335	0.01822	211.83
LDA (70 topics)	0.00349	0.02093	241.88
LDA (80 topics)	0.00344	0.02374	281.38
LDA (90 topics)	0.00375	0.02730	316.67
LDA (100 topics)	0.00379	0.02496	357.30
LDA (Wikipedia) (50 topics)	n.a.	0.02207	205.95
LDA (Wikipedia) (100 topics)	n.a.	0.03040	427.78
LDA (Wikipedia) (200 topics)	n.a.	0.05649	843.91
LDA (Wikipedia) (300 topics)	n.a.	0.08350	1259.50
LDA (Wikipedia) (400 topics)	n.a.	0.10329	1839.01
LDA (Wikipedia) (500 topics)	n.a.	0.12746	2332.03
Lipczak	0.00153	0.00716	164.98
TitleToTitle	0.00158	0.00687	166.36
Tag popularity multiplier	0.00142	0.00691	164.98
TitleToTitle & popularity multiplier	0.00157	0.00703	166.37

Table D.3: Processing time per post and memory usage results on Wiki10+. Times are measured over the entire training/testing stage and divided by the number of posts.

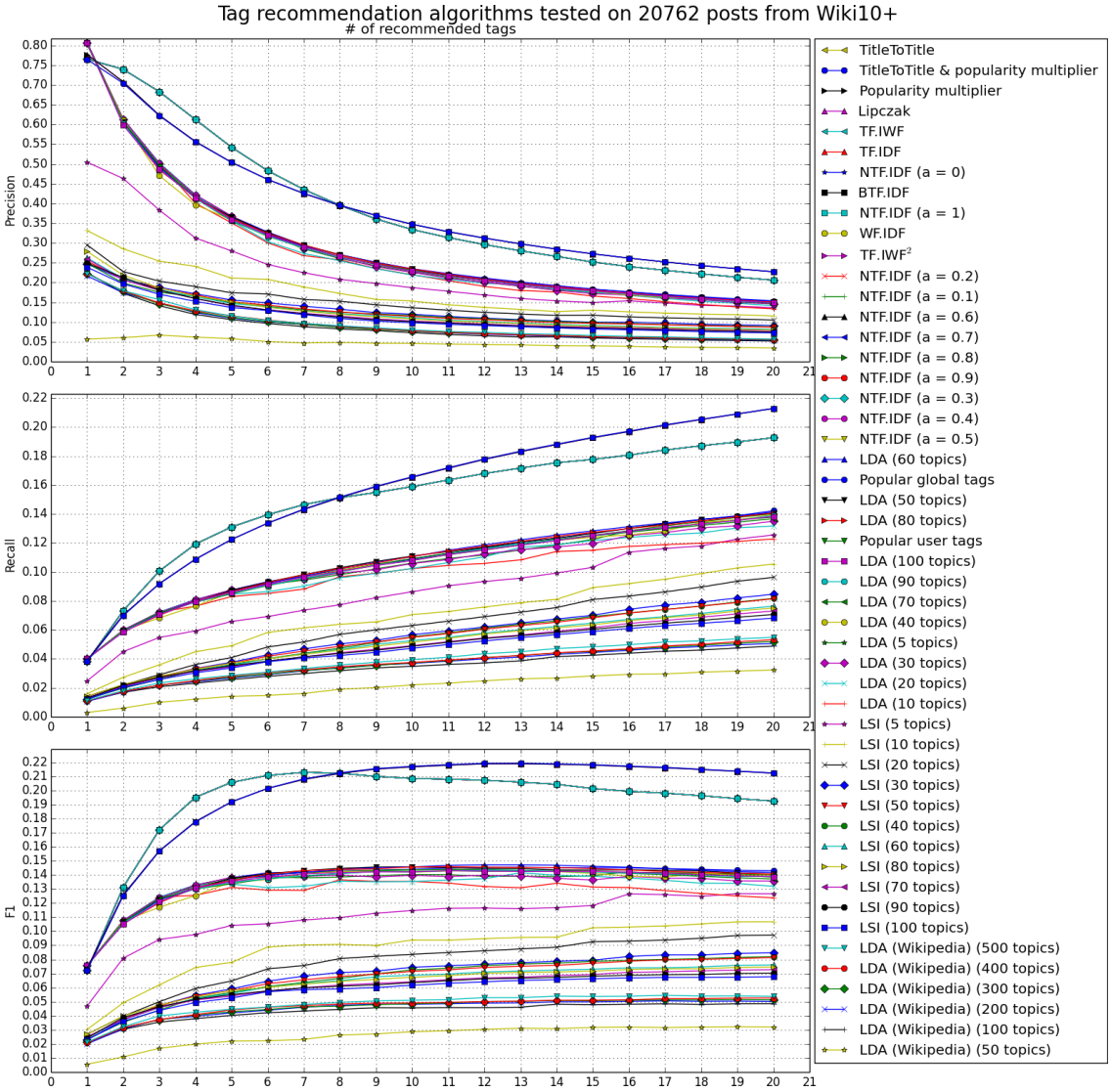


Figure D.3: Precision, Recall and F1 results on Wiki10+ dataset.
Algorithms are sorted by summed F1 scores.

We see two plots near the top: the light-blue line at the top between 1-8 tags (TF.IDF & variants) and the dark-blue line >8 tags (Lipczak & contributions). In Figure D.4 we zoom in on the results between 5-10 tags.

In Figure D.5 we zoom on the Lipczak results between 9-17 tags. We see two lines: the TitleToTitle recommender (& popularity multiplier) and second the original system (& popularity multiplier). The popularity multiplier appears to have no significant effect on the results, but the TitleToTitle recommender shows an improvement on the original system.

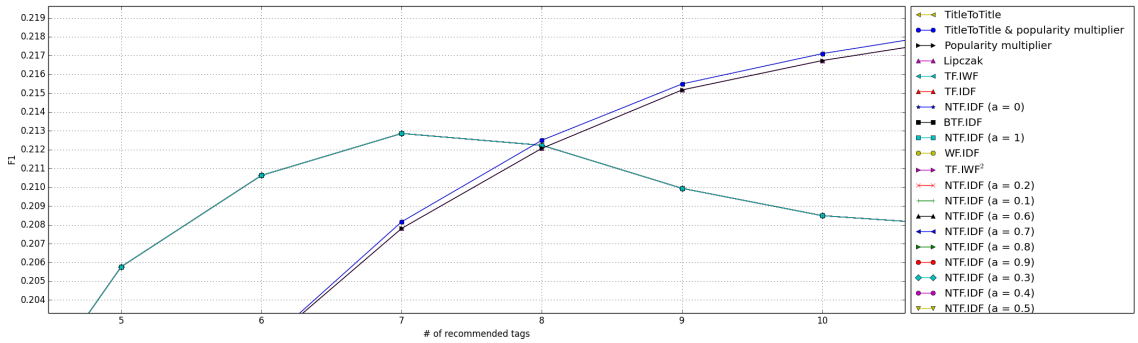


Figure D.4: F1 results on Wiki10+ dataset between 5-10 recommended tags. We see Lipczak and additions increasing in F1 score while TF.IDF and variants decrease after 7 tags.

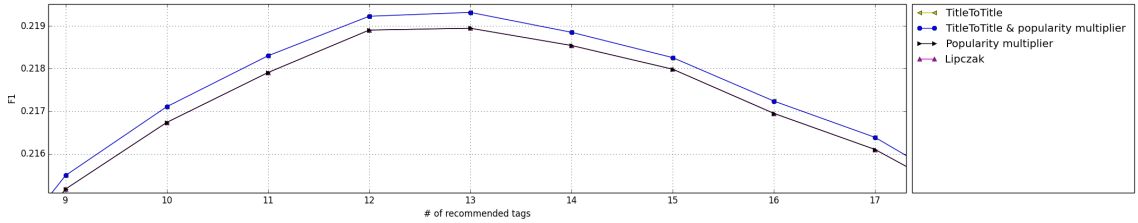


Figure D.5: Maximum F1 results on Wiki10+ dataset between 9-17 tags.

Wiki10+ consists of 20762 posts. 80% = 16610 posts serve as training data and 20% = 4152 posts serve as test data. Lipczak's system trains on 80% of the training data = 13288 posts and learns parameters on 20% of the training data = 3322 posts. The recommenders produced the following statistics:

Title recommender Learned on 26346 titlewords of which 13003 unique, 1.983 average titlewords per post (1.981 unique)

TitleToTag recommender Learned on 13003 titlewords and 452987 related tagwords = 34.84 related tagwords per titleword

TagToTag recommender Learned on 293074 tagwords, of which 72761 unique and 4741887 relations = 65.17 related tagwords per tagword

TitleToTitle recommender Learned on 26346 titlewords of which 13003 unique and 32780 relations = 2.52 related titlewords per titleword

	# of tags																			
Technique	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Popular global tags	0.0757	0.1076	0.1241	0.1328	0.1380	0.1415	0.1421	0.1430	0.1434	0.1440	0.1447	0.1453	0.1455	0.1451	0.1451	0.1450	0.1445	0.1440	0.1433	0.1429
Popular user tags	0.0754	0.1068	0.1235	0.1319	0.1370	0.1402	0.1416	0.1426	0.1433	0.1438	0.1438	0.1437	0.1434	0.1428	0.1422	0.1418	0.1417	0.1412	0.1407	0.1400
TF.IDF	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
TF.IWF	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
TF.IWF^2	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
WF.IDF	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
BTF.IDF	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.1)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.2)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.3)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.4)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.5)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.6)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.7)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.8)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 0.9)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
NTF.IDF (a = 1)	0.0721	0.1313	0.1720	0.1951	0.2058	0.2106	0.2129	0.2122	0.2099	0.2085	0.2080	0.2073	0.2059	0.2043	0.2014	0.1994	0.1981	0.1963	0.1942	0.1924
LSI (5 topics)	0.0469	0.0809	0.0942	0.0977	0.1042	0.1053	0.1080	0.1097	0.1128	0.1147	0.1163	0.1165	0.1159	0.1168	0.1183	0.1265	0.1260	0.1248	0.1267	0.1266
LSI (10 topics)	0.0303	0.0492	0.0619	0.0743	0.0779	0.0888	0.0903	0.0907	0.0899	0.0938	0.0937	0.0947	0.0956	0.0958	0.1025	0.1030	0.1037	0.1052	0.1067	0.1067
LSI (20 topics)	0.0269	0.0396	0.0500	0.0593	0.0647	0.0733	0.0757	0.0806	0.0822	0.0836	0.0849	0.0861	0.0874	0.0887	0.0926	0.0929	0.0938	0.0950	0.0970	0.0973
LSI (30 topics)	0.0229	0.0369	0.0474	0.0545	0.0591	0.0646	0.0682	0.0707	0.0717	0.0744	0.0755	0.0766	0.0777	0.0787	0.0794	0.0821	0.0832	0.0832	0.0842	0.0847
LSI (40 topics)	0.0233	0.0354	0.0440	0.0525	0.0565	0.0607	0.0636	0.0666	0.0697	0.0726	0.0737	0.0755	0.0767	0.0773	0.0786	0.0794	0.0800	0.0806	0.0815	0.0819
LSI (50 topics)	0.0246	0.0376	0.0471	0.0540	0.0578	0.0627	0.0655	0.0678	0.0696	0.0715	0.0726	0.0743	0.0751	0.0758	0.0773	0.0788	0.0796	0.0800	0.0807	0.0813
LSI (60 topics)	0.0245	0.0373	0.0457	0.0523	0.0558	0.0603	0.0628	0.0650	0.0677	0.0688	0.0696	0.0711	0.0717	0.0724	0.0731	0.0740	0.0741	0.0747	0.0758	0.0761
LSI (70 topics)	0.0250	0.0373	0.0459	0.0512	0.0540	0.0575	0.0599	0.0618	0.0629	0.0642	0.0656	0.0669	0.0674	0.0685	0.0693	0.0709	0.0712	0.0719	0.0724	0.0727
LSI (80 topics)	0.0268	0.0398	0.0476	0.0534	0.0579	0.0602	0.0625	0.0644	0.0659	0.0673	0.0685	0.0701	0.0708	0.0711	0.0717	0.0729	0.0733	0.0734	0.0744	0.0746
LSI (90 topics)	0.0242	0.0387	0.0464	0.0517	0.0549	0.0577	0.0598	0.0609	0.0619	0.0635	0.0649	0.0662	0.0666	0.0673	0.0681	0.0688	0.0692	0.0694	0.0700	0.0702
LSI (100 topics)	0.0230	0.0360	0.0438	0.0494	0.0526	0.0570	0.0586	0.0591	0.0600	0.0617	0.0630	0.0641	0.0650	0.0657	0.0663	0.0668	0.0673	0.0673	0.0674	0.0677
LDA (5 topics)	0.0757	0.1076	0.1241	0.1328	0.1380	0.1387	0.1379	0.1385	0.1395	0.1403	0.1402	0.1390	0.1391	0.1387	0.1395	0.1422	0.1403	0.1406	0.1400	0.1383
LDA (10 topics)	0.0757	0.1076	0.1241	0.1255	0.1311	0.1293	0.1291	0.1366	0.1354	0.1354	0.1342	0.1318	0.1309	0.1341	0.1314	0.1311	0.1289	0.1269	0.1250	0.1238
LDA (20 topics)	0.0757	0.1076	0.1240	0.1329	0.1335	0.1310	0.1319	0.1354	0.1351	0.1353	0.1364	0.1373	0.1415	0.1398	0.1390	0.1378	0.1362	0.1341	0.1339	0.1319
LDA (30 topics)	0.0757	0.1076	0.1236	0.1327	0.1383	0.1392	0.1390	0.1399	0.1384	0.1397	0.1388	0.1398	0.1392	0.1375	0.1364	0.1390	0.1377	0.1375	0.1358	0.1356
LDA (40 topics)	0.0757	0.1072	0.1170	0.1253	0.1352	0.1366	0.1407	0.1425	0.1427	0.1419	0.1428	0.1430	0.1430	0.1424	0.1411	0.1391	0.1385	0.1413	0.1396	0.1401
LDA (50 topics)	0.0757	0.1064	0.1225	0.1301	0.1377	0.1409	0.1430	0.1447	0.1456	0.1457	0.1451	0.1445	0.1445	0.1436	0.1434	0.1436	0.1433	0.1428	0.1414	0.1407
LDA (60 topics)	0.0757	0.1058	0.1223	0.1310	0.1362	0.1397	0.1417	0.1439	0.1451	0.1456	0.1468	0.1471	0.1471	0.1469	0.1461	0.1455	0.1444	0.1434	0.1423	0.1418
LDA (70 topics)	0.0757	0.1064	0.1231	0.1300	0.1337	0.1374	0.1389	0.1410	0.1426	0.1426	0.1430	0.1426	0.1428	0.1424	0.1421	0.1411	0.1402	0.1390	0.1380	0.1371
LDA (80 topics)	0.0757	0.1054	0.1202	0.1310	0.1360	0.1404	0.1431	0.1439	0.1443	0.1456	0.1459	0.1455	0.1453	0.1452	0.1444	0.1435	0.1424	0.1418	0.1412	0.1406
LDA (90 topics)	0.0757	0.1058	0.1212	0.1302	0.1347	0.1373	0.1395	0.1409	0.1424	0.1421	0.1430	0.1434	0.1433	0.1426	0.1420	0.1414	0.1407	0.1398	0.1388	0.1391
LDA (100 topics)	0.0757	0.1050	0.1209	0.1307	0.1354	0.1384	0.1407	0.1417	0.1421	0.1421	0.1430	0.1427	0.1424	0.1424	0.1425	0.1420	0.1412	0.1402	0.1394	0.1388
LDA (Wikipedia) (50 topics)	0.0056	0.0108	0.0169	0.0198	0.0220	0.0222	0.0232	0.0264	0.0271	0.0287	0.0293	0.0304	0.0311	0.0308	0.0317	0.0321	0.0316	0.0321	0.0322	0.0321
LDA (Wikipedia) (100 topics)	0.0209	0.0303	0.0355	0.0380	0.0403	0.0422	0.0433	0.0446	0.0457	0.0456	0.0457	0.0457	0.0461	0.0482	0.0479	0.0482	0.0485	0.0481	0.0485	0.0485
LDA (Wikipedia) (200 topics)	0.0200	0.0308	0.0375	0.0396	0.0421	0.0439	0.0461	0.0474	0.0483	0.0482	0.0485	0.0490	0.0491	0.0499	0.0498	0.0502	0.0506	0.0503	0.0504	0.0501
LDA (Wikipedia) (300 topics)	0.0212	0.0316	0.0371	0.0405	0.0428	0.0445	0.0463	0.0469	0.0479	0.0485	0.0490	0.0500	0.0503	0.0509	0.0506	0.0510	0.0514	0.0514	0.0516	0.0514
LDA (Wikipedia) (400 topics)	0.0208	0.0315	0.0373	0.0409	0.0441	0.0461	0.0472	0.0484	0.0490	0.0491	0.0495	0.0500	0.0504	0.0511	0.0511	0.0514	0.0521	0.0521	0.0526	0.0526
LDA (Wikipedia) (500 topics)	0.0213	0.0323	0.0399	0.0425	0.0446	0.0462	0.0480	0.0495	0.0505	0.0510	0.0515	0.0528	0.0529	0.0539	0.0537	0.0539	0.0545	0.0540	0.0541	0.0538
Lipczak	0.0734	0.1259	0.1571	0.1779	0.1919	0.2014	0.2078	0.2121	0.2152	0.2167	0.2179	0.2189	0.2189	0.2185	0.2180	0.2170	0.2161	0.2148	0.2136	0.2123
TitleToTitle	0.0721	0.1252	0.1568	0.1776	0.1918	0.2015	0.2082	0.2125	0.2155	0.2171	0.2183	0.2192	0.2193	0.2189	0.2183	0.2172	0.2164	0.2150	0.2137	0.2124
Popularity multiplier	0.0734	0.1259	0.1571	0.1779	0.1919	0.2014	0.2078	0.2121	0.2152	0.2167	0.2179	0.2189	0.2189	0.2185	0.2180	0.2169	0.2161	0.2148	0.2136	0.2123
TitleToTitle & popularity multiplier	0.0721	0.1252	0.1568	0.1776	0.1918	0.2015	0.2082	0.2125	0.2155	0.2171	0.2183	0.2192	0.2193	0.2189	0.2183	0.2172	0.2164	0.2150	0.2137	0.2124

Table D.4: F1 results on Wiki10+ dataset.
Green = best result per algorithm, orange = best overall result.

Fold	TitleToTag-TagToTag
1	0.18
2	0.02
3	0.13
4	0.18
5	0.11

(a) Lipczak's content-based merge coefficients

Fold	Title-TitleToTitle	TitleToTag-TagToTag
1	1	0.19
2	1	0.01
3	1	0.13
4	1	0.05
5	0.74	0.12

(b) Merge coefficients with TitleToTitle

Fold	TitleToTag-TagToTag
1	0.18
2	0.02
3	0.13
4	0.18
5	0.11

(c) Merge coefficients with popularity multiplication

Fold	Title-TitleToTitle	TitleToTag-TagToTag with popularity multiplier
1	1	0.19
2	1	0.01
3	1	0.13
4	1	0.05
5	0.74	0.12

(d) Merge coefficients with TitleToTitle and popularity multiplication

Table D.5: Merge coefficients of different systems on Wiki10+

D.3. MovieLens 10M

	MovieLens 10M		
Technique	Training stage (seconds per post)	Testing stage (seconds per post)	Memory usage (MB)
Popular global tags	0.000001	0.00015	0.32
Popular user tags	0.000002	0.00024	1.39
TF.IDF	0.000002	0.00016	0.60
TF.IWF	0.000002	0.00016	0.61
TF.IWF^2	0.000002	0.00017	0.61
WF.IDF	0.000002	0.00016	0.60
BTF.IDF	0.000002	0.00016	0.60
NTF.IDF (a = 0)	0.000002	0.00017	0.60
NTF.IDF (a = 0.1)	0.000002	0.00016	0.60
NTF.IDF (a = 0.2)	0.000002	0.00016	0.60
NTF.IDF (a = 0.3)	0.000002	0.00016	0.60
NTF.IDF (a = 0.4)	0.000002	0.00016	0.60
NTF.IDF (a = 0.5)	0.000002	0.00016	0.60
NTF.IDF (a = 0.6)	0.000002	0.00016	0.60
NTF.IDF (a = 0.7)	0.000002	0.00016	0.60
NTF.IDF (a = 0.8)	0.000002	0.00015	0.60
NTF.IDF (a = 0.9)	0.000002	0.00016	0.60
NTF.IDF (a = 1)	0.000002	0.00016	0.60
LSI (5 topics)	0.00008	0.00127	2.03
LSI (10 topics)	0.00008	0.00120	3.25
LSI (20 topics)	0.00009	0.00134	5.71
LSI (30 topics)	0.00007	0.00126	8.13
LSI (40 topics)	0.00008	0.00124	10.56
LSI (50 topics)	0.00009	0.00126	12.98
LSI (60 topics)	0.00008	0.00139	15.40
LSI (70 topics)	0.00009	0.00133	17.79
LSI (80 topics)	0.00009	0.00136	20.18
LSI (90 topics)	0.00010	0.00134	22.56
LSI (100 topics)	0.00011	0.00131	24.99
LDA (5 topics)	0.00046	0.00132	3.10
LDA (10 topics)	0.00039	0.00130	5.28
LDA (20 topics)	0.00038	0.00172	9.63
LDA (30 topics)	0.00038	0.00194	13.90
LDA (40 topics)	0.00037	0.00186	19.23
LDA (50 topics)	0.00038	0.00222	23.56
LDA (60 topics)	0.00039	0.00273	27.34
LDA (70 topics)	0.00040	0.00306	32.35
LDA (80 topics)	0.00041	0.00338	37.87
LDA (90 topics)	0.00043	0.00379	42.27
LDA (100 topics)	0.00043	0.00418	44.77
LDA (Wikipedia) (50 topics)	n.a.	0.02145	203.36
LDA (Wikipedia) (100 topics)	n.a.	0.03140	425.19
LDA (Wikipedia) (200 topics)	n.a.	0.05861	841.32
LDA (Wikipedia) (300 topics)	n.a.	0.08984	1256.91
LDA (Wikipedia) (400 topics)	n.a.	0.11871	1836.42
LDA (Wikipedia) (500 topics)	n.a.	0.14396	2329.43
Lipczak	0.00014	0.00018	6.58
TitleToTitle	0.00065	0.00046	8.90
Tag popularity multiplier	0.00014	0.00018	6.58
TitleToTitle & popularity multiplier	0.00065	0.00045	8.90

Table D.6: Processing time per post and memory usage results on MovieLens 10M. Times are measured over the entire training/testing stage and divided by the number of posts.

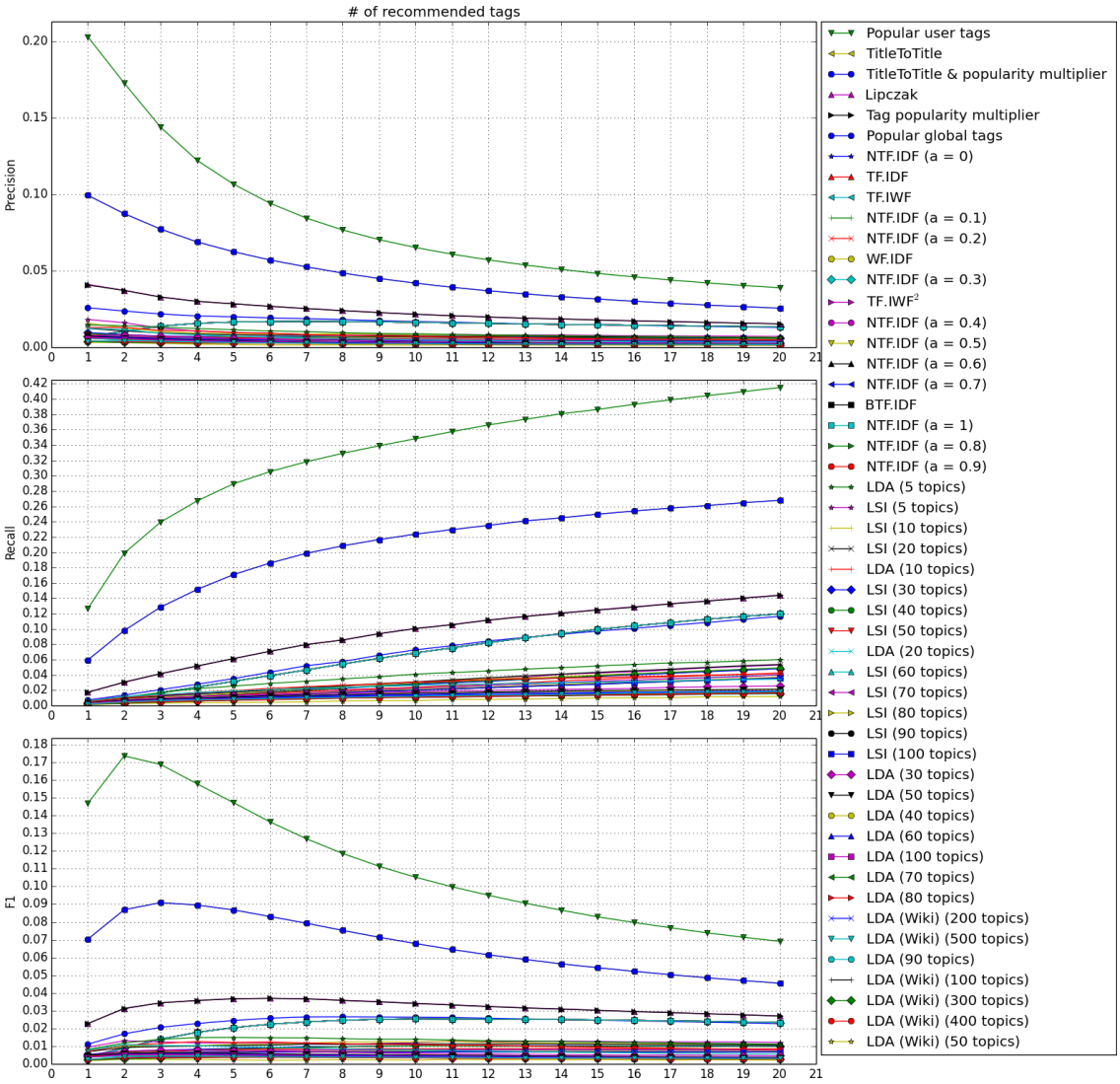


Figure D.6: Precision, Recall and F1 results on MovieLens 10M dataset.
Algorithms are sorted by summed F1 scores.

The popular user tags clearly scored the best with a peak at 2 tags giving an F1 score of 0.1736. The second best performing algorithm is Lipczak's system with the TitleToTitle recommender with a score of 0.0909 at 3 tags. Third is the original Lipczak system with a score of 0.0370 at 6 tags.

	# of tags																			
Technique	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Popular global tags	0.0111	0.0170	0.0206	0.0228	0.0245	0.0258	0.0265	0.0265	0.0265	0.0263	0.0262	0.0258	0.0254	0.0251	0.0247	0.0243	0.0239	0.0235	0.0232	0.0227
Popular user tags	0.1468	0.1736	0.1688	0.1579	0.1473	0.1364	0.1269	0.1186	0.1114	0.1052	0.0999	0.0951	0.0906	0.0867	0.0829	0.0797	0.0767	0.0739	0.0714	0.0691
TF.IDF	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
TF.IWF	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
TF.IWF^2	0.0040	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
WF.IDF	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
BTF.IDF	0.0037	0.0088	0.0139	0.0177	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0)	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.1)	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.2)	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.3)	0.0041	0.0090	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.4)	0.0040	0.0089	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.5)	0.0038	0.0089	0.0140	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.6)	0.0037	0.0089	0.0139	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.7)	0.0037	0.0088	0.0139	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.8)	0.0037	0.0088	0.0139	0.0178	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 0.9)	0.0037	0.0088	0.0139	0.0177	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
NTF.IDF (a = 1)	0.0037	0.0088	0.0139	0.0177	0.0205	0.0224	0.0237	0.0246	0.0252	0.0254	0.0253	0.0253	0.0252	0.0250	0.0248	0.0246	0.0243	0.0240	0.0237	0.0233
LSI (5 topics)	0.0094	0.0131	0.0124	0.0119	0.0119	0.0119	0.0115	0.0112	0.0114	0.0120	0.0128	0.0126	0.0126	0.0129	0.0127	0.0124	0.0124	0.0124	0.0123	0.0121
LSI (10 topics)	0.0078	0.0100	0.0100	0.0102	0.0111	0.0112	0.0117	0.0124	0.0125	0.0123	0.0127	0.0122	0.0119	0.0117	0.0116	0.0114	0.0113	0.0112	0.0109	0.0108
LSI (20 topics)	0.0069	0.0092	0.0098	0.0101	0.0104	0.0106	0.0112	0.0110	0.0114	0.0112	0.0113	0.0113	0.0113	0.0112	0.0111	0.0110	0.0109	0.0109	0.0108	0.0107
LSI (30 topics)	0.0054	0.0070	0.0077	0.0083	0.0087	0.0093	0.0098	0.0097	0.0101	0.0103	0.0102	0.0102	0.0102	0.0102	0.0102	0.0101	0.0101	0.0101	0.0100	0.0100
LSI (40 topics)	0.0051	0.0069	0.0074	0.0079	0.0085	0.0088	0.0093	0.0097	0.0101	0.0101	0.0102	0.0100	0.0100	0.0099	0.0100	0.0100	0.0099	0.0099	0.0099	0.0098
LSI (50 topics)	0.0049	0.0063	0.0070	0.0077	0.0080	0.0084	0.0085	0.0086	0.0086	0.0085	0.0086	0.0086	0.0086	0.0087	0.0087	0.0086	0.0086	0.0085	0.0085	0.0085
LSI (60 topics)	0.0046	0.0060	0.0065	0.0071	0.0073	0.0076	0.0077	0.0078	0.0080	0.0080	0.0080	0.0081	0.0080	0.0079	0.0079	0.0080	0.0080	0.0081	0.0081	0.0080
LSI (70 topics)	0.0046	0.0060	0.0065	0.0069	0.0071	0.0076	0.0076	0.0078	0.0079	0.0079	0.0079	0.0080	0.0080	0.0080	0.0080	0.0079	0.0079	0.0078	0.0078	0.0078
LSI (80 topics)	0.0049	0.0060	0.0063	0.0066	0.0067	0.0070	0.0071	0.0071	0.0072	0.0072	0.0072	0.0071	0.0072	0.0072	0.0072	0.0072	0.0072	0.0071	0.0072	0.0071
LSI (90 topics)	0.0042	0.0056	0.0059	0.0062	0.0065	0.0068	0.0068	0.0068	0.0069	0.0068	0.0068	0.0068	0.0068	0.0068	0.0068	0.0068	0.0069	0.0069	0.0069	0.0070
LSI (100 topics)	0.0046	0.0060	0.0062	0.0064	0.0064	0.0067	0.0067	0.0067	0.0068	0.0067	0.0068	0.0067	0.0067	0.0067	0.0067	0.0068	0.0068	0.0068	0.0068	0.0067
LDA (5 topics)	0.0081	0.0118	0.0139	0.0149	0.0149	0.0147	0.0147	0.0142	0.0139	0.0139	0.0134	0.0130	0.0129	0.0126	0.0123	0.0121	0.0119	0.0116	0.0114	0.0112
LDA (10 topics)	0.0072	0.0106	0.0115	0.0126	0.0121	0.0123	0.0118	0.0114	0.0110	0.0109	0.0105	0.0106	0.0101	0.0098	0.0094	0.0091	0.0089	0.0086	0.0084	0.0083
LDA (20 topics)	0.0076	0.0108	0.0103	0.0103	0.0098	0.0094	0.0090	0.0085	0.0082	0.0078	0.0074	0.0072	0.0069	0.0066	0.0064	0.0062	0.0060	0.0059	0.0058	0.0056
LDA (30 topics)	0.0048	0.0072	0.0079	0.0079	0.0077	0.0072	0.0069	0.0065	0.0063	0.0060	0.0058	0.0055	0.0054	0.0052	0.0050	0.0049	0.0050	0.0049	0.0048	0.0046
LDA (40 topics)	0.0042	0.0055	0.0059	0.0059	0.0058	0.0056	0.0054	0.0051	0.0050	0.0049	0.0047	0.0046	0.0044	0.0043	0.0042	0.0040	0.0039	0.0038	0.0037	0.0036
LDA (50 topics)	0.0048	0.0060	0.0062	0.0060	0.0059	0.0058	0.0055	0.0054	0.0052	0.0051	0.0049	0.0047	0.0046	0.0044	0.0043	0.0042	0.0040	0.0039	0.0038	0.0037
LDA (60 topics)	0.0041	0.0055	0.0057	0.0056	0.0054	0.0052	0.0050	0.0050	0.0048	0.0046	0.0044	0.0043	0.0042	0.0040	0.0039	0.0038	0.0036	0.0036	0.0035	0.0034
LDA (70 topics)	0.0038	0.0046	0.0049	0.0049	0.0047	0.0047	0.0045	0.0044	0.0042	0.0040	0.0039	0.0037	0.0036	0.0034	0.0033	0.0032	0.0032	0.0032	0.0031	0.0030
LDA (80 topics)	0.0043	0.0051	0.0050	0.0050	0.0047	0.0045	0.0043	0.0041	0.0041	0.0039	0.0038	0.0036	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0030	0.0029
LDA (90 topics)	0.0038	0.0044	0.0045	0.0044	0.0044	0.0043	0.0040	0.0039	0.0037	0.0036	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0027
LDA (100 topics)	0.0038	0.0051	0.0053	0.0052	0.0051	0.0050	0.0048	0.0046	0.0045	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036	0.0035	0.0034	0.0034	0.0033
LDA (Wiki) (50 topics)	0.0017	0.0026	0.0025	0.0024	0.0022	0.0021	0.0021	0.0023	0.0022	0.0021	0.0025	0.0024	0.0023	0.0023	0.0023	0.0022	0.0022	0.0023	0.0022	0.0021
LDA (Wiki) (100 topics)	0.0024	0.0029	0.0038	0.0037	0.0037	0.0039	0.0037	0.0039	0.0038	0.0036	0.0035	0.0034	0.0033	0.0033	0.0032	0.0031	0.0030	0.0031	0.0031	0.0030
LDA (Wiki) (200 topics)	0.0023	0.0035	0.0040	0.0042	0.0042	0.0044	0.0043	0.0042	0.0040	0.0039	0.0038	0.0036	0.0035	0.0035	0.0034	0.0033	0.0032	0.0031	0.0031	0.0030
LDA (Wiki) (300 topics)	0.0023	0.0028	0.0032	0.0034	0.0040	0.0040	0.0039	0.0038	0.0036	0.0035	0.0034	0.0033	0.0032	0.0032	0.0031	0.0030	0.0029	0.0029	0.0028	0.0027
LDA (Wiki) (400 topics)	0.0026	0.0035	0.0036	0.0036	0.0037	0.0037	0.0037	0.0036	0.0035	0.0033	0.0033	0.0032	0.0031	0.0031	0.0031	0.0030	0.0029	0.0029	0.0028	0.0027
LDA (Wiki) (500 topics)	0.0024	0.0041	0.0044	0.0043	0.0042	0.0041	0.0040	0.0038	0.0038	0.0036	0.0036	0.0035	0.0034	0.0033	0.0033	0.0033	0.0032	0.0031	0.0031	0.0030
Lipczak	0.0228	0.0311	0.0344	0.0357	0.0367	0.0370	0.0367	0.0359	0.0351	0.0342	0.0333	0.0324	0.0316	0.0310	0.0303	0.0296	0.0290	0.0283	0.0277	0.0270
TitleToTitle	0.0703	0.0869	0.0909	0.0895	0.0868	0.0831	0.0793	0.0752	0.0714	0.0678	0.0645	0.0615	0.0589	0.0564	0.0542	0.0522	0.0503	0.0485	0.0470	0.0455
Tag popularity multiplier	0.0227	0.0313	0.0344	0.0357	0.0367	0.0369	0.0366	0.0358	0.0350	0.0341	0.0332	0.0323	0.0315	0.0309	0.0302	0.0295	0.0289	0.0282	0.0276	0.0270
TitleToTitle & popularity multiplier	0.0703	0.0869	0.0909	0.0895	0.0868	0.0831	0.0793	0.0752	0.0714	0.0678	0.0645	0.0615	0.0589	0.0564	0.0542	0.0522	0.0503	0.0485	0.0470	0.0455

Table D.7: F1 results on MovieLens 10M dataset.
Green = best result per algorithm, orange = best overall result.

MovieLens 10M consists of 81167 posts. 80% = 64934 posts serve as training data and 20% = 16233 posts serve as test data. Lipczak's system trains on 80% of the training data = 51947 posts and learns parameters on 20% of the training data = 12987 posts. The recommenders produced the following statistics:

Title recommender Learned on 177460 titlewords of which 7415 unique, 3.4162 average titlewords per post (3.3593 unique)

TitleToTag recommender Learned on 7415 unique titlewords and 64707 related tagwords = 8.73 related tagwords per titleword

TagToTag recommender Learned on 100659 tagwords, of which 10130 unique and 120360 relations = 11.88 related tagwords per tagword

TitleToTitle recommender Learned on 177460 titlewords of which 7415 unique and 63388 relations = 8.55 related titlewords per titleword

Fold	TitleToTag-TagToTag
1	0.95
2	0.81
3	1
4	0.69
5	0.81

(a) Lipczak's content-based merge coefficients

Fold	Title-TitleToTitle	TitleToTag-TagToTag
1	0.99	0.99
2	0.99	0.97
3	0.99	1
4	0.99	0.99
5	1	0.81

(b) Merge coefficients with TitleToTitle

Fold	TitleToTag-TagToTag
1	0.95
2	0.81
3	1
4	0.69
5	0.81

(c) Merge coefficients with popularity multiplication

Fold	Title-TitleToTitle	TitleToTag-TagToTag with popularity multiplier
1	0.99	0.99
2	0.99	0.97
3	0.99	1
4	0.99	0.99
5	1	0.81

(d) Merge coefficients with TitleToTitle and popularity multiplication

Table D.8: Merge coefficients of different systems on MovieLens 10M