INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Sugestões de Tags para Digg

Bruno Miguel Ferreira Santos de Oliveira

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente:	Prof. Ernesto Morgado
Vogal:	Prof. Arlindo Oliveira
Orientadora:	Prof. Helena Sofia Pinto
Co-orientador:	Prof. Pável Calado

Setembro de 2007

Acknowledgements

First, I would like to thank my supervisors, Prof. Helena Sofia Pinto and Prof. Pável Calado, whose wisdom and ever-present guidance helped me throughout this thesis work. Their expertise and enthusiasm kept me motivated and compelled me to try and do things the right way.

I also would like to thank my INESC-ID colleagues: Pedro Andrade, Marta Kalupeteka and Pedro Cuba for sharing with me both their joys and misfortunes and aiding me whenever they could; Rudi Araújo and Peyman Sazedj for being such good advisers, providing insightful and valuable suggestions.

Next I would like to thank all my fellows with whom I spent some of the most intense times on this 5-year course. Together we made it through this journey safe and sound.

Last, but certainly not least, I must express my gratitude to my close family and dearest friends. They managed to keep me company along this winding road, always supportive of my decisions and temper. Also, when the going got tough, they were the ones who reached out for me and helped keeping my head up.

Abstract

Social tagging systems are becoming increasingly popular, mainly because of their ease of use and low entry barriers. The subject is fairly new, and although there is not much literature on the topic, there are quite a few systems available. However, despite its popularity, tagging is usually a manual process. When publishing on a social tagging system, the user is asked for the tags he wishes to assign to the resource being made available. These keywords help users organizing their resources and allow for better retrieval of resources, when querying the system. A tag suggester is a system that proposes a set of relevant keywords for the user to choose from, making his task of organizing resources, easier.

In this Thesis, a text-based tag suggesting system, entitled Tess, is presented. It is based on machine learning and information retrieval techniques. The data collection needed by the algorithm is gathered from *Digg* and the suggester fits this site's hierarchy of topics. When queried, the system processes the input document and uses the files on the corpus to find relevant keywords to be recommended.

Tess was evaluated by a group of users. Some statistical measures were applied to infer the system's performance and the preliminary results looked promising. In average, over half of the suggested tags are considered relevant by at least one user. Nonetheless, there is still room for improvement. Hence, some ideas on how to further develop the system are suggested.

Keywords

Tagging, Tag Suggesting Systems, *Digg*, *Folksonomies*, Information Management Systems

Resumo

Os sistemas de *tagging* sociais estão a tornar-se cada vez mais populares, devido, em grande parte, à sua facilidade de utilização. Este assunto é relativamente recente e, apesar de não haver muita literatura, existem já bastantes sistemas. Todavia, apesar de popular, o processo de *tagging* é, normalmente, manual. Quando um utilizador publica um recurso, são-lhe pedidas as *tags* para serem associadas ao recurso. Estas palavras-chave ajudam os utilizadores a organizarem os recursos e facilitam a procura. A tarefa de um sistema de sugestão de *tags* é escolher e propôr um conjunto de palavras que se adequem ao recurso a ser publicado, para que o utilizador possa escolher as que lhe interessam, facilitando a sua tarefa de organizar os recursos.

Nesta Tese, apresenta-se um sistema de sugestão de *tags*, denominado Tess. É baseado em técnicas de *machine learning* e recuperação de informação. A colecção de dados, necessária à execução do algoritmo, é recolhida a partir do *Digg*, pelo que, o Tess segue a mesma hierarquia de tópicos. Quando o sistema recebe um documento, este é processado e, juntamente com os documentos no corpus, são extraídas palavras relevantes para serem depois sugeridas.

O sistema foi avaliado por um grupo de utilizadores. Foram aplicadas algumas medidas estatísticas para aferir a sua qualidade. Os primeiros resultados foram promissores. Em média, mais de metade das *tags* sugeridas são consideradas relevantes por, pelo menos, um utilizador. Contudo, o sistema ainda pode ser melhorado. Como tal, apresentam-se algumas ideias para melhorar a qualidade do sistema.

Palavras-chave

Tagging, Sistemas de Sugestão de Tags, *Digg*, *Folksonomies*, Sistemas de Gestão de Informação

Table of Contents

List of Figures

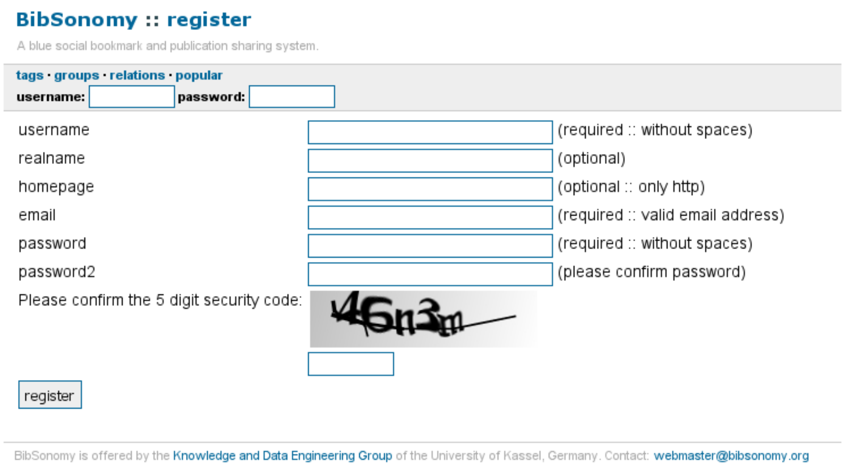
List of Tables

1 Introduction

1.1 Context

The shape of the World Wide Web has been changing, as websites tend to be more interactive and user-centric than in the past. Some of the most successful systems on this “Web 2.0” generation are the social tagging ones [Tre06]. These systems have their content managed by the users, who publish resources along with sets of tags. Tags are simply keywords to help organize and describe each resource. Since there is no predefined vocabulary or semantics, the user can choose whichever string he prefers to use as a tag. Typically there is no way for the user to establish relations between tags or resources, which means they will not form hierarchies. Resources vary from system to system. For instance, *flickr* [1] is a website for photos sharing, *YouTube.com* [2] for videos, *del.icio.us* [3] for bookmarks and *WordPress.com* [4] for *blog* entries.

Social tagging systems share a common set of features: resource publishing, browsing and searching. The bookmarking and publication sharing system *BibSonomy* [5] will be used to illustrate each feature.



BibSonomy :: register

A blue social bookmark and publication sharing system.

tags · groups · relations · popular

username: password:

username	(required :: without spaces)
realname	(optional)
homepage	(optional :: only http)
email	(required :: valid email address)
password	(required :: without spaces)
password2	(please confirm password)
Please confirm the 5 digit security code:	

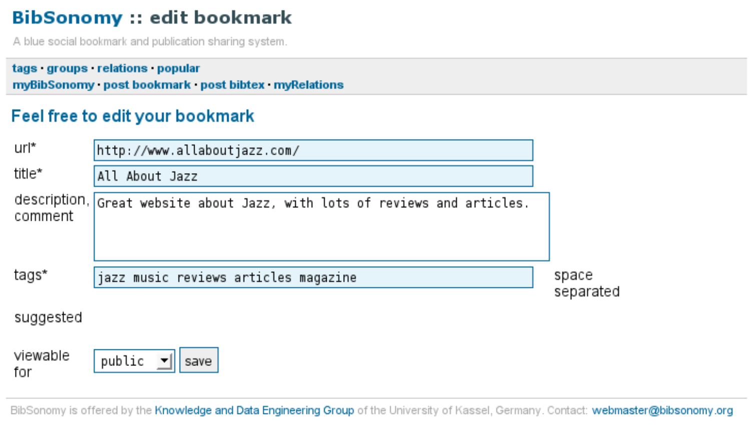
register

BibSonomy is offered by the Knowledge and Data Engineering Group of the University of Kassel, Germany. Contact: webmaster@bibsonomy.org

Figure 1.1: Creating a new account on *BibSonomy*

To take full advantage of the system, a user must create an account, providing a username and a password, Fig. 1.1. The registered user can publish any resource he wishes, as long as the system requirements are fulfilled. For instance, in *YouTube* resources must be videos up to 10 minutes long and less than 100 MB. Typically, the user uploads the resource and associates a set of tags to be attached to it, as well as any additional information the system may require, Fig. 1.2. Some systems suggest a set of tags for the user to choose from, making his task easier.

[1] http://www.flickr.com/
[2] http://www.youtube.com/
[3] http://del.icio.us/
[4] http://wordpress.com/
[5] http://www.bibsonomy.org/


BibSonomy :: edit bookmark

A blue social bookmark and publication sharing system.

tags · groups · relations · popular
myBibSonomy · post bookmark · post bibtex · myRelations

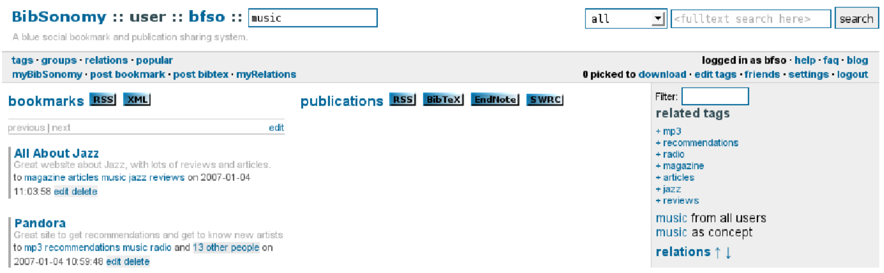
Feel free to edit your bookmark

url*	http://www.allaboutjazz.com/
title*	All About Jazz
description, comment	Great website about Jazz, with lots of reviews and articles.
tags*	jazz music reviews articles magazine — space separated
suggested	
viewable for	public — save

BibSonomy is offered by the Knowledge and Data Engineering Group of the University of Kassel, Germany. Contact: webmaster@bibsonomy.org

Figure 1.2: Posting a bookmark

BibSonomy and *del.icio.us*, for example, present the most popular tags associated to a resource when it was already tagged by other users and is being resubmitted.


BibSonomy :: user :: bfso :: music
all
<fulltext search here>
search

A blue social bookmark and publication sharing system.

tags · groups · relations · popular
myBibSonomy · post bookmark · post bibtex · myRelations
logged in as bfso · help · faq · blog
0 picked to download · edit tags · friends · settings · logout

bookmarks RSS XML — publications RSS BibTeX EndNote SWRC

previous | next — edit

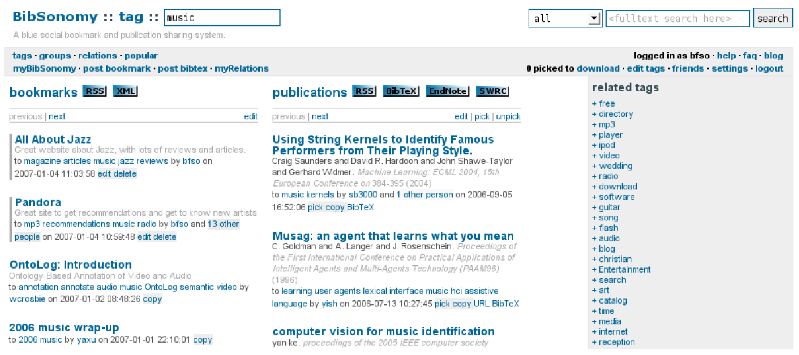
All About Jazz
Great website about Jazz, with lots of reviews and articles.
to magazine articles music jazz reviews on 2007-01-04 11:03:58 edit delete

Pandora
Great site to get recommendations and get to know new artists
to mp3 recommendations music radio and 13 other people on 2007-01-04 10:59:48 edit delete

Filter:
related tags
+ mp3
+ recommendations
+ radio
+ magazine
+ articles
+ jazz
+ reviews
music from all users
music as concept
relations ↑ ↓

Figure 1.3: Displaying user's resources with a given tag

Typically there are two ways of browsing: through tags or through users. Browsing through tags enables the user to find the resources that were assigned a certain tag. In Fig. 1.3 the system is displaying all the resources that *bfso* assigned the tag *music*, while in Fig. 1.4 it is showing all the resources in the system associated with that tag.


BibSonomy :: tag :: music
all
<fulltext search here>
search

A blue social bookmark and publication sharing system.

tags · groups · relations · popular
myBibSonomy · post bookmark · post bibtex · myRelations
logged in as bfso · help · faq · blog
0 picked to download · edit tags · friends · settings · logout

bookmarks RSS XML

previous | next — edit

All About Jazz
Great website about Jazz, with lots of reviews and articles.
to magazine articles music jazz reviews by bfso on 2007-01-04 11:03:58 edit delete

Pandora
Great site to get recommendations and get to know new artists
to mp3 recommendations music radio by bfso and 13 other people on 2007-01-04 10:59:48 edit delete

OntoLog: Introduction
Ontology-Based Annotation of Video and Audio
to annotation annotate audio music OntoLog semantic video by wcrosbie on 2007-01-02 08:48:26 copy

2006 music wrap-up
to 2006 music by yaxu on 2007-01-01 22:10:01 copy

publications RSS BibTeX EndNote SWRC

previous | next — edit | pick | unpick

Using String Kernels to Identify Famous Performers from Their Playing Style.
Craig Saunders and David R. Hardoon and John Shawe-Taylor and Gerhard Widmer. *Machine Learning: ECML 2004, 15th European Conference on* 384-395 (2004)
to music kernels by sb3000 and 1 other person on 2006-09-05 16:52:06 pick copy BibTeX

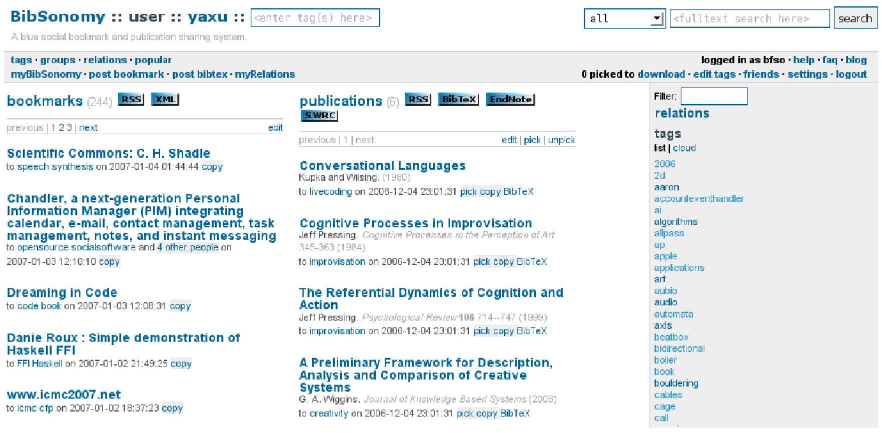
Musag: an agent that learns what you mean
C. Goldman and A. Langer and J. Rosenschein. *Proceedings of the First International Conference on Practical Applications of Intelligent Agents and Multi-Agents Technology (PAAM96)* (1996)
to learning user agents lexical interface music hci assistive language by yish on 2006-07-13 10:27:45 pick copy URL BibTeX

computer vision for music identification
yan ke. *proceedings of the 2005 IEEE computer society*

related tags
+ free
+ directory
+ mp3
+ player
+ ipod
+ video
+ wedding
+ radio
+ download
+ software
+ guitar
+ song
+ flash
+ audio
+ blog
+ christian
+ Entertainment
+ search
+ art
+ catalog
+ time
+ media
+ internet
+ reception

Figure 1.4: Displaying the system's resources with a given tag

Since every resource has its owner, there is also the ability to browse through users, that is, to find all resources and tags published by a certain user, Fig. 1.5. Generally users combine both browsing modes. While navigating through the resources in the system, one can find other users that may have similar tastes and resources of interest. Most social tagging systems have a searching feature. Depending on the available information about users and resources, it can proceed in various ways, . For instance, in *flickr*, users are searched by their interests or personal info whereas photos are searched by tags or description.


BibSonomy :: user :: yaxu ::
<enter tag(s) here>
all
<fulltext search here>
search
A blue social bookmark and publication sharing system.
tags · groups · relations · popular
myBibSonomy · post bookmark · post bibtex · myRelations
logged in as bfso · help · faq · blog
0 picked to download · edit tags · friends · settings · logout
bookmarks (244) RSS XML
previous | 1 2 3 | next
edit
publications (6) RSS BibTeX EndNote SWRC
previous | 1 | next
edit | pick | unpick
Filter:
relations
tags
list | cloud

Scientific Commons: C. H. Shadle
to speech synthesis on 2007-01-04 01:44:44 copy

Chandler, a next-generation Personal Information Manager (PIM) integrating calendar, e-mail, contact management, task management, notes, and instant messaging
to opensource socialsoftware and 4 other people on 2007-01-03 12:10:10 copy

Dreaming in Code
to code book on 2007-01-03 12:08:31 copy

Danie Roux : Simple demonstration of Haskell FFI
to FFI Haskell on 2007-01-02 21:49:25 copy

www.icmc2007.net
to icmc cfp on 2007-01-02 18:37:23 copy

Conversational Languages
Kupka and Wilsing. (1980)
to livecoding on 2006-12-04 23:01:31 pick copy BibTeX

Cognitive Processes in Improvisation
Jeff Pressing. *Cognitive Processes in the Perception of Art* 345-363 (1984)
to improvisation on 2006-12-04 23:01:31 pick copy BibTeX

The Referential Dynamics of Cognition and Action
Jeff Pressing. *Psychological Review* **106** 714--747 (1999)
to improvisation on 2006-12-04 23:01:31 pick copy BibTeX

A Preliminary Framework for Description, Analysis and Comparison of Creative Systems
G. A. Wiggins. *Journal of Knowledge Based Systems* (2006)
to creativity on 2006-12-04 23:01:31 pick copy BibTeX

2006
2d
aaron
accounteventhandler
ai
algorithms
allpass
ap
apple
applications
art
aubio
audio
automata
axis
beatbox
bidirectional
boiler
book
bouldering
cables
cage
call

Figure 1.5: User's area showing their resources and tags

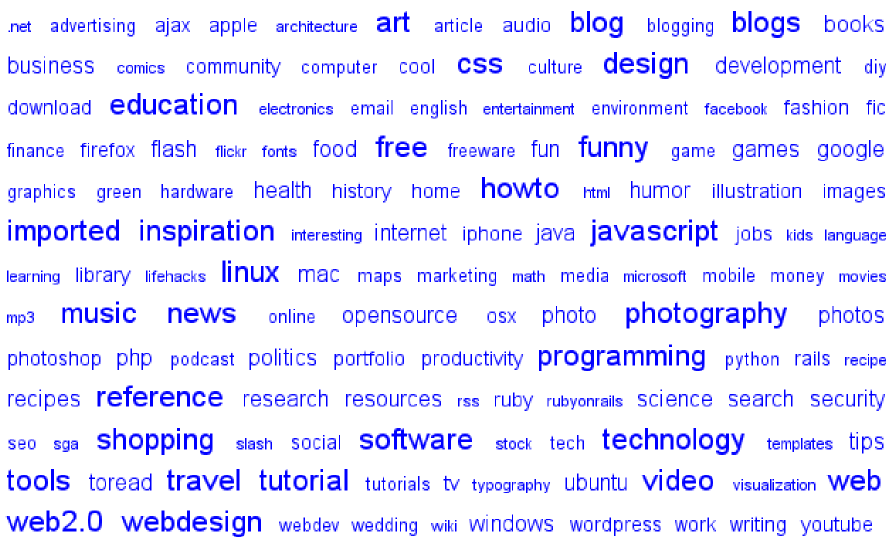
Along with the features presented so far, some websites display other interesting features as well. Tag cloud is one of such popular features. It consists of a visual representation of the most frequently used tags on a website. Usually the tags are presented in alphabetic order and are given different emphasis according to their popularity. Examples of websites using tag clouds include *flickr*, *del.icio.us*, *BibSonomy* and *WordPress.com*. Another interesting feature that may be found in some social tagging systems is the creation of groups. Generally, groups are organized in categories and the resources shared among the group are all on the same thematics. In *flickr*, for instance, there are several interest groups, where photos are published and discussed among its members. A somewhat less common characteristic is the distinction between users. Some websites allow users to choose among different account types, each one with their own features. In *YouTube* one may choose to sign up as a regular user, a musician or a comedian, whereas in *Last.fm* [6] one may register as a regular user or as a label or artist.

One may ask what made tagging so popular and interesting. First of all, it is very easy. The concept is simple and requires no special skills. Also, it gives users a sense of freedom and power, as there are no restrictions to the resources organization and description. Another appealing feature that can be found in these systems is the ability to share resources. People like showing their work, participating in communities of interest and discovering new things within their interests. They can do all this in most social tagging systems. Furthermore, these systems are constantly evolving and reflect the trends going on in society. This is actually a consequence of

[6] http://www.last.fm/

having its contents managed by the users. When dealing with non-textual resources, such as photographs or videos, tagging is a rather straightforward technique to help improve search and information retrieval.

As one would expect, tagging has its drawbacks as well, mostly due to the lack of structure and unrestricted vocabulary. Fig. 1.6, a real example taken from *del.icio.us*, will be be used to illustrate the some of these drawbacks. One problem is polysemy, which consists on one word having multiple meanings. For example, the word ‘ruby, might refer to a computer programming language or to a red gemstone. Another is synonymy, which occurs when different words have the same meaning. For instance the words ‘fun’ and ‘humor’, may have the same meaning. A third problem arises when users employ some tags to help organize their agendas, instead of describing the resource. The tag ‘toread’ is probably assigned when the user does not have the opportunity to read the resource, acting more as a reminder. The fourth problem is due to the fact that these systems are likely to be used by people from all over the world, so, probably, more than



.net advertising ajax apple architecture art article audio blog blogging blogs books business comics community computer cool css culture design development diy download education electronics email english entertainment environment facebook fashion fic finance firefox flash flickr fonts food free freeware fun funny game games google graphics green hardware health history home howto html humor illustration images imported inspiration interesting internet iphone java javascript jobs kids language learning library lifehacks linux mac maps marketing math media microsoft mobile money movies mp3 music news online opensource osx photo photography photos photoshop php podcast politics portfolio productivity programming python rails recipe recipes reference research resources rss ruby rubyonrails science search security seo sga shopping slash social software stock tech technology templates tips tools toread travel tutorial tutorials tv typography ubuntu video visualization web web2.0 webdesign webdev wedding wiki windows wordpress work writing youtube

Figure 1.6: *del.icio.us* tag cloud with the most popular tags on July 20, 2007

one idiom will be used simultaneously. All these issues make it difficult to infer the tags semantics in a precise way. Since there aren’t relations between tags, it is also difficult to find out which ones have broader or narrower meanings or which ones are related [MNBD06].

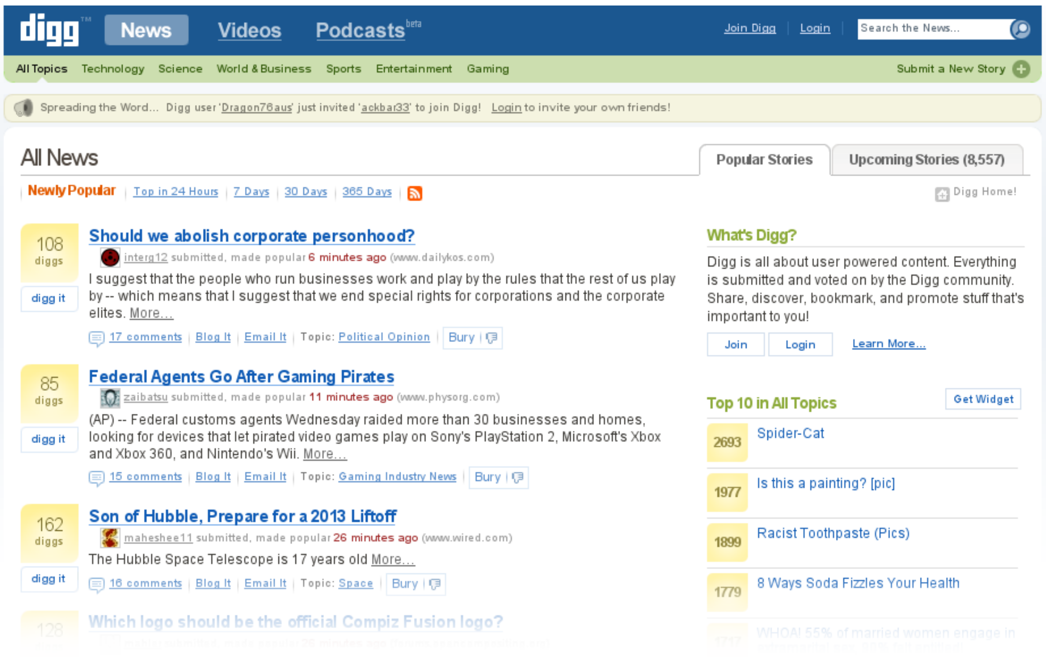
The use of tags as metadata – data about data – whose semantics emerges from its community is called “folksonomy”. Thomas Vander Wal was the first to use this word in a discussion on an information architecture mailing list [Mat04]. It is a combination of the words “folk” and “taxonomy”.

Different properties can be found in folksonomies, depending on the number of users tagging the same item. As Quintarelli suggests [Qui05], there are two types of folksonomies:

- Broad – many users tag the same item using their own preferences and vocabulary. These systems can be used to investigate trends by analyzing tag popularity over time. (e.g. *del.icio.us*)
- Narrow – small number of users tag the same item. In such cases, tagging is used only as a means to help users organize and retrieve their resources. (e.g. *flickr*)

Folksonomy enthusiasts claim that the semantics emerging from a community is more accurate than when it is defined by a committee or a single person. They also say that, since content creation is decentralized, its description be decentralized as well. Furthermore, they argue that hierarchical classification schemes are too rigid to classify web data [BM06a].

Digg [7] is one of the most popular social networking systems. It was launched in November 2004 as a news sharing website, democratically managed by its user community. Initially, it was mainly focused on technology news [Wat06]. Today, *Digg* has a wider scope, with several distinct categories, such as science and sports and two new sections dedicated to videos and podcasts.



Spreading the Word... Digg user 'Dragon76aus' just invited 'ackbar33' to join Digg! Login to invite your own friends!

All News

108 diggs — Should we abolish corporate personhood?
interg12 submitted, made popular 6 minutes ago (www.dailykos.com)
I suggest that the people who run businesses work and play by the rules that the rest of us play by -- which means that I suggest that we end special rights for corporations and the corporate elites. More...
17 comments | Blog It | Email It | Topic: Political Opinion

85 diggs — Federal Agents Go After Gaming Pirates
zaibatsu submitted, made popular 11 minutes ago (www.physorg.com)
(AP) -- Federal customs agents Wednesday raided more than 30 businesses and homes, looking for devices that let pirated video games play on Sony's PlayStation 2, Microsoft's Xbox and Xbox 360, and Nintendo's Wii. More...
15 comments | Blog It | Email It | Topic: Gaming Industry News

162 diggs — Son of Hubble, Prepare for a 2013 Liftoff
maheshee11 submitted, made popular 26 minutes ago (www.wired.com)
The Hubble Space Telescope is 17 years old More...
16 comments | Blog It | Email It | Topic: Space

128 — Which logo should be the official Compiz Fusion logo?

What's Digg?
Digg is all about user powered content. Everything is submitted and voted on by the Digg community. Share, discover, bookmark, and promote stuff that's important to you!

Top 10 in All Topics

2693	Spider-Cat
1977	Is this a painting? [pic]
1899	Racist Toothpaste (Pics)
1779	8 Ways Soda Fizzles Your Health
	WHOA! 55% of married women engage in

Figure 1.7: *Digg*'s front page, displaying stories from different sections

The content of this website is chosen by its community, who acts as the editor board of a newspaper. A registered user can submit a story by providing a title, its link, a brief description of the story and the most appropriate topic, of all the ones available on the website. Unlike many popular social networking systems, *Digg* does not include a tagging mechanism. When a new story is submitted, it is placed in the "Upcoming Stories" queue, awaiting for votes. If a story receives enough positive votes – or, using the website terminology, 'diggs' – it gets promoted to

[7] http://www.digg.com/

the front page, as can be seen in Fig. 1.7. There is also the possibility to vote down – or, 'bury'. Thus, if a story is reported as duplicate, spam, off-topic, inaccurate or unsatisfactory or if it simply does not get enough 'diggs' it disappears.

Besides submitting stories, registered users can also discuss with each other by commenting on the stories, keep track of the stories they 'dugg', published or commented, build a friends network, email stories, track *Digg*'s activity in real-time and block users, just to name a few interesting features [Lay06]. As for unregistered users, only the basic features are available. They may browse through the website, read the stories and comments and search the news. Registering is free and requires only a user name, a valid email address, a password, and the birth date.

The process of promotion and decay of user submitted stories is run by a proprietary algorithm, kept undisclosed in order to avoid abuse and exploit. It is an automatic technique based on several factors, such as the number of 'diggs', time of day and category [Dig07]. As one of the system's most influential component, the algorithm is continuously enhanced and improved. The main concerns regarding its improvement, are to stimulate the diversity of subjects and ensure a democratic process on story promotion [Pon07].

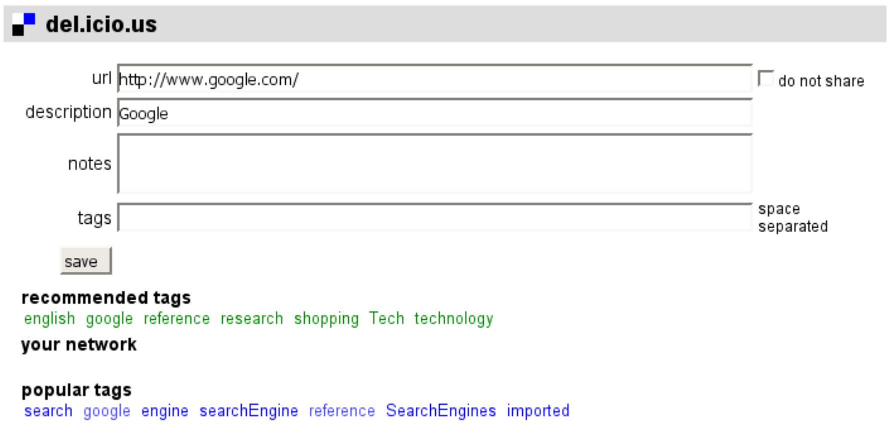
Other popular content sharing websites, similar, in some way, to *Digg* include *Netscape* [8], *reddit* [9], *StumpleUpon* [10] and *Slashdot* [11]. According to the web traffic tracker *Alexa* [12], *Digg* has the highest traffic rank, making it, supposedly, the most popular among the websites of this category.

1.2 Goals of this Thesis

Despite its growing popularity and inclusion in many successful websites, tagging is still a manual process. Hence, the user is the only responsible for assigning keywords to resources. Some users may feel it is rather obtrusive to be prompted by the system, in order to provide a set of tags, when submitting a resource and may not be willing to engage in such a chore. To overcome this shortfall, some websites have a built-in tag suggesting mechanism, which suggests a set of words for the user to choose from and assign to the resources. The core concept is simple and has been used commercially in many recommender systems, included in websites such as *Amazon.com* [13] or *IMDb* [14]. When browsing through the website, the user is presented with related or similar resources, hoping he will find interest in the suggested items [CLA+03]. Correspondingly, in tag suggesting systems, the idea is to provide the set of tags that best relates to the resource being published, leaving the final decision, of choosing which should be assigned, to the user.

[8] http://www.netscape.com/
[9] http://reddit.com/
[10] http://www.stumbleupon.com/
[11] http://slashdot.org/
[12] http://www.alexa.com/
[13] http://www.amazon.com
[14] http://www.imdb.com

An example of tag suggesting can be seen in *del.icio.us*. The system will propose a set of tags if the resource has already been submitted and tagged by other users. In such cases, it gathers the most popular tags among the ones assigned to the given resource and displays them to the user. Additionally, it recommends tags the user has already employed and were also used by other users on the resource being submitted, as can be seen in Fig. 1.8. The user may choose from the suggested tags and also write down some new ones, if none of the recommended tags suits


del.icio.us

url	http://www.google.com/	do not share
description	Google	
notes		
tags		space separated

save

recommended tags
english google reference research shopping Tech technology

your network

popular tags
search google engine searchEngine reference SearchEngines imported

Figure 1.8: Tag suggestions when bookmarking *Google*'s homepage on *del.icio.us.*

his needs [del07]. Despite useful in several situations, it does not work under certain conditions. If the user is the first to submit a particular resource, he will not be given any suggestion. Also, this kind of method would not work in a website where each resource is unique. For instance, in a 'blogging' website, each 'blog' entry, at the time of publishing, is presumably new to the world, making it impossible for the system to retrieve any tag.

Considering the importance and, simultaneously, the lack of sophisticated and flexible tag suggesters, this thesis work aims at developing such a system. Due to the small number of publications found on this topic, the system was built from scratch. It is focused on text-based documents only and the need for tags or users was avoided. In this case, the tags suggested by the system are extracted from documents, by combining and adapting standard information retrieval and clustering techniques, but not from a set of previously employed tags. Hence, the system is likely to keep up with the changes and evolution of the communities, their interests and resources. Since there is no dependency on previously used tags, the system may come across new words and find them meaningful enough to be considered as tags and, consequently, present them to the users. Furthermore, this tag suggester is able to deal with resources that have never been tagged before, as long as they are text based, which makes it adaptable to a wide range of systems.

The ability to aid users by suggesting appropriate tags they could not think of at the moment, was another important goal taken into consideration. In fact, recommender systems are intended to make people's tasks easier. When asking an expert for guidance, one expects to be enlightened by his knowledge on the subject and, often, obtain the advice one could not find out by oneself. Ideally, the same should happen with recommender systems.

Albeit *Digg* is used as subject and data source on this research, the resulting suggester should be adaptable to other websites. The main reasons for choosing *Digg* over similar websites are its popularity and, therefore, the existence of a very large community of active users, who make sure the website is kept up to date and contribute to its voluminous stories database; the use of a hierarchy of topics to enforce the site's organization, making it easier to know each story subject and group them accordingly; the fact that most resources are text based and, thus, apposite to the tag suggesting system being developed. The fact that *Digg* does not support tagging is not very relevant to this work, since the idea is to develop a tag suggester for any website with similar characteristics. In fact, *Digg* was chosen mainly because of its content attributes, as described above.

In short, this thesis work main goal is to design and build a tag recommender system, using text based resources as its data source. It should be able to evolve along with the system on which it is built-in and be easily adapted to different websites. Also, unlike many tag suggesters, it is supposed to work with resources never tagged before. *Digg* website was used as the data source due to its popularity, variety of subjects and large amount of resources.

1.3 Report's Layout

This Thesis report has a total of five chapters. The first chapter is an introduction that contextualizes this project and depicts the set of goals to be achieved. Next, an overview on some of the techniques found useful, regarding the improvement of folksonomies, is presented. The two subsequent chapters describe the system developed over this thesis work and its performance evaluation. Lastly, the conclusions and further improvements on the tag suggesting system are presented.

The second chapter brings insight on some of the most relevant work related, in some way, to this thesis. For organization purposes, the techniques found to be pertinent were grouped according to their intent. Resource organization improvement is the first topic presented in this chapter. The following section focuses on search engine enhancements. To finish the chapter, some techniques about the relation between tags and resources are described.

In the third chapter, the tag suggesting system developed on this thesis is presented. Firstly, its main features and general purpose are outlined. Then, the architecture and each of the components of the system are thoroughly described, as well as the supporting algorithms.

Chapter 4 reports the evaluation proceedings and the gathered results. Also, a critical analysis on the data and the system is presented.

Final remarks and future work can be found in the last chapter.

2 Related Work

Most social tagging systems are rather simple and, thus, do not exploit folksonomies to its full potential. The following sections present some of the techniques that can be used to improve folksonomies. They were grouped into categories according to their most relevant features.

2.1 Resource Organization

Folksonomies lack of structure and organization makes some user tasks very difficult. Although its chaotic and uncontrolled nature induces serendipitous discoveries, it becomes hard to find specific resources [Mat04]. To overcome this issue, some authors tried to cluster related resources and place them in broader categories. Additionally, some tried to automatically infer a topical hierarchy, identical to those used in many websites, as a means of helping users navigate through its content.

2.1.1 Grouping Similar Resources

Wu *et al.* present a probabilistic method, based on co-occurrences, to derive emergent semantics on folksonomies [WZY06]. They claim that tags are usually semantically related when used on the same or related resources. The same idea applies to users and resources, that is, users may share common interests if they use many semantically related tags and resources tagged by users with similar interests. This can be represented using frequencies of co-occurrences between users, resources and tags.

The algorithm proposed to group similar resources is as follows. Each entity (user, tag or resource) is represented in a conceptual space, which corresponds to a multi-dimensional vector where each dimension represents a knowledge category. The more related an entity is to a certain category, the higher will be the score in the corresponding vector dimension. Ambiguous tags will have high scores in different dimensions.

In order to determine the number of dimensions and the conceptual space, they used a Separable Mixture Model, a statistical framework for modeling and predicting co-occurrences of events [HP98]. The process goes as follows:

1. Choose the number of knowledge categories, that is, the number of dimensions of the conceptual space.
2. For each dimension d_α, measure how strongly it relates with:
    - the user – $p(u_i|d_\alpha)$.
    - the resource – $p(r_j|d_\alpha)$.
    - the tag – $p(t_k|d_\alpha)$.

3. Compute the probability of co-occurrence:

$$p(u_i, r_j, t_k) = \sum_{\alpha=1}^{D} p(d_\alpha)p(u_i|d_\alpha)p(r_j|d_\alpha)p(t_k|d_\alpha) \tag{2.1}$$

4. Apply Expectation-Maximization (EM) method to (2.2) to estimate the required probabilities:

$$\sum_{ijk} n_{ijk} \log(p(u_i, r_j, t_k)) \tag{2.2}$$

- Iterate E-step and M-step until convergence on a local maximum.
- E-step will determine the indicator matrix I with dimensions $C \times D$, where C is the number of triples <user, resource, tag>. $I_{c\alpha}$ is the probability of assigning record c to dimension α:

$$I_{c\alpha}^{(t)} = \frac{p(d_\alpha)^{(t)} p(u_{u(c)}|d_\alpha)^{(t)} p(t_{t(c)}|d_\alpha)^{(t)} p(r_{r(c)}|d_\alpha)^{(t)}}{\sum_{\alpha=1}^{D} p(d_\alpha)^{(t)} p(u_{u(c)}|d_\alpha)^{(t)} p(t_{t(c)}|d_\alpha)^{(t)} p(r_{r(c)}|d_\alpha)^{(t)}} \tag{2.3}$$

- M-step uses the following equations:

$$p(d_\alpha)^{(t+1)} = \frac{\sum_{c=1}^{C} I_{c\alpha}^{(t)}}{C} \tag{2.4}$$

$$p(u_i|d_\alpha)^{(t+1)} = \frac{\sum_{c:u(c)=i} I_{c\alpha}^{(t)}}{\sum_{c=1}^{C} I_{c\alpha}^{(t)}} \tag{2.5}$$

$$p(r_j|d_\alpha)^{(t+1)} = \frac{\sum_{c:r(c)=j} I_{c\alpha}^{(t)}}{\sum_{c=1}^{C} I_{c\alpha}^{(t)}} \tag{2.6}$$

$$p(t_k|d_\alpha)^{(t+1)} = \frac{\sum_{c:t(c)=k} I_{c\alpha}^{(t)}}{\sum_{c=1}^{C} I_{c\alpha}^{(t)}} \tag{2.7}$$

According to the authors, the algorithm successfully grouped tags according to their semantics, as depicted in Tab. 2.1. Moreover, it identified tags ambiguity and synonymy, the users interests

1	java programming Java eclipse software
2	css CSS web design webdesign
3	blog blogs design weblogs weblog
4	music mp3 audio Music copyright
5	search google web Google tools
6	python programming Python web software
7	rss RSS blog syndication blogs
8	games fun flash game Games
9	gtd productivity GTD lifehacks organization
10	programming perl development books Programming

Table 2.1: Top 5 tags in 10 randomly selected dimensions from the set of 40 [WZY06]

and the resources semantics. Despite these favorable results, there are some drawbacks. A lot of experimentation is needed in order to find the appropriate parameters – number of dimensions D and number of iteration steps on the EM method. Also, the whole conceptual space has to be re-calculated whenever the annotation set changes.

A different approach, to serve a somewhat different purpose, is presented in [WZM06]. Folksonomies lack some fundamental features to be used as the supporting infrastructure of knowledge management activities, such as the identification of topical communities of users and/or resources and ways to find high quality sources. In order to solve these issues, two algorithms are proposed.

To identify communities in social tagging systems, they propose the use of a spectral method, as follows:

1. Represent the folksonomy as a graph in an adjacency matrix:
    - Resources, tags and users are represented as nodes.
    - Links are established between:
        - tags and associated resources.
        - users and the tags they created or accessed.
        - users and the resources they accessed.
2. Apply singular value decomposition to the matrix.
3. Highest singular values produce the main topics of interest. Left singular vectors indicate the most important users on the topic, and right singular vectors indicate the key documents.

To identify high quality documents and experts, the HITS (Hypertext Induced Topic Selection) [BYRN99] algorithm was adapted, resulting in the following procedure:

1. Collect all the documents tagged by the given keyword.
2. Expand the document set by including all tags associated to its documents, the documents under those tags and the users who accessed the tags.
3. Build a graph to represent this set as done in the previous algorithm – matrix T'.
4. Capture the hyperlink structure among documents in a matrix A'. A_{ij} indicates whether there is a link from node i to node j.
5. Compute the principal eigenvectors in $A'T'A'$ to find the experts.
6. Compute the principal eigenvectors in $A'A'T'$ to find the high quality documents.

The authors were still experimenting different ways to assign weights to the links and combinations with hyperlink analysis to improve the algorithms. Nevertheless, the preliminary results looked promising.

The work presented by Brooks and Montanez in [BM06b] is more of a study on the effectiveness of tagging as a means to categorize documents. The idea is to determine how useful are tags in clustering similar articles. Intuitively, a group of documents sharing the same tag is more similar than a group of randomly selected articles. In order to find out whether this hypothesis is correct or not, they clustered a set of documents from a social tagging system and then compared them with an upper and lower bound. They used the following procedure:

1. Fetch the 350 most popular tags from *Technorati* [1].
2. For each tag, retrieve the 250 most recent articles that had been assigned this tag.
3. Pre-process the text – remove HTML tags and stop words.
4. Compute the $tfidf$ score [BYRN99] for each remaining word:
$$tfidf(word) = termFreq(word) * \log\left(\frac{|corpus|}{DocFreq(word)}\right) \tag{2.8}$$
5. Build a cluster for each tag, containing the vectors of all documents having that tag.
6. Calculate the average pairwise cosine similarity of all articles in each cluster C:
$$aveSim(C) = \frac{\sum_{a,b\in C,a\neq b} cosSim(a,b)}{\sum_{i=1}^{|C|-1} i} \tag{2.9}$$
where
$$cosSim(A,B) = \frac{\sum_{w\in A\cup B} A[w]B[w]}{\sqrt{\sum_{v\in A} A[v]^2 \sum_{v\in B} B[v]^2}} \tag{2.10}$$
A and B being the term vectors, that is, vectors with all the words appearing on their corresponding documents.

To evaluate the performance of this methodology, two bounds were defined. The lower one was obtained by randomly clustering articles. The upper one was determined by clustering articles reported as being similar by *Google News* [2]. They concluded that the process of tagging, groups articles into broad categories, making it difficult for a user to find more specific topics.

2.1.2 Inducing Hierarchies

The authors of [WZM06] state that hierarchies are very useful for navigation and as means to assist in keyword-based search. They further claim that it is possible to use tags to induce such structures, using a hierarchical clustering algorithm. This article describes an agglomerative clustering approach:

- Identify the set of documents and associated tags for which the hierarchy will be generated.
- Construct a document-tag matrix A, where $A_{ij} = 1$ if document i was assigned the tag j.
- Build a tag-tag matrix T using a thesaurus. The purpose of this matrix is to capture the semantic similarities between tags.
- Multiply the two matrices $A \times T$. Each document i is now represented by its corresponding row vector A_i.
- Apply a hierarchical clustering technique to the matrix.
- Label each category by extracting keywords from the tags of all the documents in the new category.

[1] http://www.technorati.com

[2] http://news.google.com/

The method described can produce different results, depending on the clustering and labeling methods used. The authors state that using Jaccard similarity measure, which indicates the degree of overlap between two sets [EM02], combined with the average linkage method produces the best results. Different labeling methods were still being evaluated. In order to evaluate the algorithm, several hierarchies were generated using different methods (including manually) and presented to human evaluators. According to the article, the one generated by their method obtained the highest classification.

Likewise, Brooks and Montanez [BM06b] present an agglomerative clustering technique as a means to generate hierarchies. They propose the following procedure:

1. Fetch the 250 most popular tags from *Technorati*.
2. For each tag, retrieve 20 articles that had been assigned this tag and form a cluster.
3. Apply the following procedure until there is only one cluster:
   (a) Compare each cluster to every other cluster using the pairwise cosine similarity, as described in the work by the same authors in Section 2.1.1.
   (b) Replace the two closest-similarity clusters by a new abstract cluster, which consists in merging them.
   (c) The new abstract cluster is assigned an abstract tag, which is the conjunction of the clusters tags.
4. Construct the tree according to the order in which the clusters were grouped.

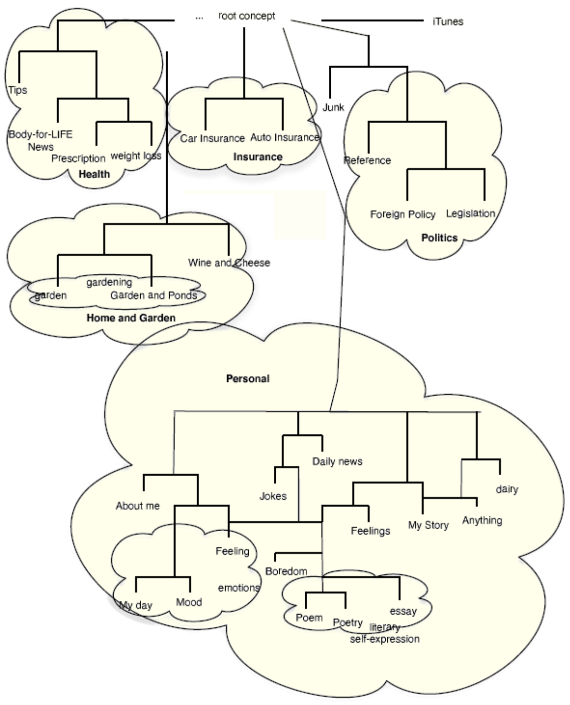


Figure 2.1: Subset of the hierarchy generated using *Technorati*'s tags [BM06b]

According to the authors, the generated hierarchy resembles hand-built taxonomies. A portion of the resulting tree is represented in Fig. 2.1. Although the cluster merging was based on document similarity, that is, the tags were only used to build the original clusters, tags fit into a topical hierarchy. This remark supports the idea of being possible to automatically generate a category hierarchy, which can be used to organize resources.

2.2 Search Engine Improvement

Searching is one of the main features in social tagging systems. It is the only mechanism, other than browsing, that enables users to find specific resources. The folksonomies structure, that is, information about users, tags, resources and the relations among them, can be used to improve the performance of the searching mechanism. Standard information retrieval techniques can be adapted to take advantage of such structure.

The work proposed in [WZY06], besides presenting a method to group similar resources, Section 2.1.1, can also be used to improve search. In this case, the conceptual space is used to develop an intelligent semantic search engine. An incremental approach is followed, starting with a basic search model and improving it gradually, by extending its features. The four models described in the article are presented next.

The basic search model supports single tag queries and ranks the resources using no information from the user. In other words, when searching, the user uses a tag as query and the system makes no use of his personal information. The basic search uses the following probability equation:

$$p(r|t) = \sum_{\alpha=1}^{D} p(r|d_\alpha)p(d_\alpha|t) \tag{2.11}$$

Where r is the resource, t the tag and d a knowledge category. The retrieved resources will be ranked according to the conditional probability $p(r|t)$.

This model can be further improved by also returning semantically related resources that were not assigned the query tag.

Besides resources and tags, the conceptual space also contains information on their users. It is possible to integrate such information in the search engine so that its results can be personalized and more accurate. This leads to a new formula:

$$p(r|u,t) \sim \sum_{\alpha}^{D} p(r|d_\alpha)p(u|d_\alpha)p(t|d_\alpha)p(d_\alpha) \tag{2.12}$$

The models described so far supported single tag queries only. To overcome this shortcoming, they were extended so that the queries can be the Boolean combination of multiple tags or words appearing on the resources. Queries are thus represented in the following way:

$$q = \{k_1 : a_1, k_2 : a_2, \ldots, k_p : a_p\} \tag{2.13}$$

Where k is a tag or a word to be found on a resource and a is the weight of the component in the query. The following equations [BYRN99] are used to compute the query probability, depending on whether it is an *and* or an *or* query:

$$p(q_{and}|d) = 1 - \left[\frac{a_1^p(1 - p(k_1|d))^p + \ldots + a_n^p(1 - p(k_n|d))^p}{a_1^p + \ldots + a_n^p}\right]^{\frac{1}{p}} \tag{2.14}$$

$$p(q_{or}|d) = 1 - \left[\frac{a_1^p p(k_1|d)^p + \ldots + a_n^p p(k_n|d)^p}{a_1^p + \ldots + a_n^p}\right]^{\frac{1}{p}} \tag{2.15}$$

When the query component is a word w appearing on a resource the following probability formula is used:

$$p(w|d_\alpha) = \sum_{j=1}^{N} p(w|r_j)p(r_j|d_\alpha) \tag{2.16}$$

The language model proposed by Jelinek and Mercer in [JM80], is used to compute $p(w|r_j)$. Combining all the search models leads to the equation:

$$p(r|u, q) \sim \sum_{\alpha=1}^{D} p(r|d_\alpha)p(u|d_\alpha)p(q|d_\alpha)p(d_\alpha) \tag{2.17}$$

The different search models are independent and can be combined in different ways. Thus, distinct search engines can be obtained. The authors claim that the ability to discover semantically related resources is the most important feature of their models.

A ranking mechanism is also described by Hotho *et al.* in [HJSS06]. The authors claim that social tagging systems have limited search support, since searching can only be done over the tags and the resource descriptions. Besides, no ranking is done over the retrieved resources. Most systems present the results in reverse chronological order instead of displaying them by relevance. To improve this rather weak feature they start by defining a formal model to represent folksonomies and then use it to develop their algorithms for resource ranking.

A folksonomy is described as a tuple $\mathbb{F} := (U, T, R, Y, \prec)$, where:

- U, T and R are the sets of users, tags and resources, respectively.
- Y is the relation between them: $Y \subseteq U \times T \times R$.
- $\prec$ is a subtag/supertag-relation defined by the user: $\prec \subseteq U \times T \times T$.

For simplicity, the subtag/supertag relation is not used in the article. Thus, the folksonomy becomes a quadruple $\mathbb{F} := (U, T, R, Y)$.

The first algorithm was adapted from PageRank [BYRN99], which simulates a user navigating in the Web who, occasionally, jumps to a random page or follows a random hyperlink. The main idea is that a resource tagged with important tags by important users becomes important itself. The apply the algorithm proceed as follows:

1. Convert the folksonomy into an undirected weighted graph: $\mathbb{G}_{\mathbb{F}} = (V, E)$
   - The set of nodes V is the disjoint union of the sets of users, tags and resources: $V = U \dot\cup T \dot\cup R$

- The edges are defined according to the co-occurrences of tags and users:

  $E = \{\{u,t\},\{t,r\},\{u,r\}|(u,t,r) \in Y\}$

  Each edge is weighted with the number of times they co-occur in the folksonomy.

2. Spread the weight by applying (2.18) until convergence, where $\vec{w}$ represents the edge weight, A is the row-stochastic version of the adjacency matrix of $\mathbb{G}_{\mathbb{F}}$, $\vec{p}$ is a vector to express user preferences, $\alpha, \beta, \gamma \in [0,1]$ are constants such that $\alpha + \beta + \gamma = 1$, where α regulates the speed of convergence and β and γ regulate the influence of $\vec{p}$.

$$\vec{w} \leftarrow \alpha\vec{w} + \beta A\vec{w} + \gamma\vec{p} \tag{2.18}$$

3. Retrieve all web pages and compute the inverted index.
4. When searching for the term t, each retrieved resource v is ranked using the formula:

$$tf(t) \cdot \vec{w}[v] \tag{2.19}$$

where $tf(t)$ is the term's frequency in page v and $\vec{w}[v]$ is the resource's weight as computed before.

The weight method was further improved, by comparing the resulting rankings with and without the preference vector $\vec{p}$, which resulted in a new algorithm, *FolkRank*:

1. Use $\vec{p}$ to determine the topic by assigning a high value to one or more tags and/or one or more users and/or one or more resources.
2. Compute $\vec{w_0}$ using (2.18) with $\beta = 1, \alpha = \gamma = 0$.
3. Compute $\vec{w_1}$ using (2.18) with $\beta < 1$.
4. The final weight vector is computed by:

$$\vec{w} := \vec{w_1} - \vec{w_0} \tag{2.20}$$

#	**adapted PageRank**	**FolkRank**
1	boomerang	boomerang
2	shop	shop
3	lang:de	lang:de
4	software	wood
5	java	kassel
6	programming	construction
7	web	plans
8	reference	injuries
9	system:unfiled	pitching
10	wood	rdf

Table 2.2: Comparison of the top 10 resulting tags for the query-tag "boomerang"

The *FolkRank* algorithm was developed because the *adapted PageRank* did not perform very well. The experiments described in the article confirm that *FolkRank* obtains better results, as depicted in Tab. 2.2. Also, when a topical community is small, a single user can have a very strong influence on the results and, thus, provides bridges to other topics. Furthermore, the *FolkRank* algorithm can be used to recommend tags or resources of interest to a given user.

2.3 Tag Suggesting

To the best of our knowledge, only one tag recommendation system has been proposed in the literature so far: the *AutoTag* system [Mis06]. Using the most common recommender systems as an analogy, which suggest products to their users, in the *AutoTag* system, blog posts take the role of users and tags are seen as the products the user might be interested in. Similar tags are assumed to be suitable to similar blog posts. For that reason, the system suggests tags for a blog entry by examining the tags assigned to similar entries. The method described next is illustrated in Fig. 2.2.

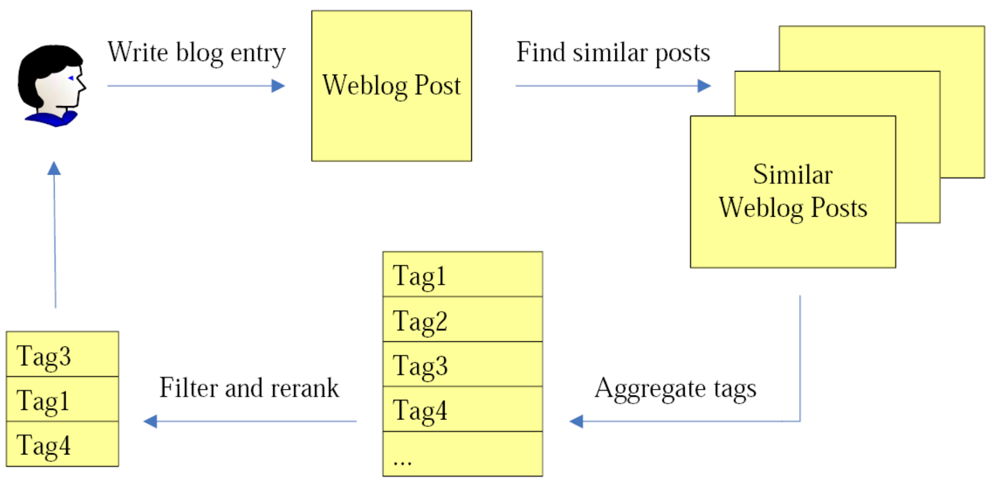


Figure 2.2: AutoTag's methodology [Mis06]

1. The user publishes a new article.
2. The system finds and collects the most similar entries by generating a query from the new article and submitting it to an information retrieval engine, which indexes all the documents in the collection. This query consists of a set of the most distinctive terms in the post, which are obtained by comparing the post's vocabulary with the general vocabulary on the corpus using standard comparison methods.
3. The system aggregates all tags assigned to the retrieved entries and rank them according to their frequency in the top posts.

4. The system filters and re-ranks the set of tags by boosting the ones that were previously used by the blogger using a constant factor.
5. The system presents the top tags to the user.

The experiments described in the article show good results, in spite of the relatively small corpus used. However, there is still room for improvement. Other query generation methods might be used, tag aggregation could be enhanced and different filtering and re-ranking techniques might provide better results. This system was specifically designed to be used with blogs. Resources containing few or no words (such as in *flickr*) will probably lead to a poor performance.

Although the authors of [BM06b] expressed their willingness to develop a tag suggester as well, they have not released any article about it yet. There was also a webservice called *Tagyu* which, when given an article, suggested a set of tags, but was shut down in August 2006 [3]. Also, no publications, regarding this system, were found. Therefore, it was impossible to analyze *Tagyu*.

If one looks at tags as being categories, the automated tagging problem may fit a well-known research area in computer science: text categorization (TC). Therefore, although not focused specifically on folksonomies, we find this area relevant to our problems and worth to be mentioned in this section. There has been a lot of research on the topic in the past years. In 2002, Fabrizio Sebastiani published a comprehensive survey about the work done until then [Seb02]. We present a brief overview of its main ideas, with not as much detail, as they are fully explained in his article.

TC consists in determining the categories of natural language texts using a set of predefined categories. A classifier is a program responsible for this labeling activity. Several methodologies can be used to build a classifier. Nowadays, machine learning (ML) techniques are the most popular ones. Basically, they are used to automatically induce a text classifier by learning through a set of preclassified documents. One of their main advantages is the accuracy that can be obtained by the generated classifiers. For this reason, the techniques presented in Sebastiani's paper follow the ML paradigm.

The classifiers are further divided into single-label and multilabel, depending on the number of categories they are supposed to assign to each document. There is also binary categorization, which is a special case of single-label in which each document is either assigned a category or its complement. The single-label case is more general than the multilabel and thus, a multilabel classification problem can be divided in various binary classification problems, one for each category (if the categories are stochastically independent, which is usually true). According to this description, a multilabel classifier would be the most suitable approach to the task of automatically tagging resources, since they are usually assigned more than one tag.

Another significant difference between classifiers is in the way they are expected to decide on the categories. One may want it to choose one or more categories and assign them to the

[3]See the author's blog entry at http://kalsey.com/2006/08/and_thanks_for_all_the_fish/

document, or to simply return a list of suitable categories sorted by their rank and let a human take the final decision. The first alternative would correspond to an automatic tagging process and the second one could be used in a recommender system as the one described in the beginning of this section.

When using ML techniques one must define two fundamental sets of documents: a training and validation set of documents, from which the classifier will be induced, and a test set of documents to compare the classifier decisions to the ones done by the expert who classified the documents. As to the classifier algorithms, several methods have been developed: probabilistic, decision tree, decision rule, regression, on-line methods, the Rocchio method, neural networks, example-based, support vector machines, among others. All these methods are thoroughly described in Sebastiani's survey. Although it is hard to predict which ones would work well in the tagging process, support vector machines, example-based methods, regression methods and boosting-based classifier committees seem to be the most accurate in TC.

There seem to be some limitations when using this kind of systems as tag suggesters or automatic taggers. First of all, they depend on a finite, predefined set of categories (or tags, in this case), which means they will not be able to automatically keep up with the users vocabulary evolution. Keeping the training set, which contains documents and categories, up to date would hardly solve this problem, since the system itself would influence the way these documents were classified. Furthermore, these systems cannot be used with non-textual resources, as they need to analyze their content, which must be text.

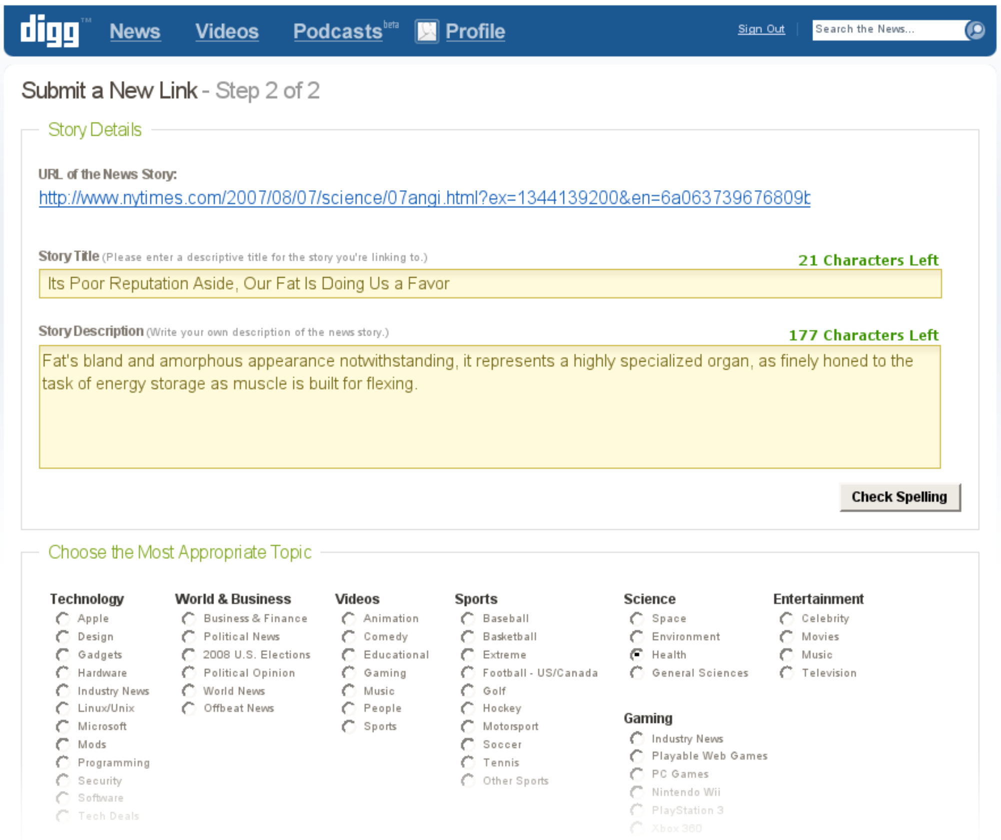
3 Tess: Tag Suggester

3.1 Overview

Tess is a tag suggesting system, that is, a program to recommend tags. When given a resource, it presents a set of keywords for the user to choose from.

This system was developed as a possible add-on to *Digg*, a content sharing website managed by its user community. Hence, Tess can use *Digg*'s taxonomy, that is, its set of topics and their hierarchical relations.

Since Tess only works with text-based resources, some of *Digg*'s sections were excluded from the very beginning, such as Videos and Podcasts. The main reason for limiting the scope of this work is the fact that this kind of resources requires different and complex techniques. The unpopular and newer topics, added in subsequent versions of the website, were ignored as well: the former ones for having few stories, making it difficult for the system to achieve satisfying results, and the latter ones for being unavailable when this project started.



Submit a New Link - Step 2 of 2

Story Details

URL of the News Story:
http://www.nytimes.com/2007/08/07/science/07angi.html?ex=1344139200&en=6a063739676809b

Story Title (Please enter a descriptive title for the story you're linking to.) 21 Characters Left
Its Poor Reputation Aside, Our Fat Is Doing Us a Favor

Story Description (Write your own description of the news story.) 177 Characters Left
Fat's bland and amorphous appearance notwithstanding, it represents a highly specialized organ, as finely honed to the task of energy storage as muscle is built for flexing.

Choose the Most Appropriate Topic

Technology: Apple, Design, Gadgets, Hardware, Industry News, Linux/Unix, Microsoft, Mods, Programming, Security, Software, Tech Deals

World & Business: Business & Finance, Political News, 2008 U.S. Elections, Political Opinion, World News, Offbeat News

Videos: Animation, Comedy, Educational, Gaming, Music, People, Sports

Sports: Baseball, Basketball, Extreme, Football - US/Canada, Golf, Hockey, Motorsport, Soccer, Tennis, Other Sports

Science: Space, Environment, [✓ Health], General Sciences

Gaming: Industry News, Playable Web Games, PC Games, Nintendo Wii, PlayStation 3, Xbox 360

Entertainment: Celebrity, Movies, Music, Television

Figure 3.1: Submitting a health article on *Digg*

When publishing a story on *Digg*, as illustrated on Fig. 3.1, the user chooses the most appropriate topic, inserts a title, a description and a link to the story itself. Likewise, Tess takes as input the story's document, its title, description and topic. The last parameter is optional, as it can be determined by a Tess unit, the categorizer. When given a resource, the categorizer suggests one of *Digg*'s topics. There are various built-in methods for selecting tags. For that reason, several sets of tags may be presented as output, depending on the combination of the methods chosen when configuring the system.

3.2 System's Architecture

There are three major components in Tess architecture, as depicted in Fig. 3.2. The first two are engaged in data acquisition and transformation, and the last one is responsible for choosing the words to be suggested as tags.

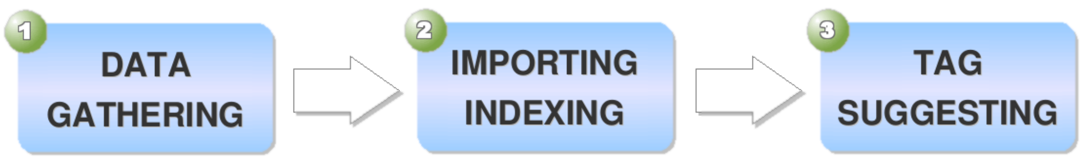


Figure 3.2: Three main components in Tess

In the first stage, *Digg*'s top stories are collected, as well as their additional information, such as the number of votes, title, submitter's name, and so on. This additional information is stored in a database, whereas the stories themselves are saved as files in a designated directory. Fig. 3.3 illustrates this component.

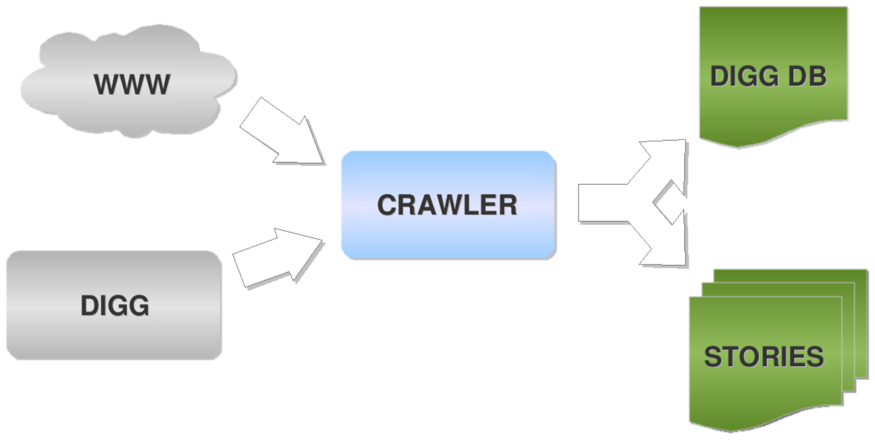


Figure 3.3: Tess first stage: data gathering

Next, the gathered documents and information are processed according to the method represented in Fig. 3.4, in order to create the data structures required in the third stage.

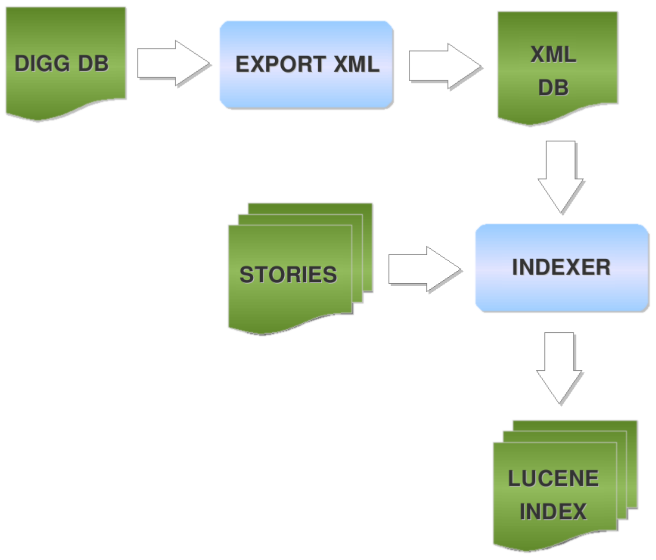


Figure 3.4: Tess second stage: data importing and indexing

Lastly, the main component does the most noticeable job on this system. Using the data collected and prepared in the previous stages and also the user query – document and its additional information – it recommends several sets of tags, according to Tess configuration. Optionally, the system's categorizer may be used to assign a topic to the document. This process is illustrated in Fig. 3.5

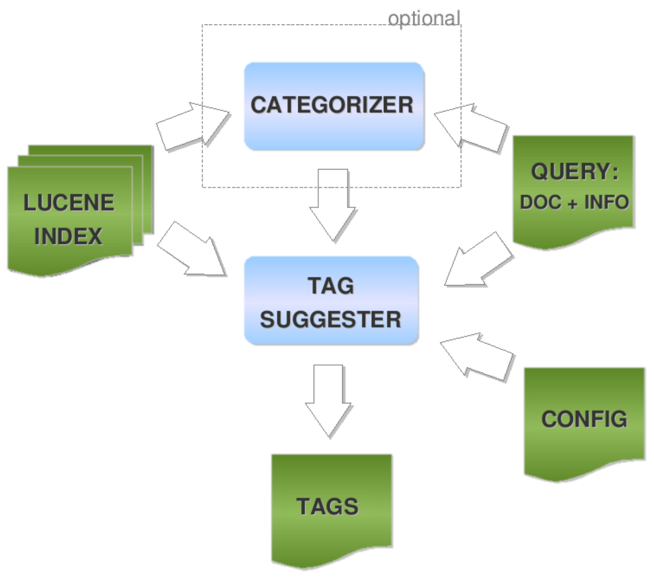


Figure 3.5: Tess third stage: tag suggesting

Each one of these components will be further detailed in forthcoming sections.

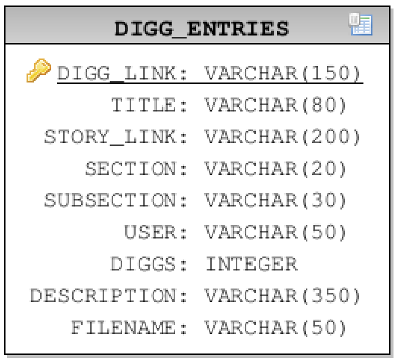
3.3 Building the Corpus

Before starting to collect data, *Digg* was analyzed in order to find out which stories should be retrieved. Besides the Videos and Podcasts sections, which were excluded from the start due to the non-textual nature of its resources, the ‘Offbeat News’ topic was ignored as well, since there is not a definite subject among such stories. Furthermore, since *Digg* is continuously being updated, some of the topics that can be found now in the website, were not available when the crawler was being developed. Another important requirement, defined in advance, was to retrieve top stories only. To get to the front page, a story needs to be voted by many users. For that reason, the stories are more likely to be of interest and were probably filed under the appropriate topics. That is, selecting top stories only, ensures a higher degree of quality.

This analysis on *Digg* also allowed for choosing the information to be retrieved and stored persistently. The following data about the stories was found to be relevant:

- Title
- Description
- Story’s link
- Topic
- Subtopic
- User name
- Number of votes

Other information, such as which users commented or blogged each story, or each user’s friend network, could be gathered from the website, but would require a lot more requests to *Digg*’s server and extra storage room, and would not be of interest to the system being developed. Besides, it can be later retrieved, if needed. The retrieved data is stored on a SQLite database, except for story documents, which are stored as HTML files in a designated directory on the filesystem. As displayed in Fig. 3.6, the database contains all the additional information of the



```
         DIGG_ENTRIES
   DIGG_LINK: VARCHAR(150)
       TITLE: VARCHAR(80)
  STORY_LINK: VARCHAR(200)
     SECTION: VARCHAR(20)
  SUBSECTION: VARCHAR(30)
        USER: VARCHAR(50)
       DIGGS: INTEGER
 DESCRIPTION: VARCHAR(350)
    FILENAME: VARCHAR(50)
```

Figure 3.6: Crawler’s database schema

stories, plus the document's file path. Hence, the database has all the information a program might need when using the collected data. The SQLite format was chosen due to its ease of use and to the small dimensions of this database, which contains only one table. Also, SQLite databases are very portable, since each one is stored in a single file and requires no setup or authentication to use it. Despite its simplicity, SQLite supports the use of SQL queries, as well as inserts and deletes, making it easy to store and search the data.

Digg has a hierarchy of topics, where each topic has its own front page, that is, a place where popular stories are placed. These sections of the website work as stacks, that is, each time a story is promoted, it is placed on top of its topic's front page. As newer entries are added to the top of the page, older ones go down and, when they reach the bottom, they move to the subsequent page. In other words, each topic's page displays 15 stories. When a new story gets popular, it is placed on top of the first page, while the story closest to the bottom moves to the top of the second page, and so on.

Since all the information needed for each story can be found on its topic's pages, a single request to the server, that is, a page download, may yield up to 15 stories. This strategy is friendlier to *Digg*'s web server, as it reduces considerably the amount of HTTP requests. For that reason, the crawler gets the stories from the topic's 'Newly Popular' pages only, doing it in depth, throughout the topic's pages until it reaches the defined limit, or until it founds a story that already is on the database. In this latter case, given the stack-like behavior of such sections, there is no need to go further, as all the subsequent stories should already be in the database.

When our crawler was written, there was no API available to retrieve data from *Digg*. For this reason, no generic crawler was used. Instead, the crawler built during this thesis work, fits *Digg*'s structure and is able to get the information needed, using regular expressions to parse the downloaded pages. The crawler was written in Python since it is a language that makes it fast to program, although interpreted, it is efficient enough, there are many libraries available due to its popularity and it can be easily adapted to a script-style of programming. However, any change in the website's HTML templates or URL hierarchy may jeopardize the program's functioning. To avoid making many requests in a short period of time and, thus, flooding *Digg*'s webserver, the crawler waits 60 seconds between requests.

A somewhat frequent phenomena, called 'Digg effect', occurs when many users try to download a webpage simultaneously, causing the server where it is stored to become very slow, or even fail, for not being able to cope with the traffic peak. This usually happens when a story gets very popular, which draws even more users to the server at issue. In such cases, the crawler discards the story, as it is unable to retrieve it. To deal with such issues and others alike, the crawler uses GNU Wget to download the stories.

The crawler was scheduled to work around 11 o'clock everyday, using cron, a scheduling service included in most Linux distributions. The data was gathered between November 1, 2006 and March 29, 2007. After those five months, the corpus was frozen in order to get stable results and start analyzing the tag extraction methods that were being developed. Otherwise, when upgrading such methods, one would not know if the changes in the results were caused by the upgrades, or by changes in the corpus.

Additionally some charts were generated, using PyChart library, for a better understanding on *Digg*'s community habits. As Fig. 3.7 illustrates, the number of stories reaching their topics front

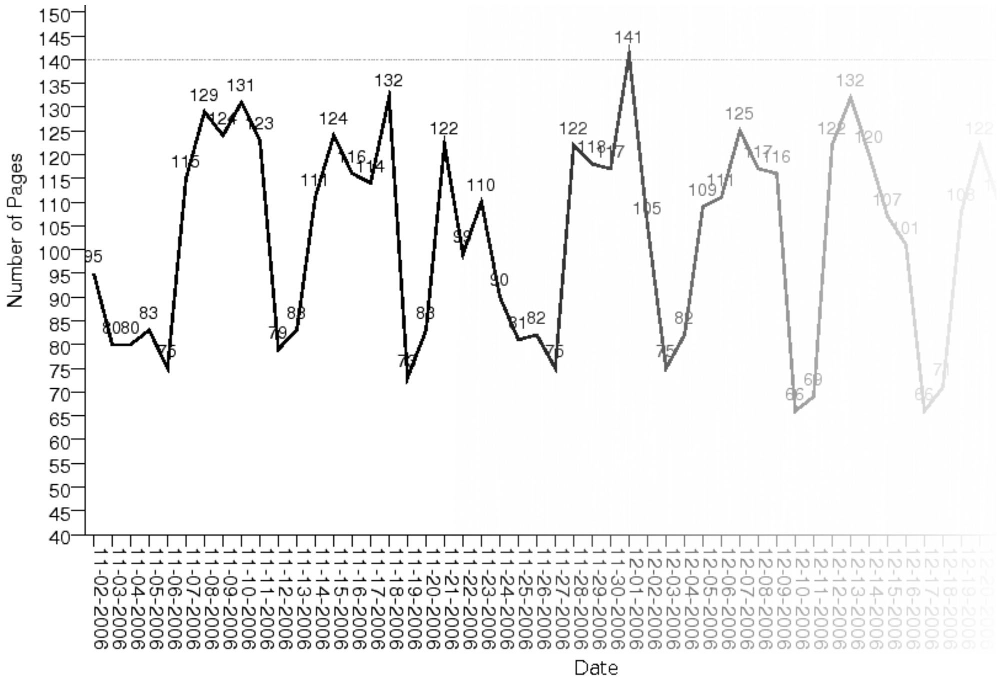


Figure 3.7: Number of stories reaching the front page a day

pages varies from day to day. It oscillates between 60 and 140, approximately, giving a rough average of 100 popular stories a day. This chart is furthered detailed by grouping pages by their main topics, as seen in Fig. 3.8, which shows the last 7 days of November 2006. The main topics

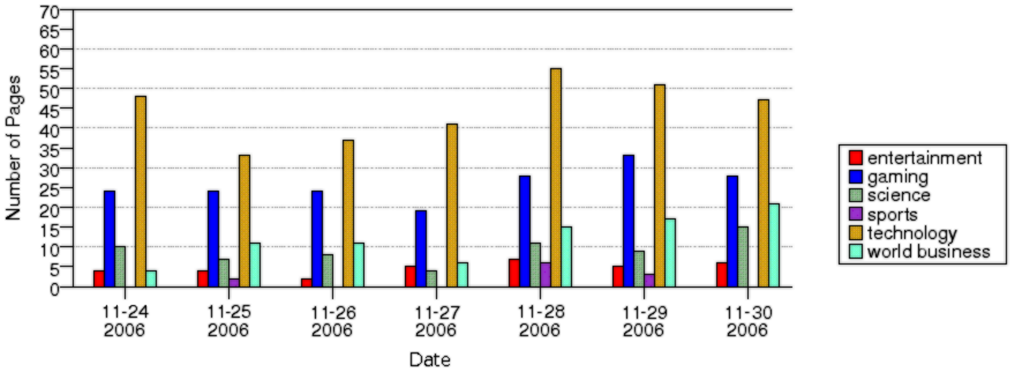


Figure 3.8: Popular stories a day, grouped by section

of interest are, undoubtedly, technology and gaming, whereas the most unpopular one is sports. The corpus status, depicted in Fig. 3.9, reinforces this inference. There are, in fact, much more

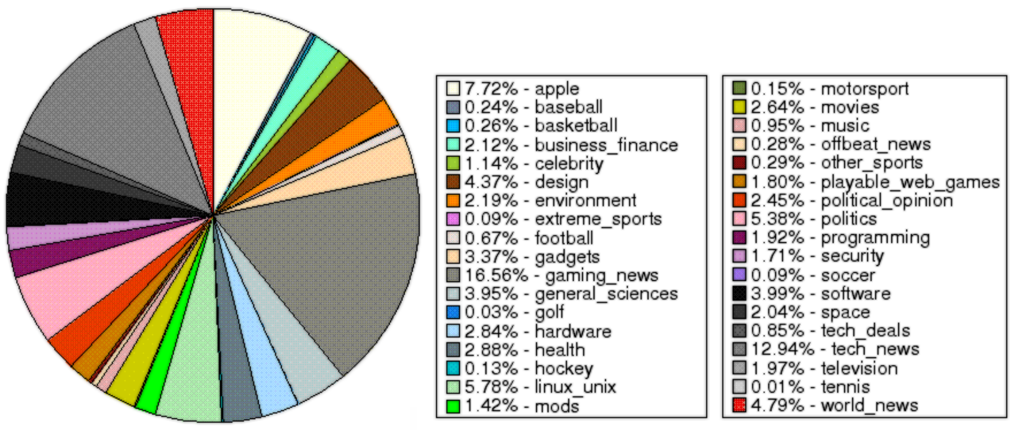


Figure 3.9: Corpus status when the crawling process was terminated

stories on technology and gaming then in any other sections. Comparing the top popular topics with the unpopular ones, in Tab. 3.1, *Digg*'s audience may be characterized mainly as tech-savvy, though it seems to be broadening its subjects of interest, as topics like Politics or World News are becoming popular. Contrarily, the Sports section does not seem to be getting much support. As a consequence of this asymmetry on the user's interest, the amount of stories on each topic varies greatly, making story distribution in the corpus rather irregular.

#	**Popular**	**Unpopular**
1	Gaming News	Tennis
2	Tech News	Golf
3	Apple	Soccer
4	Linux/Unix	Extreme Sports
5	Politics	Hockey
6	World News	Motorsport
7	Design	Baseball
8	Software	Basketball
9	General Sciences	Offbeat News
10	Health	Other Sports

Table 3.1: Top 10 Popular vs. Unpopular topics on *Digg*

3.4 Processing and Indexing Data

The information gathered using the previously described component should be processed to a meaningful representation. Otherwise it would not be very useful. Furthermore, the large amount of data calls for efficient structures, in order to speed up and reduce the memory print of the computer processes using such information.

A standard way to represent these helpful structures is through indexes, which are appropriate when using large and semi-static collections, as in this case [BYRN99]. There are several indexing techniques. The one used in this Thesis data is the inverted index mechanism. An inverted index is a data structure composed of words and occurrences and is intended to help the searching tasks. Basically, all the words found in the document collection are placed in the vocabulary set. Each word on the vocabulary set points to a list indicating the documents that contain it and, optionally, the positions where the word appears in the document. This structure is illustrated in Fig. 3.10. By using this mechanism, when a query is submitted to the search engine, there is no need to scan all the documents to find out whether they have the word or not. Instead, all it takes is a search on the index's vocabulary set, which can be easily optimized. There are some variations on this technique that may be useful, depending on the context, but they will not be approached here, as they are outside the scope of this text.

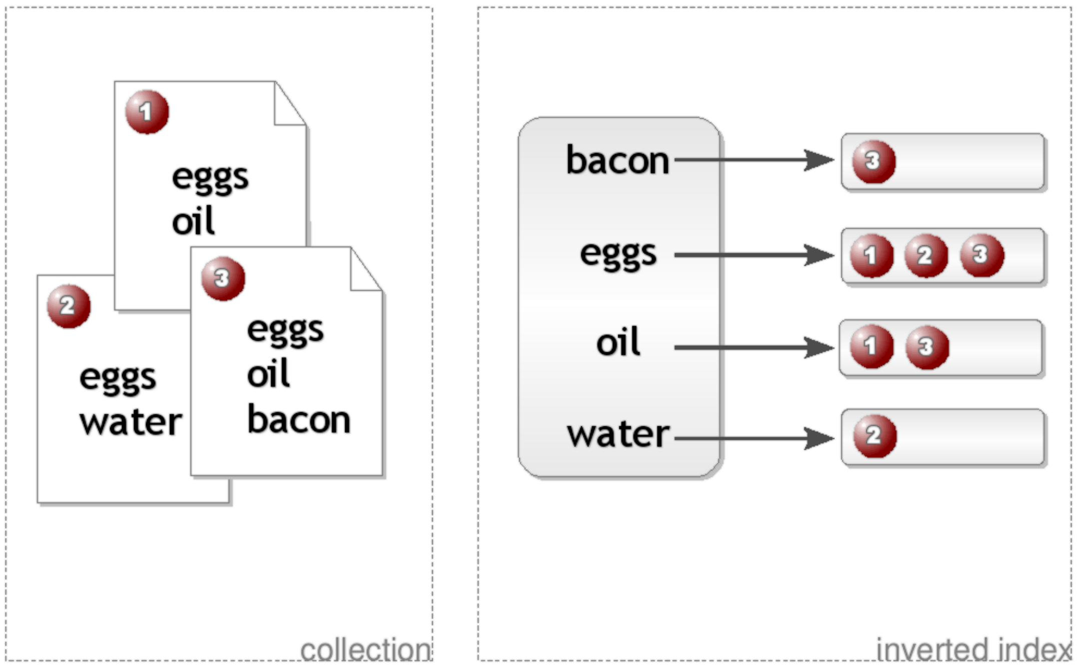


Figure 3.10: An inverted index of a collection of egg recipes

In what comes to document representation, it is common procedure, in Information Retrieval, to apply the vector model [BYRN99]. Using this classic model, documents are represented as vectors in a t-dimensional space, that is, a space with as many dimensions as the number of distinct terms in the document collection. Each position of a document vector contains the weight of its corresponding term. A term's weight can be computed using different term-weighting schemes. The *tf-idf* is the best known scheme and uses the following formula:

$$w_{ij} = tf_{ij} \times idf_{ij} \tag{3.1}$$

Where, tf_{ij} is the frequency of the term i in document j, that is, the number of times the term i is found in document j; idf_i is the inverse document frequency and is given by:

$$idf_{ij} = \log \frac{N}{n_i} \tag{3.2}$$

Where N is the total number of documents in the collection and n_i is the number of documents containing the term i.

Considering the need for efficient structures, just as described, as well as an appropriate data representation, Apache's Lucene was chosen to be used in this project. Lucene is a well-known framework for information retrieval, developed and maintained by Apache and used in many popular systems, such as *Furl* [1], *Simpy* [2], *Wikipedia* [3], among others [Luc07]. This framework was adopted for being well-known, efficient, frequently upgraded, having a good documentation and support for most of the features needed for this project.

However, Lucene is a framework for Java and, although there are ports to other programming languages, including Python, they may take some time to be upgraded and keep up with the releases of the Java version and may have bugs and poor documentation. For this reason, the indexer as well as the main component of the system, were written in Java.

Since there is few support for SQLite in Java and, at the time, the SQLite version used to store *Digg*'s info was not yet ported to Java, the database contents had to be exported to an intermediate format that could be interpreted by a Java program. For that reason, an exporting feature was added to the crawler. When requested to export the database, the crawler prints an Extensible Markup Language (XML) file containing all the stored data, as shown in Fig. 3.11.

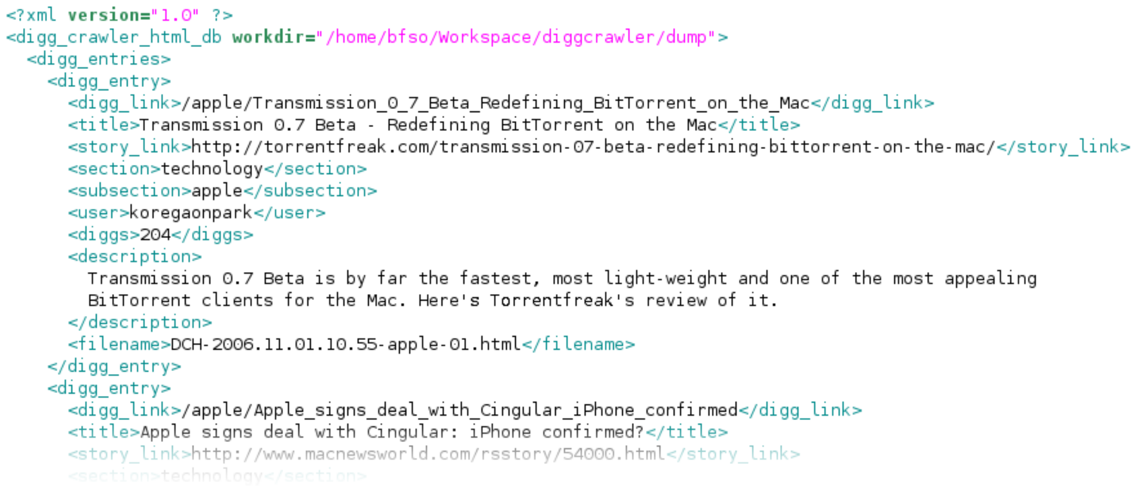
Lucene's framework generates its own index files. Thus, the only thing needed to index the documents in the collection, was a program to interpret the XML version of the database and use the framework methods for processing the files. Lucene requests that the information to be stored should be placed in 'fields', which are assigned a name to be further retrieved. Hence, besides the words in the documents, all the additional information is stored in the index files as well and may be used by the suggester or by any program that needs it.

Given the simplicity of the crawler, which does no processing on the documents it collects, some additional tasks are performed by this module. When importing each document, the program checks whether it is binary and, if so, rejects it; strips all the HTML tags, using Lucene's

[1] http://www.furl.net/

[2] http://www.simpy.com/

[3] http://www.wikipedia.org



```
<?xml version="1.0" ?>
<digg_crawler_html_db workdir="/home/bfso/Workspace/diggcrawler/dump">
  <digg_entries>
    <digg_entry>
      <digg_link>/apple/Transmission_0_7_Beta_Redefining_BitTorrent_on_the_Mac</digg_link>
      <title>Transmission 0.7 Beta - Redefining BitTorrent on the Mac</title>
      <story_link>http://torrentfreak.com/transmission-07-beta-redefining-bittorrent-on-the-mac/</story_link>
      <section>technology</section>
      <subsection>apple</subsection>
      <user>koregaonpark</user>
      <diggs>204</diggs>
      <description>
        Transmission 0.7 Beta is by far the fastest, most light-weight and one of the most appealing
        BitTorrent clients for the Mac. Here's Torrentfreak's review of it.
      </description>
      <filename>DCH-2006.11.01.10.55-apple-01.html</filename>
    </digg_entry>
    <digg_entry>
      <digg_link>/apple/Apple_signs_deal_with_Cingular_iPhone_confirmed</digg_link>
      <title>Apple signs deal with Cingular: iPhone confirmed?</title>
      <story_link>http://www.macnewsworld.com/rsstory/54000.html</story_link>
      <section>technology</section>
```

Figure 3.11: Fragment of the XML version of the corpus database

HTML parser for that matter; discards stopwords; and, using regular expressions, removes all dates, numbers and long words, that were defined as words with more than 20 characters. The idea is to clean the data by getting rid of unimportant information.

Since importing files can take several hours, depending on the size of the collection, and since the corpus started to be used before the crawling process was stopped, the module supports incremental imports. In other words, instead of deleting Lucene's files and starting the index from scratch when new files are added to the collection, this module has an option to append the files that are not in the index yet.

When developing the suggester, some of the data needed by the methods developed in this Thesis, was not made available by Lucene's framework. Computing and storing such data while indexing, would greatly reduce the execution time of such methods. Therefore, additional data processing methods were included in the indexing program.

As will be further explained in subsequent sections, the documents vectors norm was one of such required measures, and it depends on the documents collection only. It is computed using the following formula:

$$||d_i|| = \sqrt{\frac{\sum_{j \in T_i} tf_{ij}^2}{|T_i|}} \tag{3.3}$$

Where T_i is the set of terms of document i and tf_{ij} is frequency of the term j on document i.

Although this information could be stored in a designated field in each document, Lucene does not allow the fields to be updated after they are created. Therefore, a new data structure and a file to store it persistently had to be created. It consists of a simple hash map that works as a dictionary, that is, given the document number – which is the way Lucene identifies the documents on its index – it returns the document's norm.

Prabowo and Thelwall suggest the use of a frequency table to compute χ^2, Mutual Information and Information Gain values [PT06], which were used in the suggester module. Although these tables were computed by the indexer, it is more appropriate to describe them in the next section, as some context is needed to understand their usage.

3.5 Tag Suggesting

The module described in this section does the most noticeable job in the system. It is responsible for taking a document and its additional information as input and presenting appropriate tags as recommendations.

This component, as well as the indexer, described in the previous section, were written in Java, since Lucene was the framework of choice. Since both the indexer and the suggester have to share many details, which make them two closely related components, they were integrated in the same program. Fig. 3.12 represents the main classes developed. 'Tess' is the program's entry point and it is supposed to be executed using a command line interface. It uses the 'Registry' to know which classes comply with the outlined scheme and, thus, can be made available for the

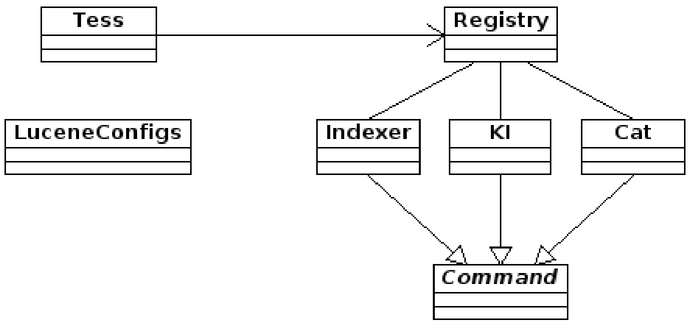


Figure 3.12: Tess class diagram

user to execute. Accordingly, for a class to be executable it should implement the 'Command' interface and register the required information in the 'Registry'. All the information about Lucene's index, such as field names or document paths, is placed in 'LuceneConfig' and can easily be accessed by any class. As for the remaining classes, 'KI' corresponds to the suggester module, 'Cat' is the categorizer and 'Indexer' is the module described in the previous section.

The algorithm for selecting tags consists of two distinct phases, which are described in the following subsections and are illustrated in Fig. 3.13. In short, the query document is processed by the vector displacement module, next its words are ranked and then filtered and presented to the user. Optionally, if no topic was assigned to the document, the categorizer can be used for such matter.

3.5.1 Vector Displacement

The first stage on the process of selecting words is based on vector displacement. The process starts with the query document being converted to a vector, just like the other documents in the

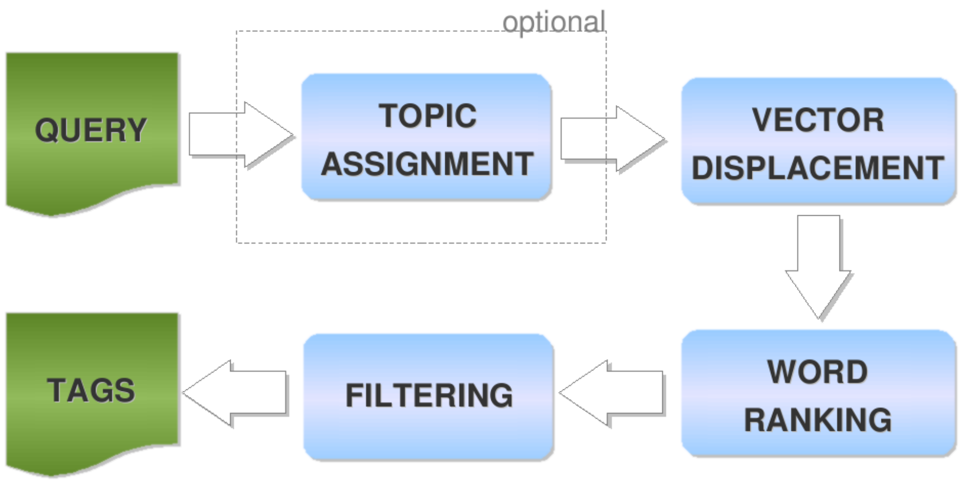


Figure 3.13: Main stages of the tag suggesting module

collection, to be placed in the vector space. The next step is to find the N most similar documents, that is, the closest ones, where N is a user-defined parameter. To quantify the degree of similarity between two documents, the following formula was used [BYRN99]:

$$\begin{aligned} sim(s,p) &= \frac{\vec{s}\cdot\vec{p}}{|\vec{s}|\times|\vec{p}|} \\ &= \frac{\sum_{i=1}^{t} w_{i,s}\times w_{i,p}}{\sqrt{\sum_{i=1}^{t} w_{i,s}^2}\times\sqrt{\sum_{i=1}^{t} w_{i,p}^2}} \end{aligned} \tag{3.4}$$

Where $\vec{s}$ and $\vec{p}$ are the two vectors to be compared, $|\vec{s}|$ and $|\vec{p}|$ are their norms, $w_{i,s}$ and $w_{i,p}$ are the weights of term i in documents s and p, respectively. This formula computes the cosine of the angle formed by the two vectors and, thus, is usually called cosine similarity. The closer the cosine value is to 1, the more similar are the two documents. When the cosine value is 0, the vectors are orthogonal, that is, there are no common terms.

The vectors of the most similar documents are then used to alter the query vector's position, that is, to change its terms weights. Six different formulas were developed for that purpose, and are described next.

Method 1

Use terms from similar documents only, that is, discard the query's terms:

$$\vec{f} = \sum_{s\in S}\frac{1}{|S|}(w_{1,s}, w_{2,s}, ..., w_{n,s}) \tag{3.5}$$

Where $\vec{f}$ is the final vector, S is the set of similar documents and $w_{i,s}$ is the weight of term i in document s.

Method 2

Use all terms:

$$\vec{f} = \sum_{s \in S} \frac{1}{|S| + 1} \Big((w_{1,s}, ..., w_{n,s}) + (w_{1,q}, ..., w_{n,q}) \Big) \tag{3.6}$$

Where q is the query document.

Method 3

Use query terms only

$$\forall_{i \in Q} : f_i = w_{i,q} + \sum_{s \in S} \frac{1}{|S|} w_{i,s} \tag{3.7}$$

Where Q is the set of query terms and f_i is the weight of the term i in the final vector.

Method 4

Use all terms but take vector distance into account

$$\begin{aligned} \vec{f} &= \sum_{s \in S} sim(q, s) \times (w_{1,s}, ..., w_{n,s}) + sim(q, q) \times (w_{1,q}, ..., w_{n,q}) \\ &= \sum_{s \in S} sim(q, s) \times (w_{1,s}, ..., w_{n,s}) + (w_{1,q}, ..., w_{n,q}) \end{aligned} \tag{3.8}$$

Where sim is the cosine similarity, defined in 3.4

Method 5

Similar to Method 2, but changes the vectors importance. Query's terms are worth 95%.

$$\vec{f} = 0.05 \times \left(\sum_{s \in S} \frac{1}{|S| + 1} (w_{1,s}, ..., w_{n,s}) \right) + 0.95 \times \left(\frac{1}{|S| + 1} (w_{1,q}, ..., w_{n,q}) \right) \tag{3.9}$$

Method 6

Based on Method 4, but changes the vectors and the distance importance.

$$\vec{f} = 0.05 \times \left(\sum_{s \in S} \frac{sim(q, s)}{2} \times (w_{1,s}, ..., w_{n,s}) \right) + 0.95 \times \left(\frac{1}{2} \times (w_{1,q}, ..., w_{n,q}) \right) \tag{3.10}$$

Two different approaches to this vector displacement procedure were implemented. The first one was inspired by the *k-means* algorithm [HK00]. Succinctly, the *k-means* is a centroid-based technique for data partitioning. It divides a set of objects into k different clusters, where k is an input parameter. The algorithm starts by randomly selecting k objects to be the clusters centers. Then, each of the remaining objects is assigned to the most similar cluster, based on the distance between the object and the cluster mean. Finally, the clusters centroids are recomputed. The process repeats itself until the criterion function converges.

The *k-means* inspired approach starts by placing the query vector in space. Then, the vector is successively displaced by applying one of the methods described above until it stabilizes, that is, its values do not change between iterations, or until a user-defined limit of iterations is reached. Several experiments indicated that the vectors never stabilize. Because of that, a different approach had to be elaborated.

The second method makes use of the document subtopics. Like in the previous approach, it starts by placing the query document in space. Then, when searching for the most similar documents, only the ones from the same subtopic are considered. In other words, instead of using a vector space with all the documents in the collection, a narrower version of such space is considered, one that contains only the documents which were assigned the same subtopic as the query document's. The vector is then displaced a single time, using one of the methods described above. The results obtained using this approach are more promising than those obtained with the first method. Fig. 3.14 summarizes the process just described.

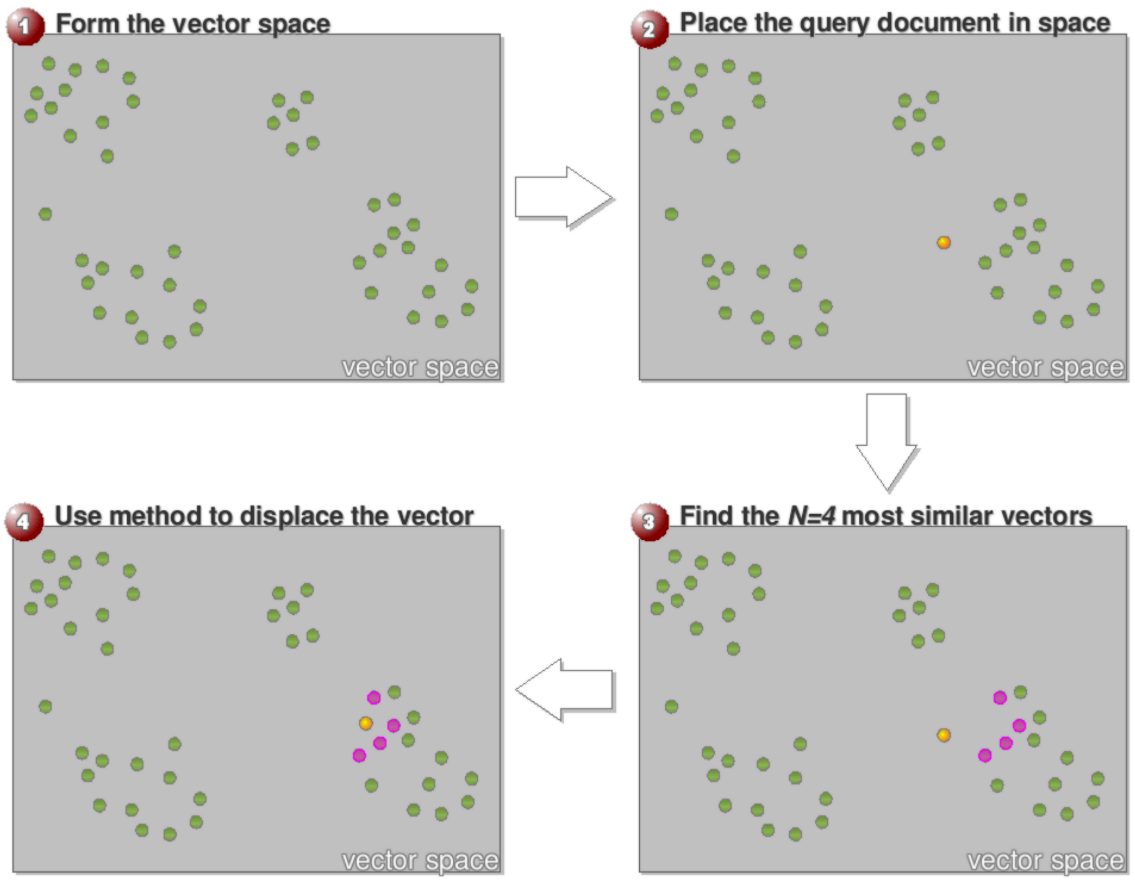

1 Form the vector space
2 Place the query document in space
3 Find the N=4 most similar vectors
4 Use method to displace the vector
vector space

Figure 3.14: Main stages of the tag suggesting module

Next, the algorithm proceeds to the following stage, which consists of choosing which of the vector's words should be recommended.

3.5.2 Tag extraction

Having displaced the original vector, the system proceeds to the extraction of terms that will be presented as tag recommendations. To do so, a criterion to distinguish the good words should be employed on the final vector. This consist in applying a set of formulas to rank each word, that is, to measure how good they are.

The first measures to be experimented were weight (3.1) and the inverse document frequency (3.2). The latter one indicates how difficult it is to find a particular word in the collection, while the first one, tries to find an equilibrium between how useful the term is in describing the document's content – term frequency – and how helpful it is for distinguishing a relevant document from an irrelevant one. Both formulas had already been used in the previous part of this suggesting system. Ranking words by weight produced some satisfactory results. The inverse document frequency, on the other hand, turned out to be less useful, as most of the words were too rare to be considered as tags. Since many documents in the collection are blog entries or have comments written by their readers, this measure was bringing out misspelled words, neologisms, several words concatenated and even onomatopoeias. Words like 'simluator', 'mousejiggler' or 'duhhhhhhhhhh', for instance, were prone to appear as top terms.

Based on these two measures, two new ones were implemented. One of such measures consisted on computing the average of the inverse document frequency of all the terms in the vector and then sort the words by their closeness to that value:

$$idfavg = \sum_{t \in T} \frac{idf(t)}{|T|} \tag{3.11}$$

$$\forall t_1, t_2 : rank(t_1) > rank(t_2) \Rightarrow |idf(t_1) - idfavg| < |idf(t_2) - idfavg| \tag{3.12}$$

Where T is the set of terms. Put in words, the closest the term's inverse document frequency is to the average inverse document frequency, the higher its rank should be.

The other measure consists of ranking the words in two phases. First, the words are ranked according to their weight. From that sorted set, only the top 50 words are kept. All the others are discarded. This new set is then sorted by inverse document frequency and the top terms are extracted. The idea is that, sometimes, rarer words are useful. However, because of the problems described above, they cannot be too rare. So, this measure, $w50idf$, selects the rare words from a set of not-so-rare ones.

The measures presented this far did not take the document topic into account. When considering the categories, other measures can be used. Inverse document frequency, for one, can be adapted to be used in a space partitioned by its documents categories using the following formula:

$$idfcat_{t,c} = \log \frac{N_c}{n_{i,c}} \tag{3.13}$$

Where i is a term, c is a category, N_c is the number of documents in category c and $n_{i,c}$ is the number of documents that contain the term i and are categorized as c. In practice it is like using only the vector space formed by the documents of the given category, instead of considering the whole vector space.

Yang and Pedersen suggest information gain, mutual information and χ^2 statistic as useful term selection methods to be used in text categorization [YP97]. Since the documents in the collection are categorized with the topics they were assigned when submitted to *Digg*, these measures can be used to rank and sort the words on the final vector.

Information gain is commonly used in Machine Learning as a means to measure how good a term is. It is computed using the following formula:

$$ig(t) = -\sum_{c \in C} P(c) \log P(c) + P(t) \sum_{c \in C} P(c|t) \log P(c|t) + P(\bar{t}) \sum_{c \in C} P(c|\bar{t}) \log P(c|\bar{t}) \tag{3.14}$$

Where C is the set of categories and t is a term.

Mutual information is another well-known method for term selection and is computed as follows:

$$mi(t, c) = \log \frac{P(t, c)}{P(t)P(c)} \tag{3.15}$$

While information gain depends only on the term to be evaluated, mutual information, on the other hand, depends on its document's category as well. Being a measure to be used in text categorization problems, the category of the submitted document is unknown. Thus, to measure how good a term is in the global space, two alternative formulas are proposed:

$$mi_{avg}(t) = \sum_{c \in C} P(c) mi(t, c) \tag{3.16}$$

$$mi_{max}(t) = max_{c \in C}\{mi(t, c)\} \tag{3.17}$$

Finally, the third method, χ^2 statistic measures the degree of dependency between a term and a category:

$$\chi^2(t, c) = \frac{N[P(t, c)P(\bar{t}|\bar{c}) - P(t|\bar{c}P(\bar{t}|c)]^2}{P(t)P(\bar{t})P(c)P(\bar{c})} \tag{3.18}$$

Where N is the number of documents in the collection. As in mutual information, this formula depends both on the term and the category. So, two formulas are presented:

$$\chi^2_{avg}(t) = \sum_{c \in C} P(c)\chi^2(t, c) \tag{3.19}$$

$$\chi^2_{max}(t) = max_{c \in C}\{\chi^2(t, c)\} \tag{3.20}$$

In order to improve the efficiency on the computation of the three feature selection methods just described, contingency tables described by Prabowo and Thelwal [PT06] were adapted and integrated in the system. These data structures are intended to store information on the frequencies of co-occurrences of terms and categories. Tab. 3.2, adapted from the referred article, relates a term t with a category c. The slots on the table contain the following information:

- A – number of documents from category c containing the term t
- B – number of documents containing the term t that are not from category c

	c	$\bar{c}$
t	A	B
$\bar{t}$	C	D

Table 3.2: Contingency table for one term and one category

- C – number of documents from category c that do not contain the term t
- D – number of documents that are not from category c and do not contain the term t

Instead of having several small tables and also some redundant information, this model was extended to the one presented in Tab. 3.3. In such table, t_j is the term j, c_i is the category i,

	c_1	c_2	$\cdots$	c_i	F
t_1	$f_{1,1}$	$f_{1,2}$	$\cdots$	$f_{1,i}$	F_1
t_2	$f_{2,1}$	$f_{2,2}$	$\cdots$	$f_{2,i}$	F_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
t_j	$f_{j,1}$	$f_{j,2}$	$\cdots$	$f_{j,i}$	F_j
S	S_1	S_2	$\cdots$	S_i	N

Table 3.3: Generalized contingency table

$f_{j,i}$ is the number of documents categorized as c_i that contain the term t_j. The total number of documents in the collection is given by N. F_j is defined as the total number of documents that contain the term t_j, and thus, is computed by $\sum_{d=1}^{i} f_{j,d}$. Finally, S_i is the total number of documents categorized as c_i and is computed by $\sum_{d=1}^{j} f_{d,i}$. Hence, given a term t and a category c, the values on the smaller table are computed in the following manner:

- $A = contingencyTable[t][c]$
- $B = contingencyTable[t][F] - A$
- $C = contingencyTable[S][c] - A$
- $D = N - (A + B + C)$

Using A, B, C, and D the feature selection formulas can be easily computed. Information gain becomes:

$$\begin{aligned} ig(t) = \quad & -\sum_{d=1}^{i} \frac{contingencyTable[S][d]}{N} \log\left(\frac{contingencyTable[S][d]}{N}\right) \\ & +contingencyTable[t][F] \sum_{d=1}^{i} \frac{A}{A+B} \log\left(\frac{A}{A+B}\right) \\ & +(1 - contingencyTable[t][F]) \sum_{d=1}^{i} \frac{C}{C+D} \log\left(\frac{C}{C+D}\right) \end{aligned} \quad (3.21)$$

Mutual information can be computed as:

$$\log\left(\frac{A \times N}{(A + C) \times (A + B)}\right) \quad (3.22)$$

Lastly, χ^2 statistic equation becomes:

$$\frac{N \times [(A \times D) - (C \times B)]^2}{(A + B) \times (B + D) \times (A + B) \times (C + D)} \tag{3.23}$$

The contingency table was computed by the indexing module and saved to a file to be later loaded and used by the suggester module, instead of being computed each time the system was queried. *Digg*'s subtopics were used as the documents categories.

Since the information gain measure was performing well and presenting some promising results, a new version of this method was tried: $ig2$. *Digg* has a 2-level hierarchy of topics, that is, each subtopic has a parent topic and several subtopics as siblings. For instance, the subtopics ‘Apple’ and ‘Design’ are both descendants of the main topic ‘Technology’. In the information gain measure presented before, only subtopics were considered. Our new version, instead of considering each subtopic as a category in the global space, considers only the documents of the parent topic. For example, when submitting a document labeled with ‘Apple’ as its subtopic, this measure considers only the documents in sibling subtopics, that is, subtopics descending from ‘Technology’. The formula used is the same as (3.14), but with C being the set of sibling subtopics of the query document's subtopic, instead of all the subtopics in the collection. The contingency table is also used to compute this measure.

Some new methods, consisting of combinations of the measures described so far, were tried as well. Basically, they consist of multiplying the term frequency or the term weight by one of the other measures.

Finally, since this system was using *Digg* as its data source, some additional information on the documents that can be used as well. Other than the story being published and the topic under which it should be filed, users are also prompted to provide a title and a description, when submitting a resource to *Digg*. These two extra fields were used as a means to boost the score of the selected words. When a query is submitted to Tess it must have the story, its topic, title and description. The story is processed like explained previously in this chapter. After applying the term extraction measures, the system searches for words in the final vector that are also in the title or description, and raises their scores. Worded differently, if the term in the final vector is in the title or description provided by the submitter, its score is raised. The new score is computed by multiplying the old one by a factor of 3, if the word is found in the title, or by a factor of 2, if it is found in the description, or by a factor of 6, if found in both fields. This feature was named ‘boost’. Since only the information on the query is used, that is, the information on the documents in the collection is ignored, it is straightforward to turn this feature off, which makes the system more adaptable to other websites that follow a different structure than *Digg*'s.

Tab. 3.4 summarizes the term extraction measures described in this subsection.

Additionally, a filtering process was experimented. The idea was to define thresholds on the terms information gain or inverse document frequency, to automatically remove words which scores would be above or below such limits, depending on the rule. However, these filters turned

Basic Measures
$weight$
idf
idf_{avg}
$w50idf$

Category-based Measures
idf_{cat}
ig
mi
mi_{avg}
mi_{max}
χ^2
χ^2_{avg}
χ^2_{max}
ig^2

Combinational Measures
tf×measure
w×measure
$boost$(measure)

Table 3.4: Term extraction methods grouped in categories

out to be useless. They were either too strict and, consequently, some important words were discarded, or too loose and would not alter the set of top tags. In fact, it is very difficult, if not impossible, to find good thresholds on such measures. Sometimes, words that are apparently good appear mixed up with uninteresting ones, that is, they have similar scores, making it hard to decide on the appropriate threshold.

If every combination was tried, it would give a total of 70 possible measures:

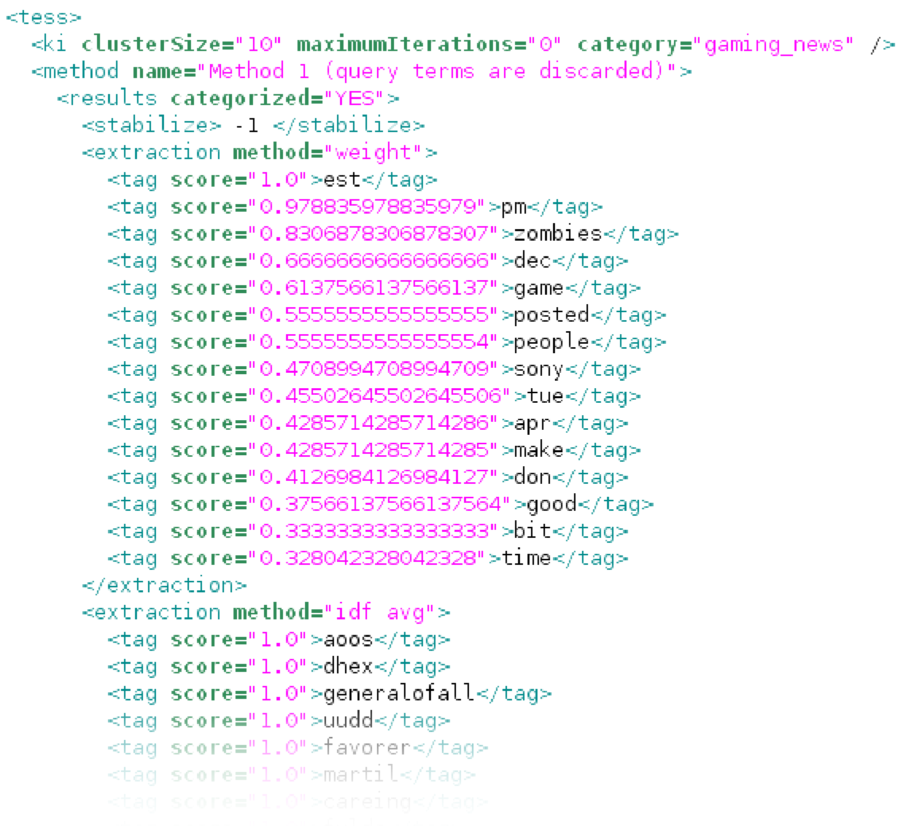
$$\begin{array}{cccccccccc}
\Big(w + idf & +(& BasicMeasures & + & CategoryMeasures &) \times & (tf * + w * + noComb) & \Big) \times & (boost + noBoost) \\
\Big(\quad 2 & +(& 2 & + & 9 &) \times & 3 & \Big) \times & 2
\end{array}$$

However, some measures were abandoned along this research for not producing satisfactory results. So, instead of 70, a total of 30 measures were tried, which are indicated in Tab. 3.5.

w	mi_{avg}	$w \times ig$	$tf \times ig2$	$tf \times \chi^2$
idf	mi_{max}	$tf \times idf_{cat}$	$w \times ig2$	$w \times \chi^2$
idf_{avg}	χ^2	$boost(w)$	$boost(tf \times ig2)$	$boost(tf \times mi)$
$w50idf$	χ^2_{max}	$boost(tf \times ig)$	$boost(w \times ig2)$	$boost(w \times mi)$
ig	χ^2_{avg}	$boost(w \times ig)$	$tf \times mi$	$boost(tf \times chi^2)$
mi	$tf \times ig$	$ig2$	$w \times mi$	$boost(w \times \chi^2)$

Table 3.5: Term extraction measures available in Tess

The last step of this term extraction process is the removal of similar words. To avoid having words like ‘game’, ‘gaming’ and ‘gamer’ on the same set of tags, a filtering procedure is applied. It consists of removing words with the same stem, using an implementation of the Porter stemming algorithm included in Lucene’s framework. Before outputting a set of tags, the top words are stemmed and compared. If two or more equal stems are found, only the word with higher score remains. The other words are removed from the set and give place to lower ranked ones, though having a different stem. By applying this procedure, the diversity of tags improves slightly.



```
<tess>
  <ki clusterSize="10" maximumIterations="0" category="gaming_news" />
  <method name="Method 1 (query terms are discarded)">
    <results categorized="YES">
      <stabilize> -1 </stabilize>
      <extraction method="weight">
        <tag score="1.0">est</tag>
        <tag score="0.978835978835979">pm</tag>
        <tag score="0.8306878306878307">zombies</tag>
        <tag score="0.6666666666666666">dec</tag>
        <tag score="0.6137566137566137">game</tag>
        <tag score="0.5555555555555555">posted</tag>
        <tag score="0.5555555555555554">people</tag>
        <tag score="0.4708994708994709">sony</tag>
        <tag score="0.45502645502645506">tue</tag>
        <tag score="0.4285714285714286">apr</tag>
        <tag score="0.4285714285714285">make</tag>
        <tag score="0.4126984126984127">don</tag>
        <tag score="0.37566137566137564">good</tag>
        <tag score="0.3333333333333333">bit</tag>
        <tag score="0.328042328042328">time</tag>
      </extraction>
      <extraction method="idf avg">
        <tag score="1.0">aoos</tag>
        <tag score="1.0">dhex</tag>
        <tag score="1.0">generalofall</tag>
        <tag score="1.0">uudd</tag>
        <tag score="1.0">favorer</tag>
        <tag score="1.0">martil</tag>
        <tag score="1.0">careing</tag>
```

Figure 3.15: Example of an output from the suggester module

When using Tess, one can choose which vector displacement methods and term extraction measures should be employed, using an XML file. The two stages combined may generate up to 180 sets of tags – 6 vector displacement methods $\times$ 30 term extraction measures. The system's output is presented in an XML file as well. This way, by means of an XML parser or a XSLT (Extensible Stylesheet Language Transformations) file, the information can be easily selected and presented in different ways. Fig. 3.15 depicts part of the output from the tag suggester module when queried with a document categorized as 'Gaming News'.

3.5.3 Categorizer

As stated in subsection 3.5.2, the use of categorized query-documents improves the system's performance. When using *Digg* as the base system, this extra information, required for the suggester to work properly, is not troublesome, since all stories to be submitted must be filed under an appropriate topic. However, not every content sharing website uses a hierarchy of topics or prompts the user to assign a category to the resources being published. Therefore, in order to make Tess a more generic suggesting system, that is, adaptable to a wider variety of websites, this dependency on the resources category had to be solved. A rather straightforward way to solve this issue was to add a categorizer to the system. That way, the main algorithm needed not to be changed as its input is the same. In fact, it just shifted the responsibility of categorizing the resource from the user to a new module in the system.

Since the documents in the collection were retrieved from *Digg* and, thus, were already categorized – using the topic assigned by their publishers – the *k-nearest neighbor* (*k*-NN) method could be employed [MA03]. A *k*-NN classifier uses a pattern space, that is, a n-dimensional space containing several training samples, which have already been classified. When given a sample, the classifier searches the pattern space for the *k* closest samples using, typically, the Euclidean distance. The classification of the query-sample is given by the majority of equal classifications in the set of neighbors. It is as if every neighbor used its own category as its vote. The most voted category gets to be assigned to the query sample. Fig. 3.16 examples a query sample – the black circle – being categorized by choosing the *k*=4 nearest neighbors – enclosed in a dashed circle. The winning category is the 'red' one.

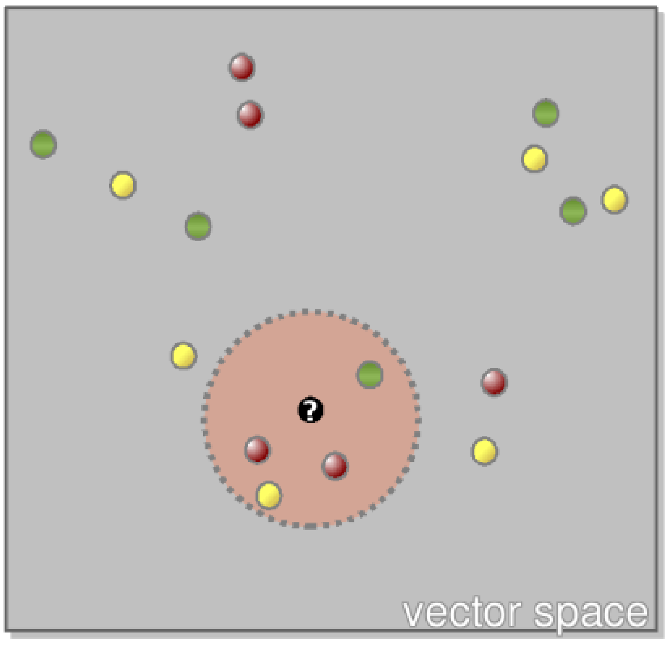


Figure 3.16: Categorizing using a *k*-nearest neighbor method with *k*=4

In the case of Tess, the pattern space corresponds to the vector space with all the documents in the collection. Cosine similarity, (see (3.4)) is used to find the closest neighbors of the query document, instead of the Euclidean distance. Therefore, the most similar vectors are selected as neighbors. The number of neighbors to choose, that is, the value of *k* is an input parameter.

The performance of the categorizer depends a great deal on *k*'s value. Hence, a simple program was written to help finding an appropriate value for *k*. The first step was to retrieve 50 documents, two for each topic, from *Digg*. Each document was tried several times on the categorizer, each time with a different value of *k*, which varied from 1 to 100. Then, for each value of *k*, the number of successfully categorized documents, that is, documents whose topic matched the suggested category, was counted. The chart in Fig. 3.17 shows the percentage of successfully categorized documents by *k*, suggesting values between 10 and 13 as the most appropriate ones.

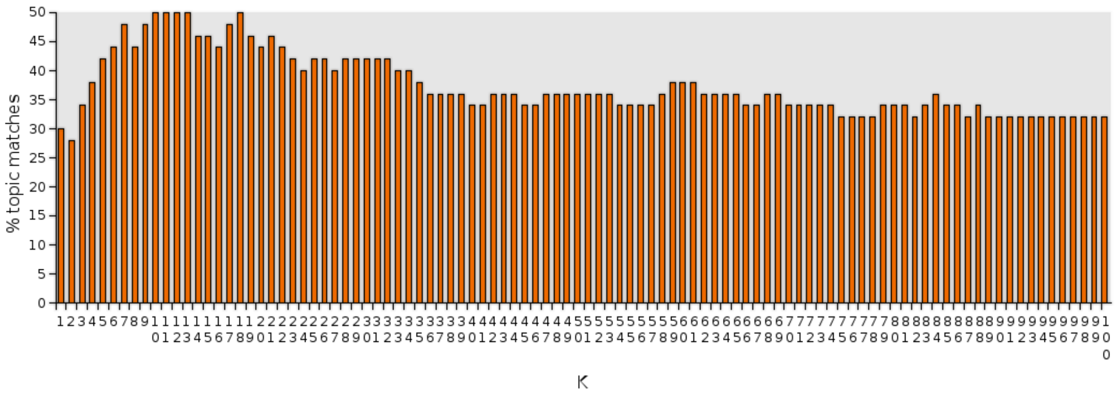


Figure 3.17: Categorizer's rate of success vs. *k*'s values

4 Evaluation

As the system's complexity increased and new features were added, the quality of the suggested tags, apparently, improved. Therefore, in order to accurately assess the system's performance, a rigorous user-based evaluation had to be carried out. Such procedure helped perceiving which methods perform better, whether the system needs further improvement and, if so,what sort of changes may help enhancing it.

4.1 Experimental Setup

Given the large number of methods and, thus, of suggested sets of tags, the first step on setting up an evaluation was to choose which methods to use. In fact, if all the available combinations were to be considered, the user's task would be long and cumbersome. Moreover, some of the methods available on the system were clearly ineffective and, therefore, not worth testing.

Since none of the vector displacement methods was clearly underachieving, they were all included in the evaluation procedure. Regarding term extraction, 16 of the 30 measures were discarded. Only the 14 most auspicious methods were kept: w, $boost(w)$, idf_{avg}, $w50idf$, $tf \times ig$, $w \times ig$, $tf \times mi$, $w \times mi$, $boost(tf \times ig)$, $boost(w \times ig)$, $boost(tf \times mi)$, $boost(w \times mi)$, $w \times ig2$, $boost(w \times ig2)$.

Another goal of this evaluation was to compare the system's performance when using its categorizer, with that obtained when using the topic selected by the story's submitter. Hence, the combinations of methods described above were used to generate sets of tags with and without the use of the system's categorizer. The *k* parameter was set to 12.

According to the conditions just described, a total of 168 sets of tags were subject to evaluation:

$$\begin{array}{ccccc} (tessCat + diggCat) & \times & (vectorDisplacement & \times & termExtraction) \\ 2 & \times & 6 & \times & 14 \end{array}$$

The size of each set of tags was fixed at 15.

Since tagging is a process made by people, the same should happen with the evaluation of this tag suggesting system, Tess.

Two different tasks were requested to the users. Besides reading the document, the users were asked to write down 2 to 5 tags, like they would do on a regular social tagging system. Next, they were prompted to validate each of the suggested tags as either good or bad.

Each user was asked to evaluate 10 documents and each document was evaluated by 3 users. Therefore, the set of user-tags would have up to 15 distinct words. Tagging is a rather subjective process. In fact, tags may be validated differently depending on the users, as some are more demanding about the quality of tags than others. Also, the expertise on the document's subject may influence the user's choices. If a tag is meaningful on the document's context but the user is unaware of their relatedness, the tag will probably be rejected. Contrarily, a user with deeper knowledge on the subject would find the tag most useful. So, by forcing a document to be evaluated by 3 different users, the set of user-tags becomes larger and the diversity on the validation of the suggested tags increases. Hopefully, the issues described above exert less influence on the evaluation.

The experiment was taken by 4 groups of 3 users, which resulted in 12 users and 40 different documents, 10 for each group.

4.2 User-based Evaluation

To proceed with the evaluation of the tag suggesting system, a program to interact with users and manage the whole process had to be built. Three different stages are involved in this procedure. First the data has to be gathered and prepared. Next, the users perform the tasks they were assigned. Lastly, the results are treated using appropriate statistical measures.

The first stage of this procedure consisted of setting up the data to be used by the evaluation tool, that is, the application that interacts with the users. Since there was already a program able to retrieve stories published on *Digg* and then submit them to the tag suggesting system, the report generation tool, there was no need for developing a new one. So, this tool was used to collect the 40 documents, 2 for each topic, and the system's tags. The cluster size, one of Tess input parameters, was set to 10. However, in order to be easily managed by the evaluation tool the produced information, needed further processing. Hence, a new script was implemented. Such setup tool imports the XML result files and processes them, by joining the sets of tags on each document, so that, when displaying the suggested tags to the user, no repeated words appear. Furthermore, the setup tool is responsible for creating the user groups and randomly assigning documents to each group, and consequently, to each user. It also generates the passwords needed for user authentication.

The second and most important stage consists on the user evaluation of the suggester's performance. To interact with the users, an online application was crafted. Small web applications are quick and fairly straightforward to develop, can easily store the results in a server, and work in any operating system, as long as an Internet connection and a compliant web browser are available.

Seeing that the content of each webpage would depend on the user and also the evaluation results had to be stored, a traditional HTML-only website would not be cut for the job. Thus, a more complex engine was required. After designing the HTML templates, the engine for managing data and saving results was written. PHP was the language of choice, mainly for being suited to server-side scripting and because, it was already available at the server where the evaluation took place. In some aspects, PHP is similar to Python and Perl. However, PHP code can be embedded in regular HTML pages, making it especially useful to write web applications.

The web application is responsible for selecting the appropriate information, that is, the documents and tags under evaluation, and display them to the user. Moreover, it stores the results in their proper place, along with the user and document identification.

The user interaction goes as follows:

1. Log in by entering the username and password
2. Pick a document to evaluate
    (a) Read the document
    (b) Write down 3 to 5 tags
    (c) Classify each and every suggested tag as either good or bad

The process terminates when all ten documents are evaluated. There is no time limit for this procedure and the user needs not evaluate every document in one session. As long as no document is under evaluation, he can logout and return later to complete his task. Another factor worth noticing is that the user is asked to write down his own tags before seeing the system's suggestions, so that they will not influence his reasoning.

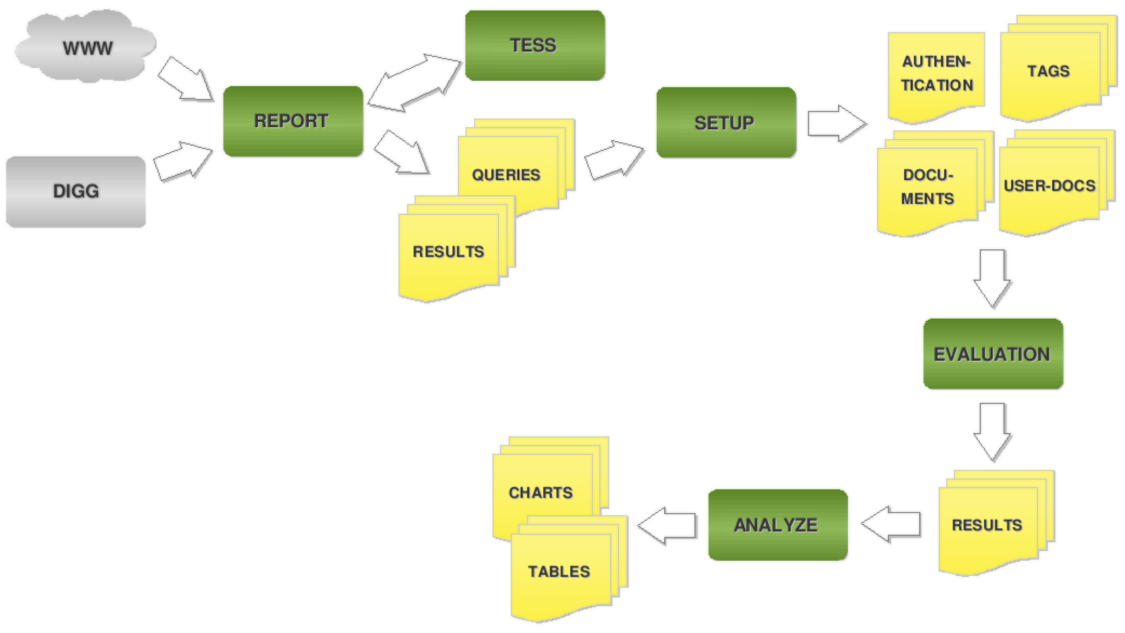


Figure 4.1: General view of the evaluation tools and data files

The auxiliary data, needed for the application to manage the evaluation process, is stored in three types of files. For authentication, it uses a table that associates users with their passwords. This module prevents the system from unwanted visits, while making it easier to select which information to display or store. For each document under evaluation there is a corresponding file containing the tags suggested by the system, ready to be presented to the users. Finally, there is a file for each user that indicates which documents he should evaluate and which ones are already evaluated. As to the results saved by the application, a file for each user-document association is created to store both the user tags and the classification he assigned to the ones suggested by the system.

The evaluation process terminates when all the users have finished evaluating the documents they were assigned to. Having all the required data, an application to join the results and apply the appropriate statistical measures had to be developed. A Python script was created for this purpose. Using PyChart library, besides processing the information and generating several tables, the most relevant charts were printed to help in the subsequent analysis.

The 3-stages process, described in this section, is summarized in Fig. 4.1. In short, the report tool retrieves the documents and queries the tag suggester; the setup script processes the data to be used in the evaluation; the web application manages the interaction with users and stores the results; the analysis tool generates charts and tables using proper statistical measures.

4.3 Evaluation Results

System performance can be assessed over different dimensions. Therefore, when carrying an evaluation process such as the one described in this chapter, one must carefully choose the proper measures.

Space and time consumption are usually worth measuring. The smaller the space and the shorter the response time, the better. In this particular system, such aspects are not particularly relevant, mainly because the suggester was not yet put to production. Tess was not yet combined with *Digg* and put into work in real performance. In fact, it was constantly under development, as it is more of a research tool. Therefore, efforts were made towards enhancing the quality of the system's output, and not so much on its efficiency in terms of time and space consumption. If Tess were to be integrated in a social tagging system, it should first undergo a process of optimization. Besides, not every method combination and data structure would be needed. By reducing the number of sets to be generated, the need for time and space would be reduced as well.

In any case, currently Tess uses a corpus of 844MB, which, through the indexing process, reduces its size to 197MB, approximately. When running, the suggester needs around 150MB of RAM and takes about 95 seconds to generate 180 sets of tags, with the cluster size of 10. Hence, on average, each set takes roughly 0.5 seconds to be produced.

Baeza-Yates and Ribeiro-Neto present some well-known measures in the field of information retrieval, found to be very relevant and useful, as they focus on the quality of the system's output and the users expectations [BYRN99]. In our case, we are going to discuss results for precision, coverage/recall and novelty. Such measures were adapted to better suit the tag suggesting problem.

Precision is among the most commonly used measures. In Tess' case, it indicates the amount of good tags in all of the suggested ones. Let T be the set of tags suggested by one of Tess methods and R_T a subset of T, containing only the tags found to be relevant by the users:

$$precision = \frac{|R_T|}{|T|} \tag{4.1}$$

Since there were three users evaluating the tags suggested for each document, three precision measures were derived from eq. (4.1):

- $precision1^+$ – a tag is considered relevant if at least 1 user finds it so
- $precision2^+$ – a tag is considered relevant if at least 2 users find it so
- $precision3$ – for a tag to be considered relevant, all 3 users must find it so

The average of each precision formula was computed for every combination of vector displacement methods with tag extraction measures. For instance, to find out the *precision1+* of the combination of the first vector displacement method with $tf \times ig$ term extraction measure, the following formula was employed:

$$\frac{1}{40} \times \sum_{d=1}^{40} \frac{|R1^+_{Td}|}{|T|}$$

Where Td is the set of tags produced by 'Method 1', described in Section 3.5.1 combined with

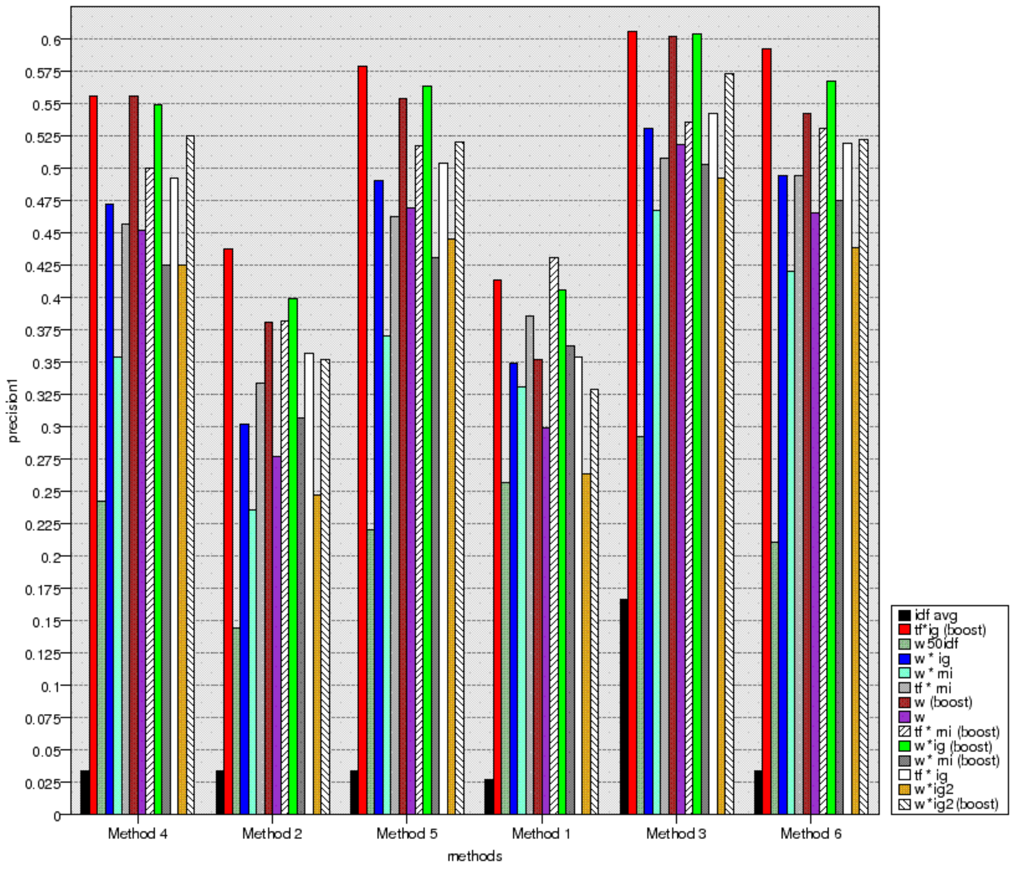


Figure 4.2: Chart for $precision1^+$

'$tf \times ig$' for document d and $R1^{+}_{Td}$ is the set of tags considered relevant by at least 1 of the users. An analogous procedure applies when using other precision formulas or method combinations.

The charts for $precision1^{+}$, $precision2^{+}$ and $precision3$, when using the topic chosen by *Digg*'s submitter, are presented in Fig. 4.2, 4.3, 4.4, respectively. Clearly, the precision value decreases as the relevancy condition gets stricter. This result emphasizes the lack of consensus among users when it comes to decide whether a tag is relevant or not, and reinforces the idea that tagging is a subjective process.

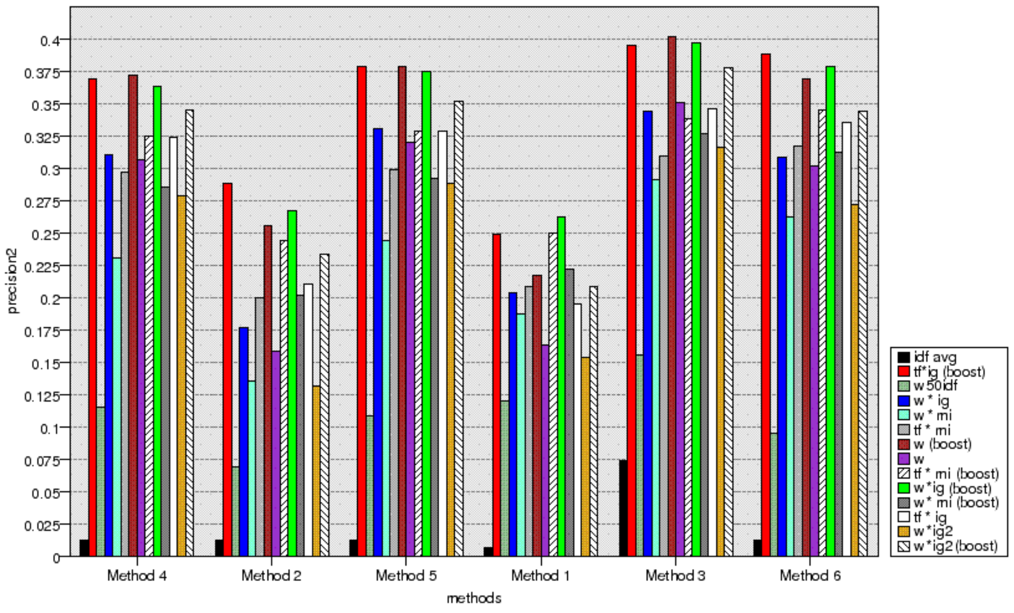


Figure 4.3: Chart for $precision2^{+}$

The most effective methods combinations are the same in both $precision1^{+}$ and $precision2^{+}$. In $precision3$, no method really stands out, as their performances are low and somewhat similar. In terms of vector displacement methods, 'Method 3', which uses terms in the query document only, produces the best results, closely followed by 'Method 5'. As to term extraction measures,

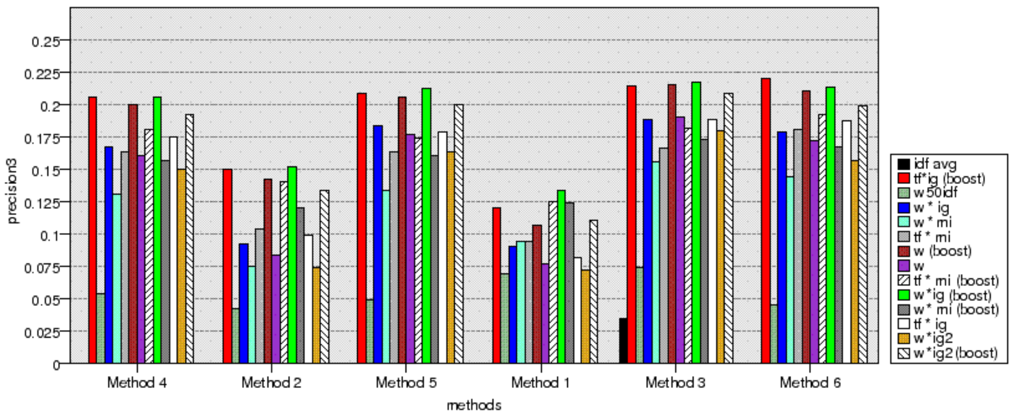


Figure 4.4: Chart for $precision3$

‘$boost(tf \times ig)$’, ‘$boost(w \times ig)$’, and ‘$boost(w)$’ are usually the strongest ones. Also, the term extraction measures always perform better when using the words from *Digg*’s title and description to boost the tags scores. The first version of the information gain measure, ig, outperforms the second one, $ig2$, no matter the vector displacement method being used. As for the worst term extraction measures, idf-related ones have the lowest precision values, which is, probably, because they bring out words that are too unusual to be considered good tags.

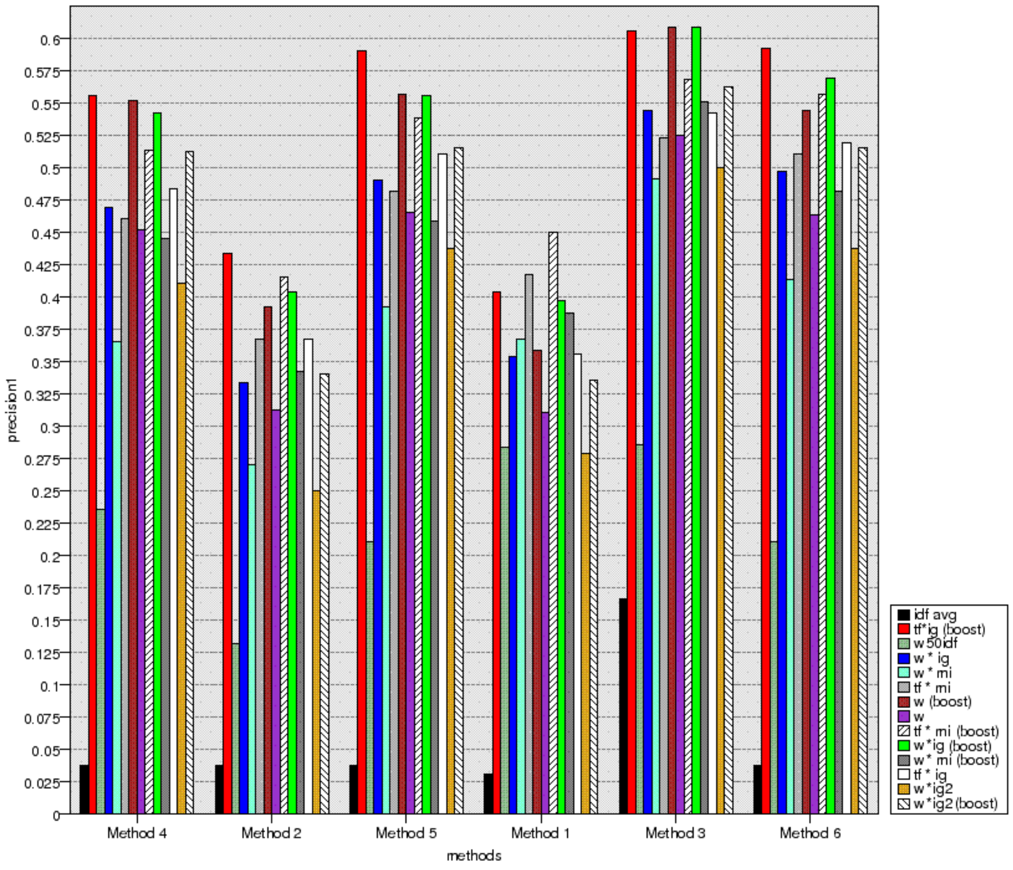


Figure 4.5: Chart for $precision1^{+}$ when using Tess built-in categorizer

The results for precision when using the topic suggested by the system’s categorizer are very similar to those obtained using the topic assigned by *Digg*’s user, as depicted in Fig. 4.5. In fact, in some cases, the categorizer can help improve the performance of some methods combinations, although the difference is not very significant. These results may be explained by the existence of some degree of similarity among different topics, or through the use of inappropriate topics, by some of *Digg* users, when publishing resources. Moreover, several stories could easily and adequately be filed under different topics, as they approach several subjects.

All in all, this tag suggesting system can achieve an average precision of 60%. Some methods have similar performances, which might suggest that further combinations should be tried to improve the system’s precision. The results also suggest that vector displacement methods should consider terms in the query document as more prominent.

Coverage is a user-oriented evaluation measure. It is similar to the more well-known 'recall', as it indicates the amount of tags the users found to be relevant that were suggested by the system. The original formula was adapted to better suit Tess:

$$coverage = \frac{|T \cap U|}{|U|} \tag{4.2}$$

Where T is the set of tags recommended by Tess and U is the set of tags written by the users. Analogously to what was done with the precision measures, the average coverage was computed for each combination of methods.

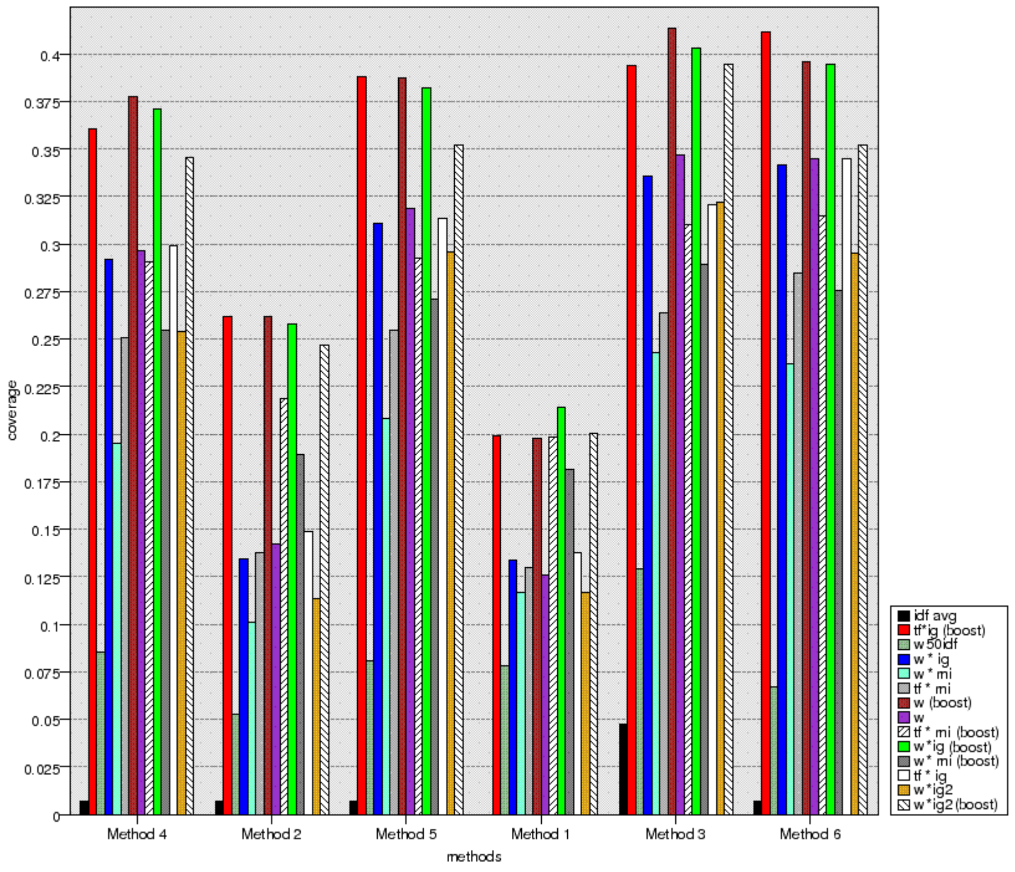


Figure 4.6: Chart for *coverage*

The results for coverage, presented in Fig. 4.6, are consistent with the precision ones. The best vector displacement method is 'Method 3', closely followed by 'Method 6'. Regarding term extraction, '$boost(w)$' and '$boost(w \times ig)$' turn out to be the best measures for 'Method 3', though '$boost(tf \times mi)$' and '$boost(tf \times ig)$' are not too far behind. In 'Method 6', on the other hand, '$boost(tf \times mi)$ does not stands out as a top measure. Results, so far, point to 'Method 3' followed by '$boost(w \times ig)$' or '$boost(w)$' or '$boost(tf \times ig)$' as the strongest combinations. On average, over 40% of the user-tags are covered by the system.

In information retrieval evaluations it is usual to plot a precision vs. recall chart. Since coverage is the user-based version of recall, employed in this system's evaluation, a precision vs. coverage curve was plotted. However, using one of the precision measures defined above would not make much sense in this case, as coverage takes into account the user-suggested tags only, whereas precision considers the system-suggested tags and how they were validated by the users. Hence, the general formula for precision (4.1) is adapted to:

$$precisionC = \frac{|U \cap T|}{|T|} \tag{4.3}$$

Where U is the set of user-suggested tags and T is the set of tags recommended by the system.

This type of chart requires the tags to be ordered by their rank in a descending manner. To understand how the precision vs. coverage curves are plotted, an example is presented next. Let U_d be the set of user-suggested tags for document d:

$$U_d = \{a, b, d, j, n, o, p, q\}$$

Now suppose the system suggested the following tags:

1.	$a\star$	6.	$o\star$	11.	y
2.	e	7.	l	12.	u
3.	x	8.	z	13.	w
4.	$d\star$	9.	k	14.	v
5.	s	10.	$q\star$	15.	$n\star$

The tags that were also suggested by the users were marked with a star. Proceeding with the analysis on the system-suggested tags, the first tag is found to be relevant and corresponds to 12.5% of all the relevant tags in set U_d. So, there is a precision of 100% at 12.5% coverage. The next tag to be considered relevant is the fourth one, d. Thus, there is a precision of 50% (two relevant tags out of four) at 25% coverage. This analysis is applied to the entire set in order to plot the precision vs. coverage curve.

The proceeding described above is applicable to single documents only. Therefore, to evaluate each method combination, the average precision vs. coverage must be computed:

$$precisionC_{avg}(c) = \sum_{d=1}^{40} \frac{precisionC_d(c)}{40} \tag{4.4}$$

Where c is the coverage level and $precisionC_d(c)$ is computed at coverage level c for the document d.

Furthermore, since the number of user-suggested tags varies between documents, an interpolation procedure must be conducted. The idea is to have a fixed number of coverage levels for each query document. Usually, 11 standard levels are used in these charts. Let c_l, $l \in \{0, 1, ..., 10\}$ be the reference to the l-th coverage level:

$$precisionC(c_l) = max_{c_l < c < c_{l+1}} precisionC(c) \tag{4.5}$$

That is, the interpolated precision at the l-th standard coverage level is the maximum precision at any coverage level between c_l and c_{l+1}. The application of this procedure in the example described above would give:

coverage	precision
12.5%	100%
25.0%	50%
37.5%	50%
50.0%	40%
62.5%	33%

interpolation

$\Rightarrow$

coverage	precision
0%	100%
10%	100%
20%	50%
30%	50%
40%	50%
50%	40%
60%	33%
70%	0%
80%	0%
90%	0%
100%	0%

The interpolation procedure is applied before computing the average precision for each level. Otherwise, the coverage levels for each document would not match.

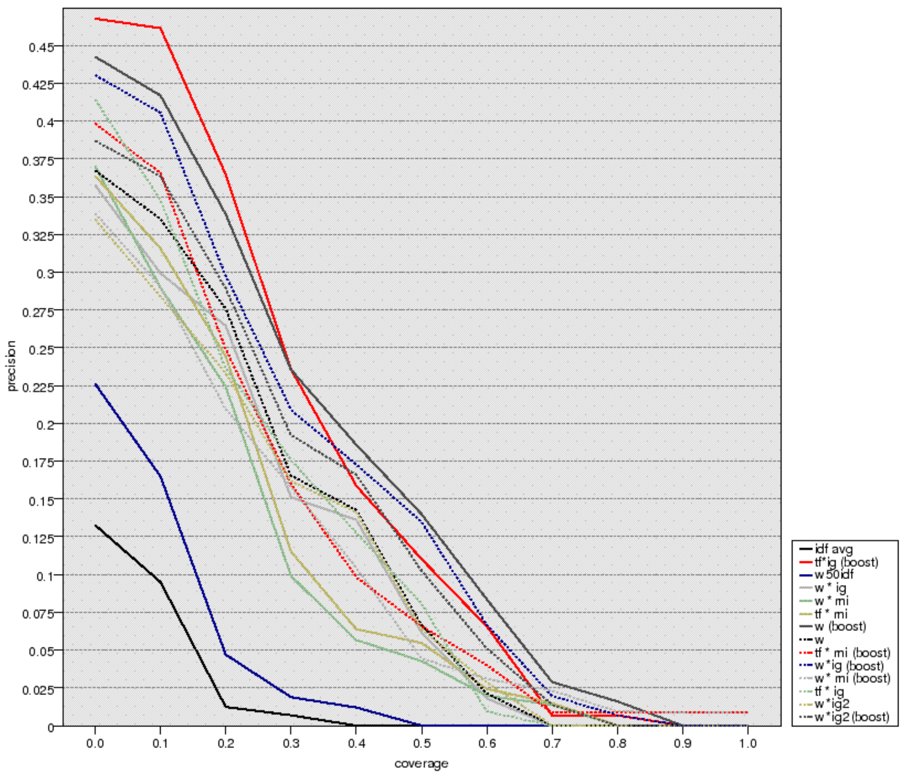


Figure 4.7: Chart for *precisionC* vs. *coverage* for 'Method 3' when using Tess categorizer

As can be seen in Fig. 4.7, '$boost(tf \times ig)$' starts as the most precise of all term extraction measures. However, the difference between other good measures gets less significant as the coverage level increases. In fact, for coverage levels of over 30%, the '$boost(tf \times ig)$', '$boost(w)$' and '$boost(w \times ig)$' curves are very similar. Another factor to consider is the curve slope, which is rather steep. This remark suggests that the user-tags which are also suggested by the system are scattered around the set, that is, there are many tags that were not suggested by the user between the ones wrote down by the users. On that account, the ranking mechanism could be enhanced.

Another meaningful measure is the novelty ratio, which can be used to find out whether the system is revealing new relevant tags the user was previously unaware of. A high ratio indicates that many relevant tags are being discovered by the user. Since Tess is a recommender system, it is somewhat important to have a good novelty ratio, as it means the system is being successful in aiding the user by suggesting tags he could not recall or think of by himself. It is computed as follows:

$$novelty = \frac{|R_T - U|}{|R_T \cup U|} \tag{4.6}$$

Where R_T is the set of suggested tags considered relevant by the users and U is the set of tags wrote down by the users.

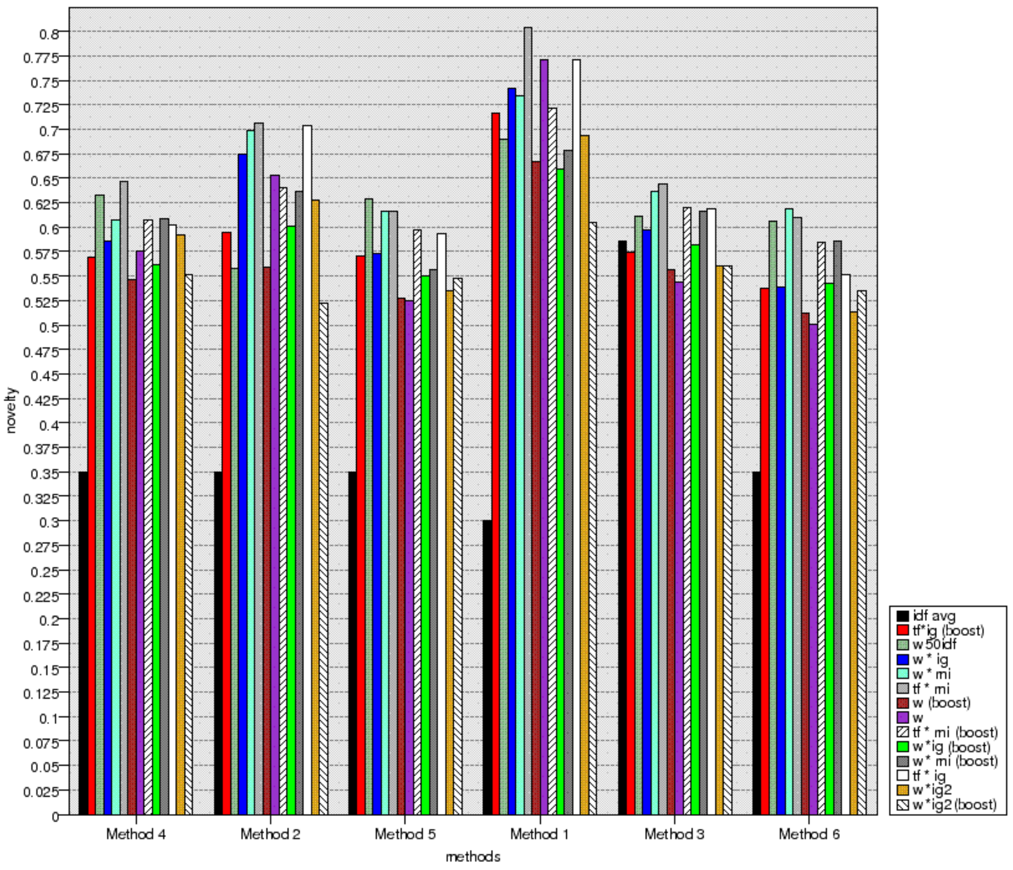


Figure 4.8: Chart for *novelty* when using the topic proposed by the categorizer

The chart presented in Fig. 4.8 indicates ‘Method 1’ as the vector displacement method with highest novelty ratio. However, this method’s precision and coverage are low. Therefore, it should not be considered a strong method, as most recommended tags are found irrelevant by the users.

As for ‘Method 3’ and ‘Method 6’, the methods with the highest precision and coverage rates, their novelty ratio is quite similar. The best term extraction measures, ‘$boost(tf \times ig)$’, ‘$boost(w \times ig)$’, ‘$boost(w)$’, have an average novelty ratio of over 50%. That is, in average, more than half of the suggested tags found to be relevant, were new to the users.

Altogether, despite useful, the novelty ratio cannot be a decisive measure by itself, as it is less important than precision and coverage.

The last charts to be plotted were the precision vs. number of tags. The idea is to figure how the precision varies with the number of suggested tags. If the tags are being properly ranked, then the curve should be descent, indicating that the appearance of lower ranked tags decreases the precision, as they are less relevant.

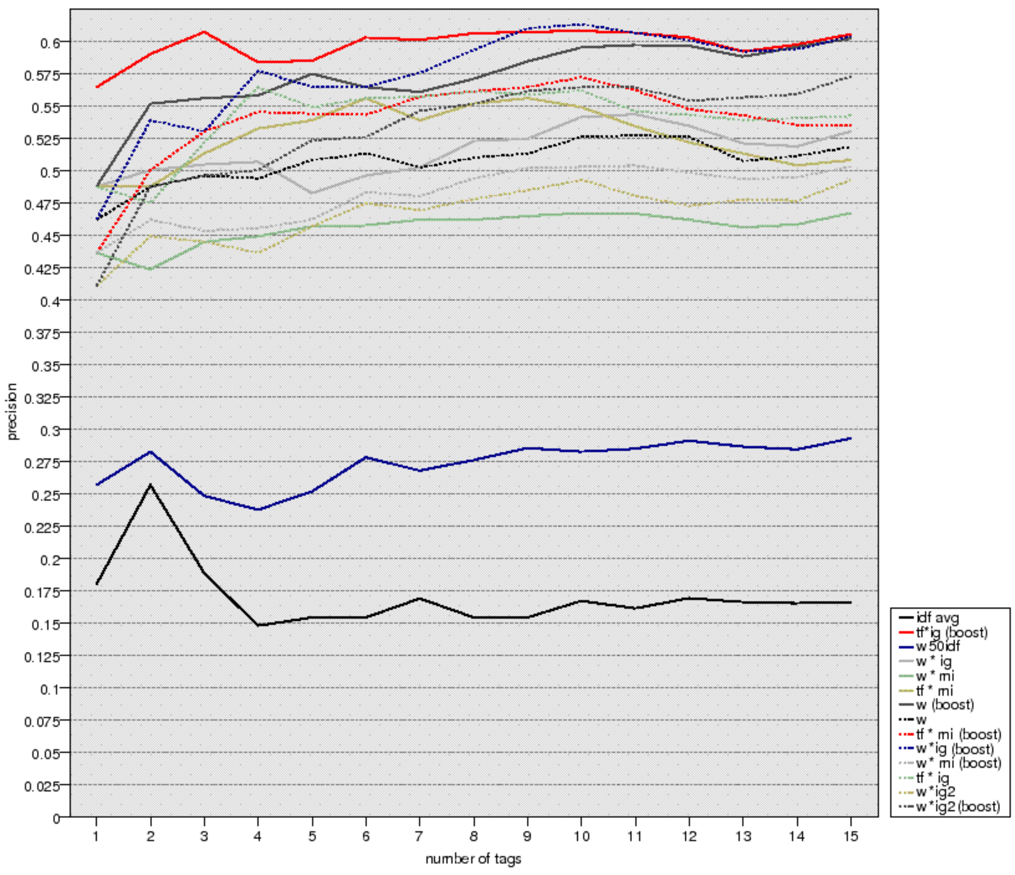


Figure 4.9: Chart for $precision1^{+}$ vs. number of suggested tags for ‘Method 3’

The sinuous curve plotted in Fig. 4.9 suggests a faulty tag ranking mechanism. The tags significance is not expected to increase as lower ranked tags are added to the set. Therefore, these results enforce the need for a better ranking of the terms to be suggested. Ideally, less relevant words should not appear before the more important ones.

The results of this experiment can also be used to examine, to some extent, how users choose the words to assign to the documents. On that account, another measure was employed to determine how many user-suggested tags are found in the document being tagged:

$$tagsInDoc_i = \frac{|U_i \cap D_i|}{|U|} \tag{4.7}$$

Where U_i is the set of tags wrote down by the users and D_i is a set with all the words in the document being tagged, that is, document i. The average was then computed:

$$\sum_{d=1}^{40} \frac{tagsInDoc_d}{40} \tag{4.8}$$

On average, approximately 80.6% of the user-suggested tags can be found in the document. This result complies with the conclusions drawn from the precision and coverage charts. Indeed, according to precision and coverage, the strongest vector displacement methods are those who give more importance to the terms in the query document. So, as expected, the amount of user-suggested tags that can be found in the documents is fairly large.

In conclusion, several measures were applied to this experiment's results in order to assess the system's performance and discover some of its drawbacks as well.

When using 'Method 3', for vector displacement, combined with '$boost(tf \times ig)$', '$boost(w \times ig)$' or '$boost(w)$' for term extraction, the tag suggesting system can achieve an average precision of 60%, approximately, which means 3 out of 5 recommended tags are thought to be relevant by at least one user.

Coverage values, on the other hand, are a bit lower, as it is harder for the system to recommend the same tags as the users. Even so, much of the tags it suggests are considered relevant, which is a good thing. Besides, when computing the average coverage, synonymy was not considered. In fact, some of the user-suggested tags might have the same meaning as some of the system-suggested ones, and thus should be considered as equivalent. Nevertheless, both coverage and precision agree as to which methods perform best. These results are supported by the fact that most user-suggested tags can be found in the query-document. Accordingly, the methods that give more importance to the terms in the query should perform better.

The steepness of the 'precision vs. coverage' curves, as well as the sinuous form of the 'precision vs. number of tags curves', point out the need for improvement of the ranking mechanism. The former ones indicate that there is a great distance between the positions of the user-suggested tags that can be found in the set of tags recommended by the system. In the latter ones, the precision should decrease as lower ranked tags are presented to the user. Both cases call for a rearrangement in the tags disposition on the final set. Ideally, tags placement in the set should reflect their relatedness to the query-document, that is, the most relevant tags should appear closer to the top.

The novelty ratio was another measure employed in this evaluation. However, it should not be used as a decisive measure, for it is not as relevant as precision or coverage. In fact, it is more important to have many good tags, though not so new to the user, than few and previously unknown ones. Besides, it is not possible to decide whether the user did not recall the tags he considered relevant or simply decided to recommend some others he found more important. Yet, the fact that the degree of novelty is higher in methods with lower precision, suggests that combining such methods may possibly improve the results.

The remarks described above are roughly the same whether using the topic suggested by *Digg*'s submitter or by the system's built-in categorizer. Also, the results are quite similar for both approaches, which means the categorizer does not influence the results very much.

5 Conclusions and Future Work

The widespread use and growing popularity of tagging mechanisms calls for enhancements on the user experience. In fact, tagging is usually a manual process and could be eased with the inclusion of tag recommending systems. So, instead of tagging the resources by himself, the user would be given a set of tags as suggestions or hints which he can decide whether they are good and worth using.

This thesis work presents such a recommending system, aimed at textual resources only. *Digg* was selected as the website on which this suggester would be integrated. Hence, Tess fits *Digg*'s hierarchy of topics and uses its top stories to build the corpus. Nonetheless, it is supposed to be adaptable to other types of websites, provided their resources are text-based.

The proposed solution is inspired in information retrieval and data mining techniques. Several methods were implemented. Some were discarded for their disappointing performance, whereas others were combined as an attempt to improve results. Indeed, the combinations proved to be advantageous, as their outcome was indicated as being the best by the evaluation procedure.

The preliminary results were promising. The system can obtain an average precision of 60%, which means over half of the tags it suggests are considered relevant. Apart from some performance tweaks, regarding memory and, especially, time consumption, Tess integration with websites and exposure to real-life situations could be experimented. According to the evaluation carried out at the final stage of this project, there are still some drawbacks to solve and further improvements to work on. In fact, precision should be enhanced as well as the ranking mechanism. Also, some other combinations could be experimented. Moreover, one of the most successful features was $boost$, which consists on raising the score of the terms found in the story's title or description chosen by its submitter. Hence, it is closely related to *Digg*'s structure, as it relies on data that is not so commonly available. This feature could not be used if Tess were to be integrated with a system that does not have this additional information. For instance, in most blogs, there is no short description of the article. In this manner, more general term extraction measures should be developed in order to make this tag suggesting system adaptable to other kinds of websites.

As to future work on this tag suggester, some possible guidelines on how to further improve the system, are presented next. As was stated before, the system can be optimized in order to reduce its memory consumption and, especially, the time it needs to find the terms it will recommend. With a faster suggester, the integration with a real life website, such as *Digg*, would be feasible. In such a situation, user feedback could be used to enhance the methods and, consequently, improve the results. Moreover, it would make it easier to evaluate the system's performance. The task of providing feedback would be unnoticed by the users. When submitting or using a resource, the system would display a set of recommended tags and the chosen ones would be considered good. Furthermore, this continuous evaluation process would probably be more accurate then the procedure described in the previous chapter, as it would use a larger set of documents and users.

Some authors, mentioned in *Related Work*, explore the graph-like structure of folksonomies in their algorithms. In fact, these can be used to guess users interests, resources relatedness and to detect some relations between tags. If Tess were to be used with a social tagging system, it could be advantageous to employ such algorithms. Instead of using cosine similarity only to find similar resources, an algorithm to estimate how related the resources are can be applied to ameliorate cluster formation. Also, the user interests can be detected and used to narrow the search space. For example, if a user whose resources are mainly on technology and environment is publishing a new resource, probably its topic will be on technology or environment. This information can be used to help the categorizer or narrow the vector space, when forming the cluster for term extraction. Also, if the document has few words, this can be a useful and perhaps more accurate way of improving results. When choosing the words to be recommended, the tags deemed related by the graph algorithms may be of value. However, the system should keep suggesting new words as well, that is, it should not use only tags other users assigned to their resources, as it would limit the evolution of the vocabulary and newer words would take more time to be captured and introduced in the corpus.

Currently, Tess deals only with single-word tags. However, there are several occasions where tags with multiple words are much more appropriate, if not inevitable. For instance, ‘Blu-ray’, a high-density optical disc format, ‘Star Wars’, a popular science fiction movie of the late 70s and ‘Mars Volta’, a progressive/experimental rock band, if used as single-word tags, would lose their meanings. In fact, if an article on the latest album of ‘Mars Volta’ was tagged with ‘Mars’ and ‘Volta’, that is, two single-word tags instead of one multi-word tag, it would probably be linked to resources about the planet Mars, according to the folksonomies graph-like structure. Hence, when searching or browsing, the users would probably stumble upon unrelated resources. However, there is not consensus on whether tags should be single- or multi-word. Indeed, popular websites, such as *flickr* or *del.icio.us*, support single-word tags only, whereas *Technorati* or *Last.fm* allow for both single- and multi-word tags. In any case, to add support for this feature in the tag suggesting system, an inference technique based on n-gram frequency could be employed [MS99]. In this case, an n-gram is a group of n words from a given string that appear next to each other. The corpus can be used to estimate the probability of two or more words appearing together. Words that are strongly related have a high probability and thus may be suggested as a multi-word tag. This kind of predicting gets more difficult as the n increases. However, tags with several words are infrequent, so, bigrams or trigrams, that is n-grams with $n = 2$ or 3, would likely suffice.

Intuitively, there is a strong relation between topics and dates. As a matter of fact, people’s interests are moved by trends and change as time goes by. For instance, after the September 11, 2001 attacks, the media was flooded with news and documentaries on terrorism, al-Qaeda, Osama Bin Laden and so forth. Hence, there is a period in time where a great deal of the stories published by the media as well as the society, at large, were focused on terrorism and the 9/11 attacks. The Apple's recently launched mobile phone, iPhone, is a similar example. Ever since the iPhone was announced until it reached the stores, there has been a lot of hype, especially among tech-savvy and Apple users. This sort of information can be a precious hint in determining the resources topics. Thus, the system could use the submission date along with the data it is already using to investigate trends and better decide on the resources subjects. Moreover, when dealing with documents with few words, this additional information can help selecting the similar ones, since that cosine similarity is a method based only on words.

Usually, not all the information on a webpage is relevant. In fact, many webpages display, other than the main story, navigation structures with hyperlinks to other sections, advertisement and, sometimes, user comments. However, this information is useless to the tag suggesting system. It may also deviate it from the actual subject of the resource, especially when the number of words of the story is considerably lower than the remaining ones. For that reason, it would be useful to strip off irrelevant data from each file. Two alternatives seem reasonable for this purpose. One would be using a machine learning technique to find patterns on a set of annotated documents. That is, the documents on the training set would be divided in sections and each would be labeled as 'advertising', 'navigation', 'comments', 'information', for instance. Then, an appropriate machine learning algorithm would use that set to find patterns and build a classifier. Another possible alternative would be to use natural language processing techniques. By looking at websites, one can see that, usually, both advertisement and navigation structures are either small sentences or words. Also, there might be hints on the HTML structure itself. A simple part of speech tagger could be used as well, in order to identify words as being nouns, verbs, adjectives, prepositions, pronouns, adverbs, conjunctions, or interjections. Short strings where almost every word is either a noun, a verb or an adjective will probably be part of the navigation structure. This sort of reasoning may also work with advertisements. Alternatively, tools like Adblock [1] with an up to date version of FilterSet.G [2], for instance, can help removing ads from the documents. Regarding user comments, they are usually smaller than the article they were assigned to and follow the same pattern. Typically the user's name, as well as the date when the comment was posted, are always attached to the text itself. So, what changes between topics is the user's name, the date and the comment itself. Their structure is the same and might be deducted. At this point it is hard to predict which approach would perform better and how would this influence the suggester's behavior.

Additionally, some other data sources could be integrated in the tag suggesting system. A thesaurus could be used to detect synonymy and related words; *del.icio.us*, *BibSonomy*, for instance, could provide the tags assigned to the stories submitted on both *Digg* and the social bookmarking website; *Wikipedia* could be used to determine whether a word is strong by checking if there is an article about it and by the number of times it is referred on other articles or to help categorizing the

[1] http://www.adblock.org/

[2] http://www.pierceive.com/

documents. However, this would reduce the system's independence and could introduce an extra load on the algorithms and thus increasing the memory consumption and the response time. All these factors must be considered, since, no matter how precise the system is, it will not be very useful if it takes longer to respond than the users are willing to wait for. Even so, these techniques may be worth trying.

In conclusion, the result of this thesis work is a tag suggesting system completely functional and ready to integrate in a social content website. The system was started from scratch, so, it took a lot of experimentation and new ideas to define which direction would be best. Now that it is more stable and mature, there are some tweaks and enhancements that may improve the system's performance, and thus are worth trying. Nonetheless, tagging is a subjective process, which makes it hard both to develop and to evaluate the performance of a recommending system. In order to become a full-fledged tag suggester, Tess should be able to deal with non-textual resources, such as video and audio, as well. However, this would require different and rather complex techniques and, perhaps, a whole new approach. Hence, the scope is, for now, limited to text-based resources only, which still accounts for the majority of the resources on *Digg*'s front pages and, probably, of other popular websites.

Bibliography

[BM06a] Christopher H . Brooks and Nancy Montanez, *An analysis of the effectiveness of tagging in blogs*, Proceedings of the 2005 AAAI Spring Symposium on Computational Approaches to Analyzing Weblogs, Stanford, CA., 2006, pp. 9–14.

[BM06b] Christopher H. Brooks and Nancy Montanez, *Improved annotation of the blogosphere via autotagging and hierarchical clustering*, WWW '06: Proceedings of the 15th international conference on World Wide Web (New York, NY, USA), ACM Press, 2006, pp. 625–632.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern information retrieval*, ACM Press / Addison-Wesley, 1999.

[CLA+03] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl, *Is seeing believing?: how recommender system interfaces affect users' opinions*, CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems (New York, NY, USA), ACM Press, 2003, pp. 585–592.

[del07] del.icio.us, *Help: What are tags?*, accessed August 3, 2007, http://del.icio.us/help/tags.

[Dig07] Digg.com, *Digg / frequently asked questions*, accessed July 31, 2007, http://www.digg.com/faq.

[EM02] L. Egghe and C. Michel, *Strong similarity measures for ordered sets of documents in information retrieval*, Information Processing & Management **38** (2002), no. 6, 823–848.

[HJSS06] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme, *Information retrieval in folksonomies: Search and ranking*, ESWC '06: Proceedings of the 3rd European Semantic Web Conference, 2006, pp. 411–426.

[HK00] Jiawei Han and Micheline Kamber, *Data mining: Concepts and techniques*, Morgan Kaufmann, San Francisco, CA, USA, 2000.

[HP98] Thomas Hofmann and Jan Puzicha, *Mixture models for co-occurrence and histogram data*, ICPR '98: Proceedings of the 14th International Conference on Pattern Recognition-Volume 1 (Washington, DC, USA), IEEE Computer Society, 1998, pp. 192–194.

[JM80] F. Jelinek and R. Mercer, *Interpolated estimation of Markov source parameters from sparse data*, Proceedings of the Workshop on Pattern Recognition in Practice, North-Holland, 1980, pp. 381–397.

[Lay06] Julia Layton, *How digg works*, June 2006, Available at: http://computer.howstuffworks.com/digg.htm.

[Luc07] Apache Lucene, *Lucene-java wiki - powered by*, accessed August 9, 2007, http://wiki.apache.org/lucene-java/PoweredBy.

[MA03] Sushmita Mitra and Tinku Acharya, *Data mining: Multimedia, soft computing, and bioinformatics*, Wiley-Interscience, 2003.

[Mat04] Adam Mathes, *Folksonomies - cooperative classification and communication through shared metadata*, Available at: http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html, 2004.

[Mis06] Gilad Mishne, *Autotag: a collaborative approach to automated tag assignment for weblog posts*, WWW '06: Proceedings of the 15th international conference on World Wide Web (New York, NY, USA), ACM Press, 2006, pp. 953–954.

[MNBD06] Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis, *Ht06, tagging paper, taxonomy, flickr, academic article, to read*, HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia (New York, NY, USA), ACM Press, 2006, pp. 31–40.

[MS99] Christopher D. Manning and Hinrich Schütze, *Foundations of statistical natural language processing*, The MIT Press, Cambridge, Massachusetts, 1999.

[Pon07] Jason Pontin, *Directing digg*, Technology Review (2007), Available at: http://www.technologyreview.com/Biztech/19057.

[PT06] Rudy Prabowo and Mike Thelwall, *A comparison of feature selection methods for an evolving rss feed corpus*, Inf. Process. Manage. **42** (2006), no. 6, 1491–1512.

[Qui05] Emanuele Quintarelli, *Folksonomies: power to the people*, Paper presented at the ISKO Italy-UniMIB meeting. Available at: http://www.iskoi.org/doc/folksonomies.htm, 2005.

[Seb02] Fabrizio Sebastiani, *Machine learning in automated text categorization*, ACM Computing Surveys **34** (2002), no. 1, 1–47.

[Tre06] Win Treese, *Web 2.0: is it really different?*, netWorker **10** (2006), no. 2, 15–17.

[Wat06] Darren Waters, *Readers are digging the news online*, BBC News (2006), Available at: http://news.bbc.co.uk/2/hi/technology/5128386.stm.

[WZM06] Harris Wu, Mohammad Zubair, and Kurt Maly, *Harvesting social knowledge from folksonomies*, HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia (New York, NY, USA), ACM Press, 2006, pp. 111–114.

[WZY06] Xian Wu, Lei Zhang, and Yong Yu, *Exploring social annotations for the semantic web*, WWW '06: Proceedings of the 15th international conference on World Wide Web (New York, NY, USA), ACM Press, 2006, pp. 417–426.

[YP97] Yiming Yang and Jan O. Pedersen, *A comparative study on feature selection in text categorization*, Proceedings of ICML-97, 14th International Conference on Machine Learning (Nashville, US), 1997, pp. 412–420.

A User Manual

The following sections describe the usage of the main components developed during this thesis work. Other auxiliary tools, considered to be less relevant, were left out of this chapter as they are fairly simple to use. The software was written for the Linux operating system, and thus, may require some minor modifications to be able to run in other operating systems. Apart from the evaluation system, which has a graphical user interface, all the tools are based on a command line interface.

A.1 Crawler

The crawler is an application to fetch top stories from *Digg*. For each story, it gathers the title, both *Digg*'s and the story's URL, the description written by the submitter, the topic and subtopic on which it was filed, the number of votes it had at the time of retrieval and the name of the user who published the story. The information is stored in a SQLite database, whereas the story itself is saved in a new file in an appropriate directory.

The application was written using Python programming language and works with version 2.5.1. Some additional libraries are required for the program to work. The access to the SQLite 3 database is done through the pysqlite2 interface. A library for chart generation, PyChart, is used for report generation.

Before running the program, the path to the working directory, that is, the place where both the stories and the database will be stored, should be introduced. This is done by editing the script and writing the full path between apostrophes, after the variable `working_dir` and the equal sign. For instance, if the files were to be saved at '/home/digg', the following line should be defined in the beginning of the script:

```
working_dir = '/home/digg'
```

Having defined the working directory, the crawler can now be run from a common shell. Fig. A.1 displays the help menu, which explains how the application should be called. The flags determine how the tool should behave.

When using the 'list' option, *Digg*'s subtopics are enumerated. Such list is intended to help the user decide which sections to use or exclude from the crawling process.

To define how deep the crawler should go on each section, that is, the maximum number of pages it should look up, the 'depth' flag must be activated, followed by a positive integer. The default value was set to 3.

The 'generate' option is useful for printing reports or exporting the database to XML. The type of resource to be generated should be indicated after the flag. For an HTML page with hyperlinks to each section on *Digg*, 'sections' should be used; to export the database to XML, the string 'db'

```
$  python digg_crawler_html.py --help
Usage:
digg_crawler_html.py [options] [sections]

Options:
  -h, --help                          display this info


  -l, --list                          list all sections


  -d, --depth <int>                   set the number of pages to be checked in each section


  -g, --generate <sections|db|stats>  print an HTML page with links to digg sections,
                                      according to the specified depth (sections) | print an
                                      XML page with the database contents | print an HTML page
                                      with statistics on the corpus and the retrieval process


  -x, --exclude                       crawl every section except the ones passed as argument


  sections:                           sections (not -- if -x is used --) to be crawled
                                      (default = all) HINT: use --list option
```

Figure A.1: Crawler's help menu

should follow the flag; to generate a report on the retrieval process and the corpus status, ‘stats’ should be used.

Apart from ‘depth’, the flags seen so far are for auxiliary features. For the crawler to retrieve data, only ‘depth’ and ‘exclude’ options can be used. The crawler retrieves pages from the sections written by the user, next to the options. Except if the ‘exclude’ flag is activated, in which case it retrieves pages from every section on *Digg*, except for those written by the user, next to the options.

Since *Digg* is constantly evolving, as its outlook changes and new sections are added, this tool can quickly become outdated. Instead of using an official API – unavailable when the crawler was developed – the information is discovered by parsing the HTML. Any change in the HTML templates may cause the crawling procedure to be unsuccessful.

A.2 Tess

The main feature of the system is located in Tess module. It is responsible for processing and indexing the data retrieved by the corpus, recommending tags when queried, and suggesting a topic to unclassified documents.

Tess was written in Java and uses Lucene framework. Thus, Sun's Java virtual machine (JVM) 1.5, or other compatible JVM, must be installed and Lucene files `lucene-core-2.1.0.jar` and `lucene-demos-2.1.0.jar`, from version 2.1.0, must be included in the classpath. Since this program requires a considerable amount of memory, Java's `-Xmx` and `-Xms` flags can be used to increase the heap size, making it less likely to run out of memory.

Running Tess from the command line with an unrecognized or no flag displays a help menu, depicted in Fig. A.2 Each flag activates a different work unit of the system.

```
$  java -Xmx256M -cp lucene-core-2.1.0.jar:lucene-demos-2.1.0.jar:. Tess
Usage: Tess --indexer db.xml
Usage: Tess --ki QueryProperties.xml cluster_size max_iterations [LuceneConfig.xml]
Usage: Tess --info corpora | terms-idf | terms-ig
Usage: Tess --cat query.html max_neighbours
```

Figure A.2: Tess help menu

To start or update the corpus, an XML file containing the information on the crawler's database should be passed as argument to the indexer, which is activated when the 'indexer' flag is used.

To use the tag recommendation module, the 'ki' flag should be used. All the additional information associated with *Digg*'s stories should be included in the query and is passed to the system through an XML file containing such properties, that is, title, description, path to the query file, story's URL, section, subsection, number of votes, and *Digg*'s URL. For now, only the first three fields are required. The properties file is composed by several key-value tuples, as illustrated in Fig. A.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="title">Samsung unveils World's First Hybrid Hard Drive</entry>
  <entry key="description">
    Just over a month since releasing their HHD to OEMs, out pops an 80GB Hybrid Hard Drive in
    Samsung's venerable M55 laptop. They're calling it an "industry first".
  </entry>
  <entry key="filename">report03/gadgets1.html</entry>
  <entry key="story_link">
    http://www.engadget.com/2007/04/23/samsungs-m55-sports-worlds-first-hhd/
  </entry>
  <entry key="section">technology</entry>
  <entry key="subsection">gadgets</entry>
  <entry key="diggs">373</entry>
  <entry key="user">2courtneysmile</entry>
  <entry key="digg_link">/gadgets/Samsung_unveils_World_s_First_Hybrid_Hard_Drive</entry>
</properties>
```

Figure A.3: Example of a properties file for a query

The other mandatory arguments are the size of the cluster and the maximum number of iterations, that is, two non-negative integers. They are both parameters for the vector displacement algorithm. The first one indicates how many similar documents should be considered. The latter one must be set to zero to use the category-based algorithm for vector displacement. Otherwise, the older version, which ignores the documents topics, is used and the vector is displaced consecutively until it stabilizes or the maximum number of iterations is reached.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <!-- vector methods configuration -->
  <entry key="VECTOR_DEFAULT">TRUE</entry>
  <entry key="METHOD_NO_QUERY">FALSE</entry>
  <entry key="METHOD_QUERY_DOCS">FALSE</entry>
  <!-- term extraction methods configuration -->
  <entry key="TERM_DEFAULT">FALSE</entry>
  <entry key="WEIGHT">TRUE</entry>
  <entry key="TF*IG">TRUE</entry>
  <entry key="WEIGHT*IG">TRUE</entry>
</properties>
```

Figure A.4: Example of a KI configuration file

Finally there is an optional argument, that tells which vector displacement methods and term extraction measures should be used. It is an XML properties file, similar to the query one. Fig. A.4 is an example of a possible configuration. In this example, by default, all vector displacement methods should be used, except for ‘Method 1’ and ‘Method 2’. As to term extraction measures, none other than ‘w’, ‘$tf \times ig$’ and ‘$w \times ig$’ should be used. Tables A.1 and A.2 show the correspondence between the methods and the XML keys.

Key	Method
METHOD_NO_QUERY	Method 1
METHOD_QUERY_DOCS	Method 2
METHOD_QUERY	Method 3
METHOD_AVG_SIM	Method 4
METHOD_QDOCS_PAR	Method 5
METHOD_AVGSIM_PAR	Method 6

Table A.1: Correspondence table for vector displacement methods

The results are presented in an XML file. Using Fig. A.5 as an example, in the ‘ki’ tag there is information on the cluster size, the maximum number of iterations and the query-document’s category. Next, for each vector displacement method, there is a set of results. If the documents categories are considered by the method, then ‘categorized’ is set to ‘YES’ and ‘stabilize’ is set to 1, as the vector is displaced only once. Otherwise, ‘categorized is set to ‘NO’ and stabilize will

Key	Measure
WEIGHT	w
IDF	idf
IDF_AVG	idf_{avg}
IDF_WEIGHT50	$w50idf$
IG	ig
MI	mi
MI_AVG	mi_{avg}
MI_MAX	mi_{max}
CHI2	χ^2
CHI2_AVG	χ^2_{avg}
CHI2_MAX	χ^2_{max}
TF*IG	$tf \times ig$
WEIGHT*IG	$w \times ig$
TF*MI	$tf \times mi$
WEIGHT*MI	$w \times mi$

Key	Measure
TF*CHI2	$tf \times \chi^2$
WEIGHT*CHI2	$w \times \chi^2$
TF*IDF_CAT	$tf \times idf_{cat}$
WEIGHT(BOOST)	$boost(w)$
TF*IG(BOOST)	$boost(tf \times ig)$
WEIGHT*IG(BOOST)	$boost(w \times ig)$
TF*MI(BOOST)	$boost(tf \times mi)$
WEIGHT*MI(BOOST)	$boost(w \times mi)$
TF*CHI2(BOOST)	$boost(tf \times \chi^2)$
WEIGHT*CHI2(BOOST)	$boost(w \times \chi^2)$
IG_CAT	$ig2$
TF*IG_CAT	$tf \times ig2$
WEIGHT*IG_CAT	$w \times ig2$
TF*IG_CAT(BOOST)	$boost(tf \times ig2)$
WEIGHT*IG_CAT(BOOST)	$boost(w \times ig2)$

Table A.2: Correspondence table for term extraction measures

```
<tess>
  <ki clusterSize="10" maximumIterations="0" category="gadgets" />
  <method name="Method 3 (use query terms only)">
    <results categorized="YES">
      <stabilize> -1 </stabilize>
      <extraction method="tf * mi">
        <tag score="1.0">aol</tag>
        <tag score="0.5767901121573205">yahoo</tag>
        <tag score="0.5231040124726032">engadget</tag>
        <tag score="0.38928015896659685">neutral</tag>
        <tag score="0.3496371154846127">bloglines</tag>
        <tag score="0.34272782946215247">flash</tag>
        <tag score="0.3291415856761632">myyahoo</tag>
        <tag score="0.30712833210800755">samsung</tag>
      </extraction>
    </results>
  </method>
</tess>
```

Figure A.5: Example of a Tess output file containing a set of 8 tags

either be the number of iterations it took for the vector to stabilize or zero if the maximum number of iterations was reached. Then, for each term extraction method, a set of tags and their scores is presented.

Tess built-in categorizer can be used to assign a topic to a document. To do so, flag ‘cat’ should be activated and the HTML document, as well as the number of neighbors – the *k* parameter of the *k*-nearest neighbors algorithm – to consider, should be passed as arguments. The result is printed to the standard output.

Finally, an auxiliary module was developed as well. It is useful for inspecting the documents in the corpus. In this manner, if the flag ‘info’ is followed by ‘corpora’, a table with the number of documents for each topic is printed. To print a list with all terms, sorted either by inverse document frequency or information gain, ‘terms-idf’ or ‘terms-ig’, respectively, should follow the ‘info’ flag.

A.3 Evaluation System

The evaluation system is composed of three tools. The first one is responsible for importing and processing the query files and the suggested tags, for the web application, which will be used by the users to evaluate the system, to be able to function. This setup script was written in Python and requires the following directories to be present:

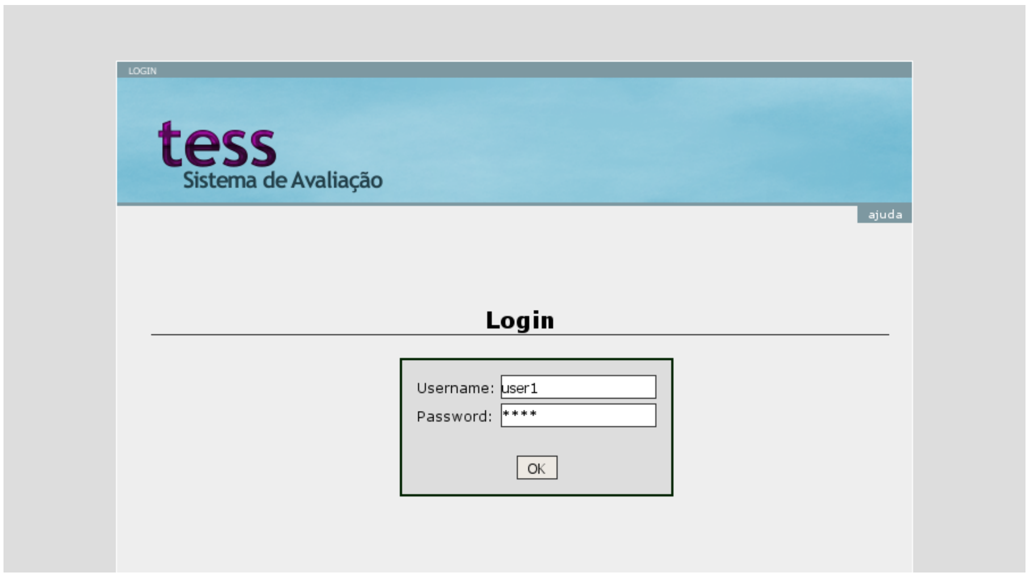
- ‘db’ – user information and the documents tags are placed here
- ‘docs’ – query documents are put in this directory
- ‘reportA’ – Tess output files when using the topic chosen by the story submitter, that is, a *Digg*’s user
- ‘reportB’ – Tess output files when using the topic assigned by its categorizer
- ‘results’ – to store Tess results and the users answers gathered by the web application

The usernames, as well as the number of users that should be assigned to each document and the number of documents each user will have to evaluate, must be defined. This can be done by directly editing the script’s variables, as depicted in the following example:

```
users = ['user1, 'user2', 'user3', 'user4', 'user5', 'user6']
numUsersPerDoc = 2
numDocsPerUser = 5
```

Since there are 6 users and each should examine 5 documents, and also, it is required that every document should be examined by 2 users, then, at least $\frac{6}{2}\times 5 = 15$ documents should be available.

After the setup, the evaluation proceeds on a regular web browser. The first step is the authentication. As illustrated in Fig. A.6, the user introduces its username and password and presses the ‘OK’ button.

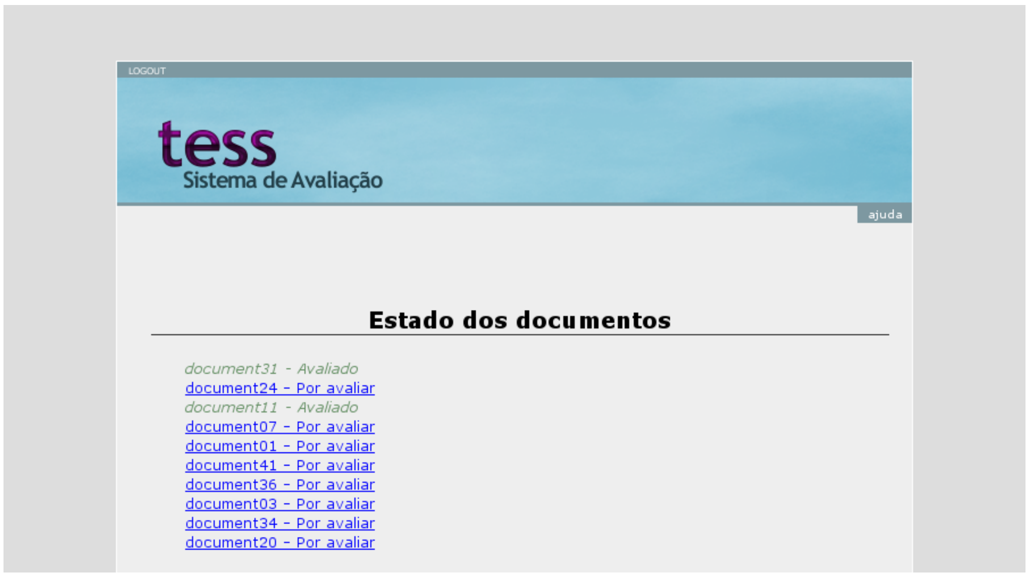


Login

Username: user1
Password: ****

Figure A.6: User logging into the evaluation tool

If the login is successful, a list of all the documents assigned to the user are displayed on the screen, as well as their status. As depicted in Fig. A.7, the documents that were not evaluated yet are underlined and have a link to their evaluation page. The user needs not examine the documents sequentially, nor it is impelled to complete the evaluation in a single session. In fact, he can log out at any time and postpone the remaining tasks. However, he should not log out in the middle of a document evaluation, as his answers will be lost.

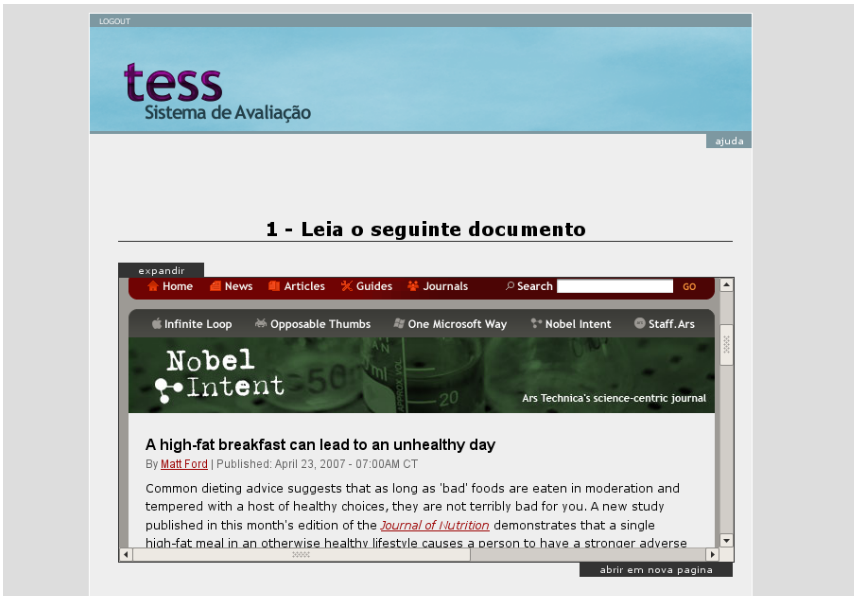


Estado dos documentos

document31 - Avaliado
document24 - Por avaliar
document11 - Avaliado
document07 - Por avaliar
document01 - Por avaliar
document41 - Por avaliar
document36 - Por avaliar
document03 - Por avaliar
document34 - Por avaliar
document20 - Por avaliar

Figure A.7: List of all documents assigned to a user and their status

Each document evaluation consists of three steps. First, the user is asked to read the document. As can be seen in Fig. A.8, the user can expand the article box to fill the screen, by using the link near the top left corner of the box, or open it in a new window or tab using the link near the bottom right corner of the box.



1 - Leia o seguinte documento

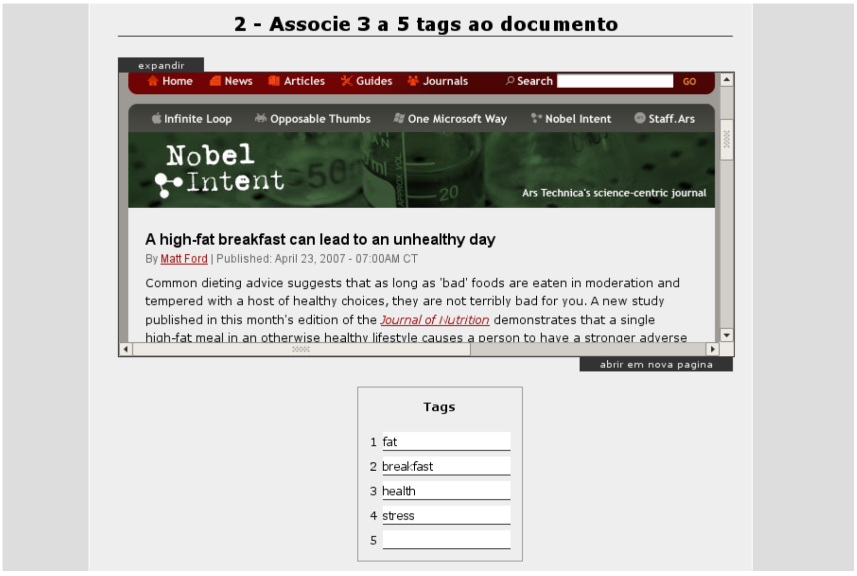
A high-fat breakfast can lead to an unhealthy day

By Matt Ford | Published: April 23, 2007 - 07:00AM CT

Common dieting advice suggests that as long as 'bad' foods are eaten in moderation and tempered with a host of healthy choices, they are not terribly bad for you. A new study published in this month's edition of the Journal of Nutrition demonstrates that a single high-fat meal in an otherwise healthy lifestyle causes a person to have a stronger adverse

Figure A.8: Evaluation's first step: reading the document

After reading the story, the user is prompted to write down 3 to 5 tags he finds relevant, just like he would do in any regular social tagging system. Fig. A.9 illustrates this procedure.



2 - Associe 3 a 5 tags ao documento

A high-fat breakfast can lead to an unhealthy day

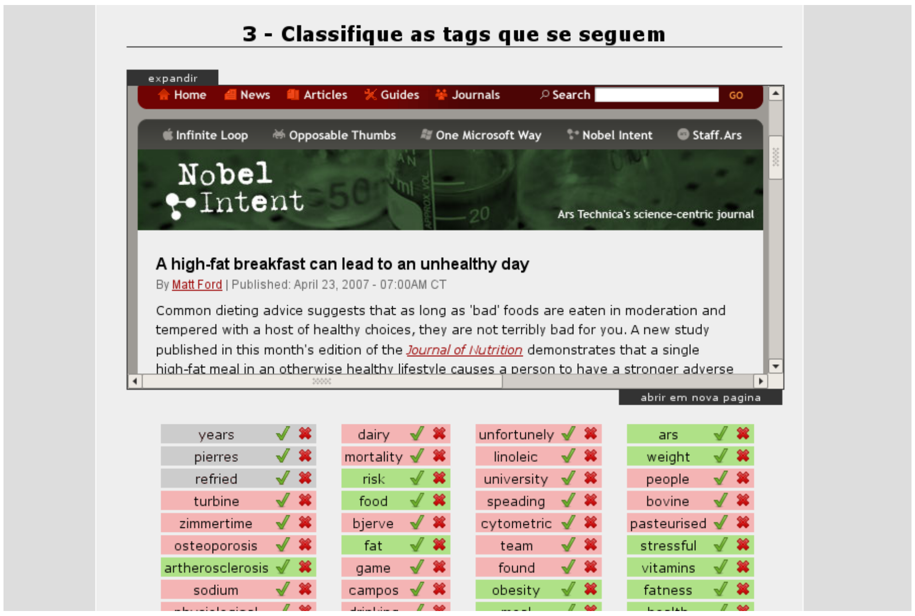
By Matt Ford | Published: April 23, 2007 - 07:00AM CT

Common dieting advice suggests that as long as 'bad' foods are eaten in moderation and tempered with a host of healthy choices, they are not terribly bad for you. A new study published in this month's edition of the Journal of Nutrition demonstrates that a single high-fat meal in an otherwise healthy lifestyle causes a person to have a stronger adverse

Tags	
1	fat
2	breakfast
3	health
4	stress
5	

Figure A.9: Evaluation's second step: assigning 3 to 5 tags

The last step consists on rating the tags recommended by the system. Like depicted in Fig. A.10, each tag can be rated as being either good – the green check mark – or bad – the red 'X' icon. When a tag is rated, its background changes from gray to green or red, depending if the vote was positive or negative. The user must rate every single tag, and may change his decision before finishing the evaluation.


3 - Classifique as tags que se seguem

expandir

A high-fat breakfast can lead to an unhealthy day

By Matt Ford | Published: April 23, 2007 - 07:00AM CT

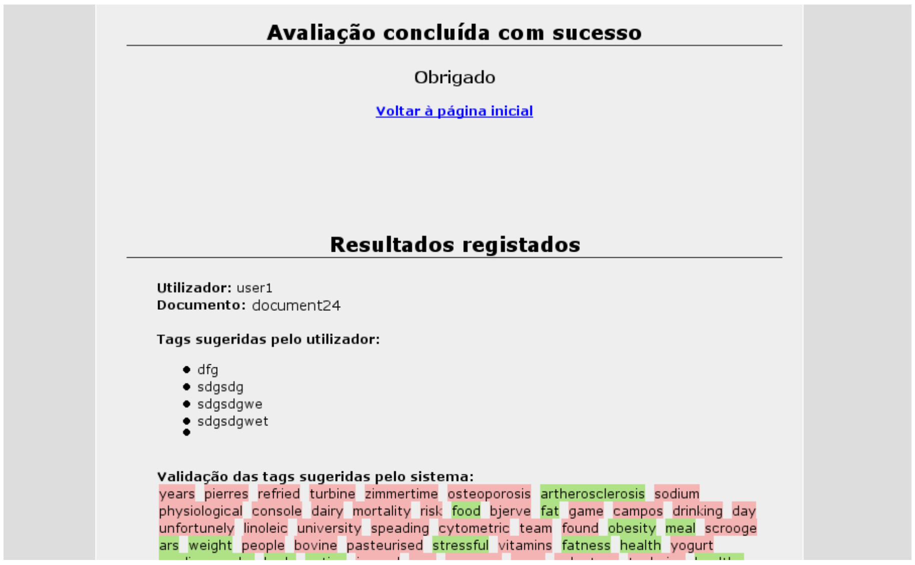
Common dieting advice suggests that as long as 'bad' foods are eaten in moderation and tempered with a host of healthy choices, they are not terribly bad for you. A new study published in this month's edition of the *Journal of Nutrition* demonstrates that a single high-fat meal in an otherwise healthy lifestyle causes a person to have a stronger adverse

abrir em nova pagina

years	dairy	unfortunely	ars
pierres	mortality	linoleic	weight
refried	risk	university	people
turbine	food	speading	bovine
zimmertime	bjerve	cytometric	pasteurised
osteoporosis	fat	team	stressful
artherosclerosis	game	found	vitamins
sodium	campos	obesity	fatness

Figure A.10: Evaluation's third step: rating the system tags

When a user finishes examining a document, the system displays his answers and he may go back to the menu to pick other documents to evaluate. This is illustrated in Fig. A.11.


Avaliação concluída com sucesso

Obrigado

Voltar à página inicial

Resultados registados

Utilizador: user1
Documento: document24

Tags sugeridas pelo utilizador:

- dfg
- sdgsdg
- sdgsdgwe
- sdgsdgwet
-

Validação das tags sugeridas pelo sistema:
years pierres refried turbine zimmertime osteoporosis artherosclerosis sodium
physiological console dairy mortality risk food bjerve fat game campos drinking day
unfortunely linoleic university speading cytometric team found obesity meal scrooge
ars weight people bovine pasteurised stressful vitamins fatness health yogurt

Figure A.11: User's answers for a given document

The evaluation procedure may terminate and give place to the analysis as soon as all the users finish examining the documents they were assigned to.

To run the analyzer tool, a directory named 'eval', must be created. The script uses the results gathered by the web application, as well as the data prepared by the setup tool and generates a series of charts and CSV (comma-separated values) files, which can easily be imported to a spreadsheets program.

Both the setup and the analyzer scripts were written in Python and are supposed to run using a command line interface. The web application was written in PHP, HTML and Javascipt. Its instructions are in Portuguese, since it is the native tongue of most evaluators.

B Complete Listing of the Evaluation Results

This appendix lists all the charts used to perform Tess evaluation. Only a few were included in Sec. 4.3 for organization purposes.

Figures B.1, B.2 and B.3 show the results for $precision1^+$, $precision2^+$ and $precision3$, respectively, when using the topic assigned by the story's submitter, that is, a user from *Digg*. Each bar represents a combination between a vector displacement method and a term extraction measure. Bars from the same vector displacement method are grouped and separated from the others to help distinguishing them.

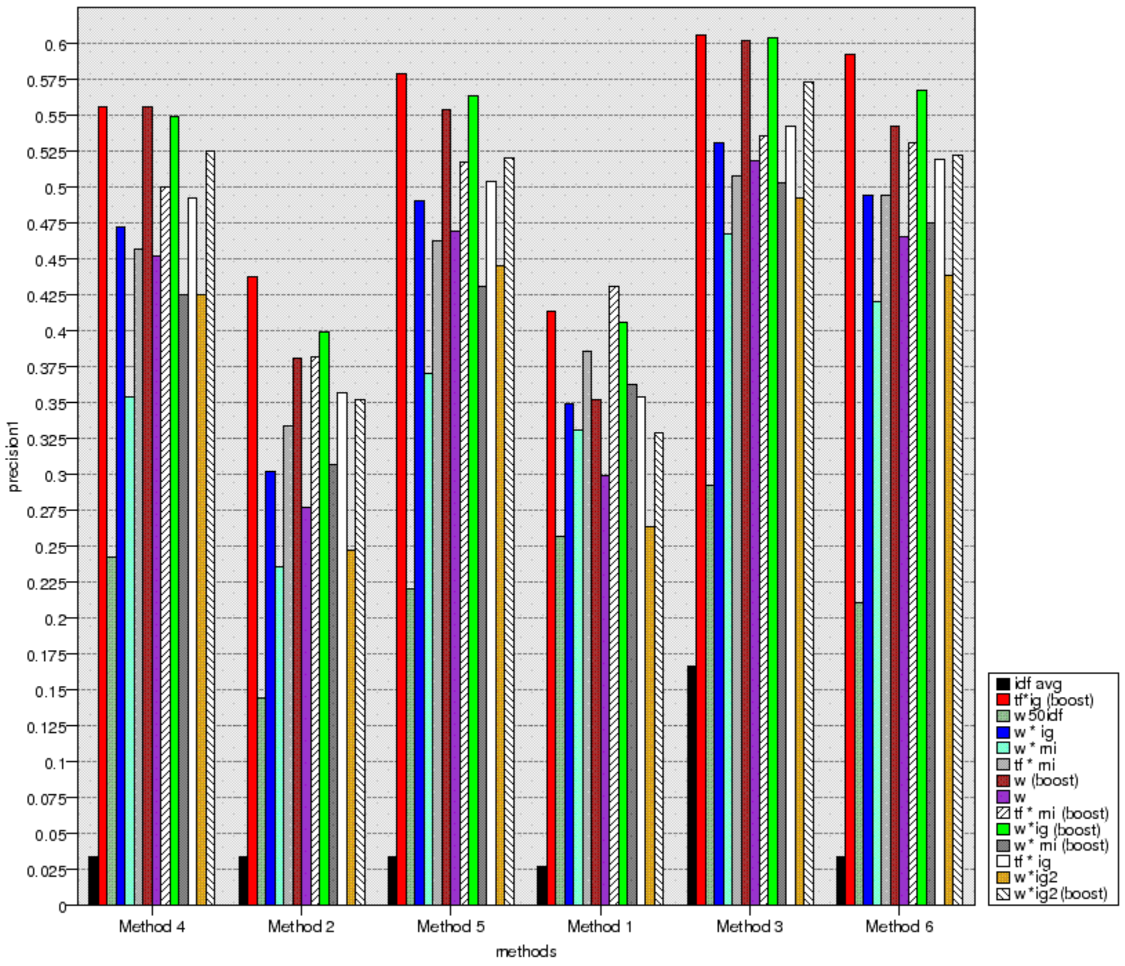


Figure B.1: Chart for $precision1^+$

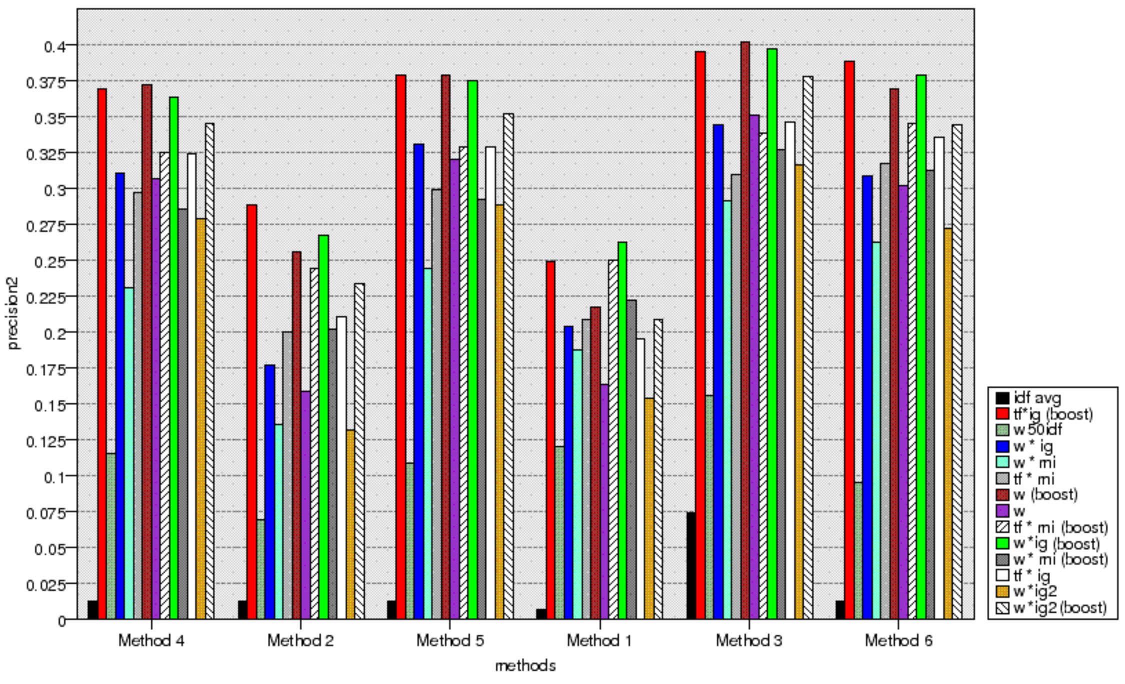


Figure B.2: Chart for $precision2^{+}$

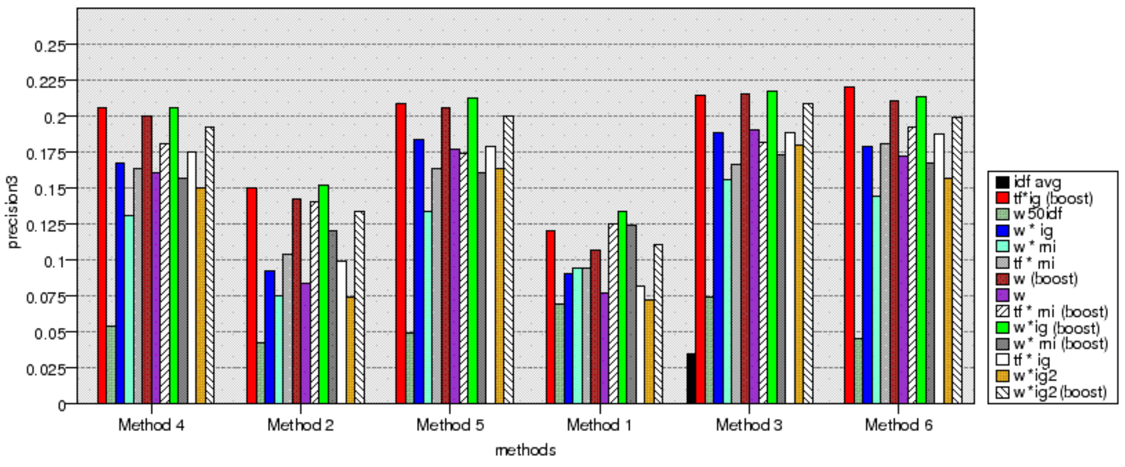


Figure B.3: Chart for $precision3$

The results for the same measures when using the topic assigned by Tess built-in categorizer are presented in Figures B.4, B.5 and B.6

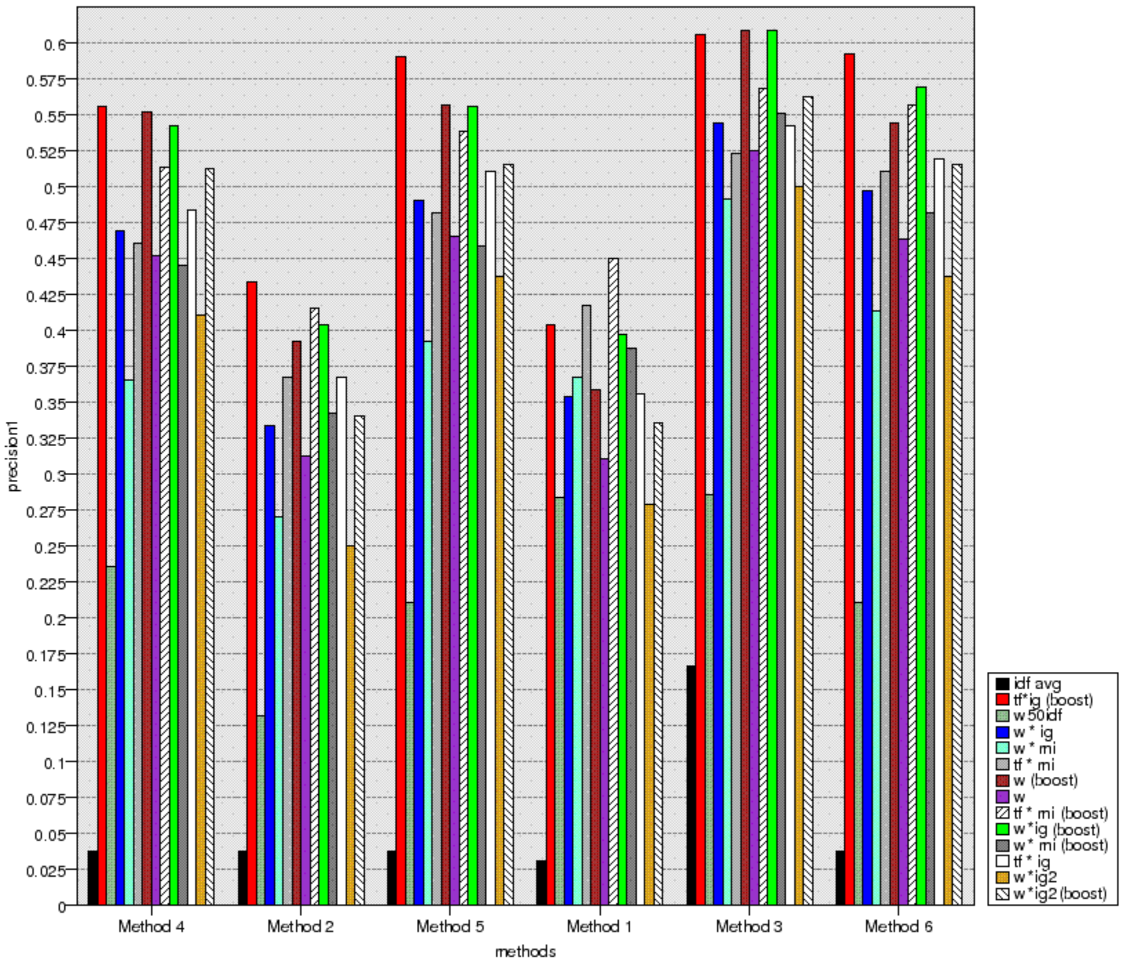


Figure B.4: Chart for $precision1^{+}$ when using Tess categorizer

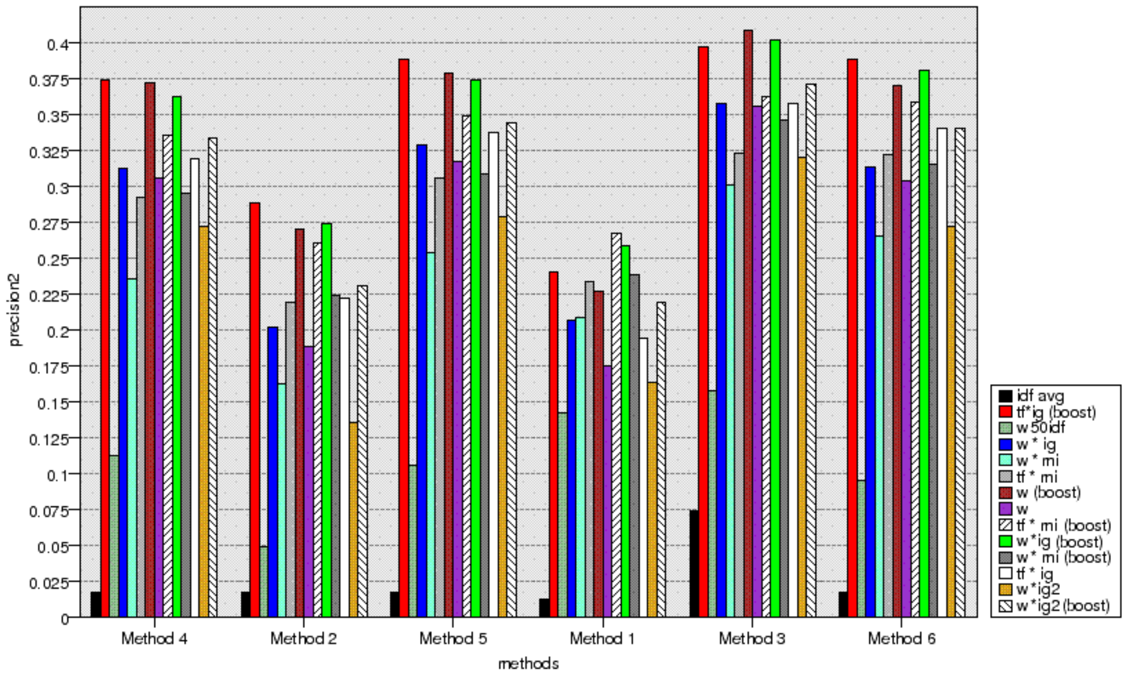


Figure B.5: Chart for $precision2^{+}$ when using Tess categorizer

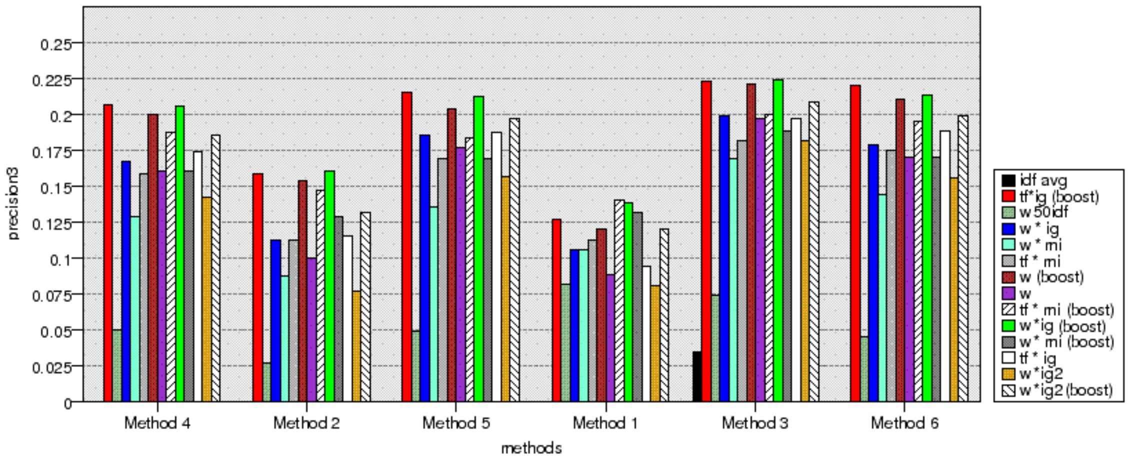


Figure B.6: Chart for $precision3$ when using Tess categorizer

The two following charts, Fig. B.7 and Fig. B.8, present the results for *coverage* when using *Digg*'s topic and when using the topic suggested by the categorizer, respectively.

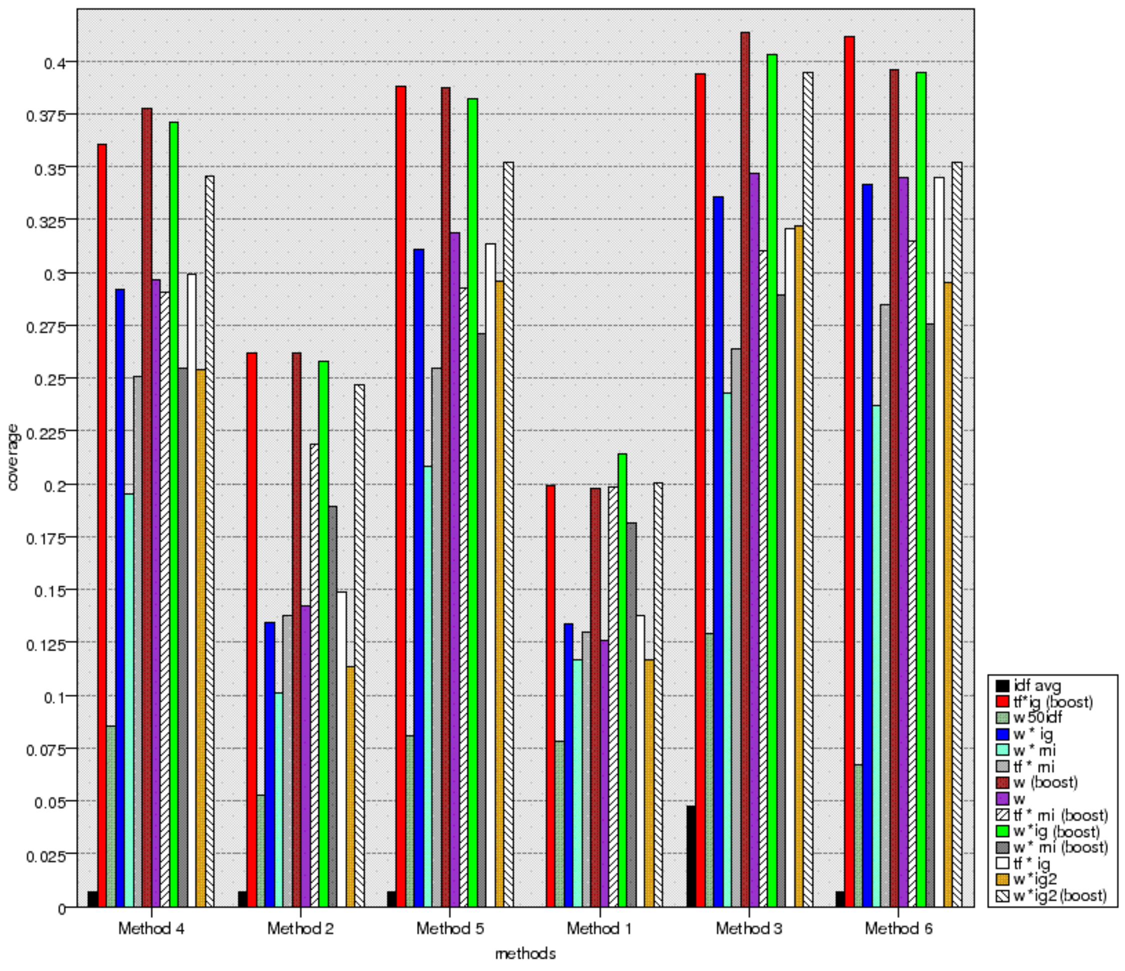


Figure B.7: Chart for *coverage*

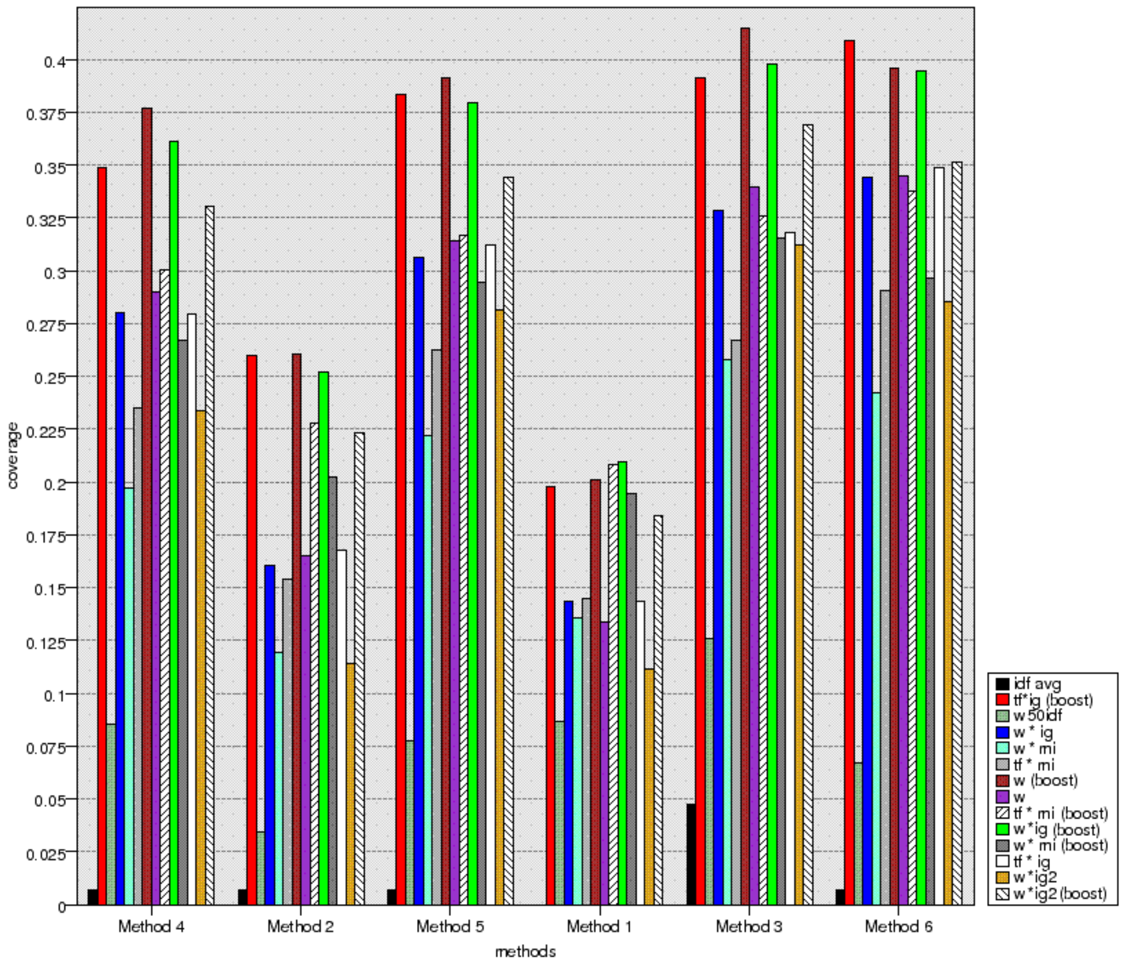


Figure B.8: Chart for *coverage* when using Tess categorizer

Next, the *precision* vs. *coverage* charts are presented in Figures B.9, B.10, B.11, B.12, B.13, B.14. Each plot corresponds to a vector displacement method. The left-most figures display the results when using the topic suggested by the user, whereas the ones on the right side display the results when using the topic suggested by Tess categorizer.

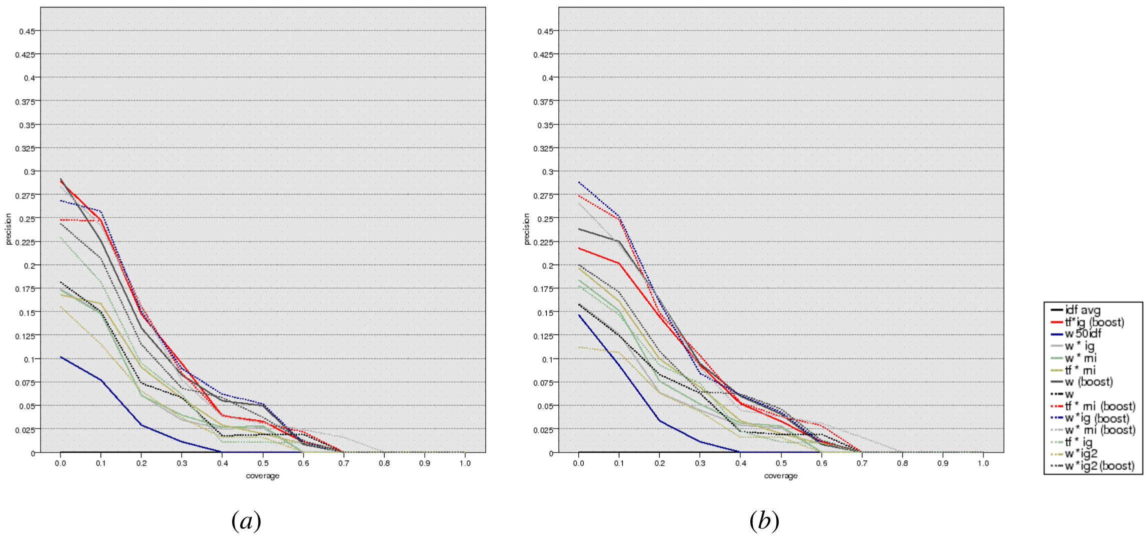


Figure B.9: Chart for *precisionC* vs. *coverage* for 'Method 1' categorized by the user (a) or by Tess (b)

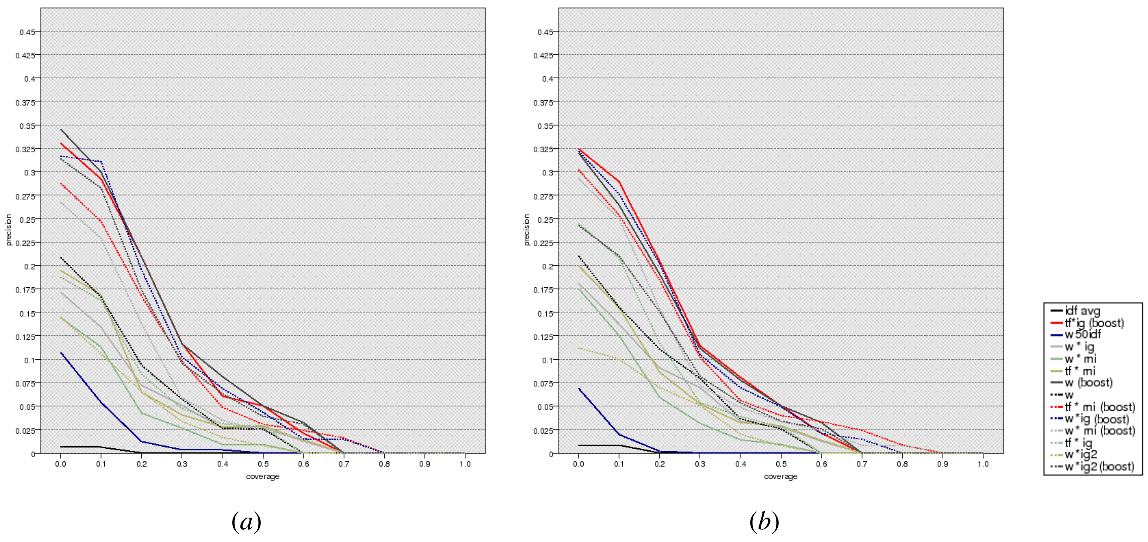
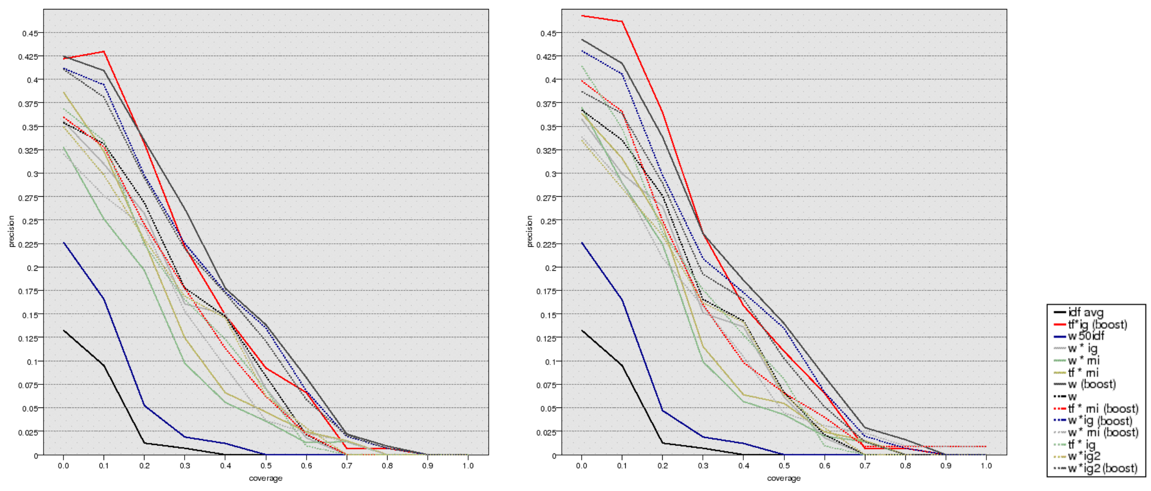
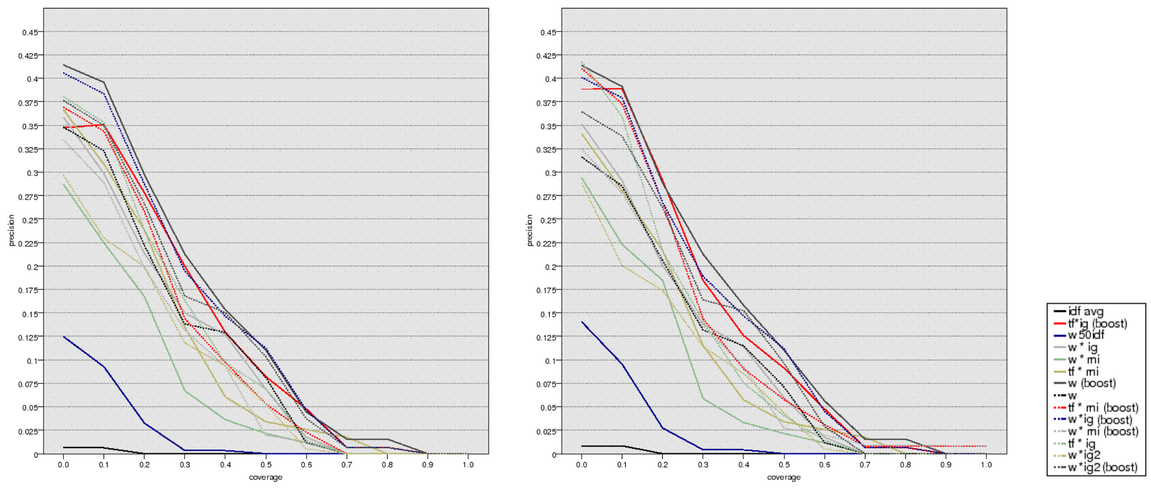


Figure B.10: Chart for *precisionC* vs. *coverage* for 'Method 2' categorized by the user (a) or by Tess (b)



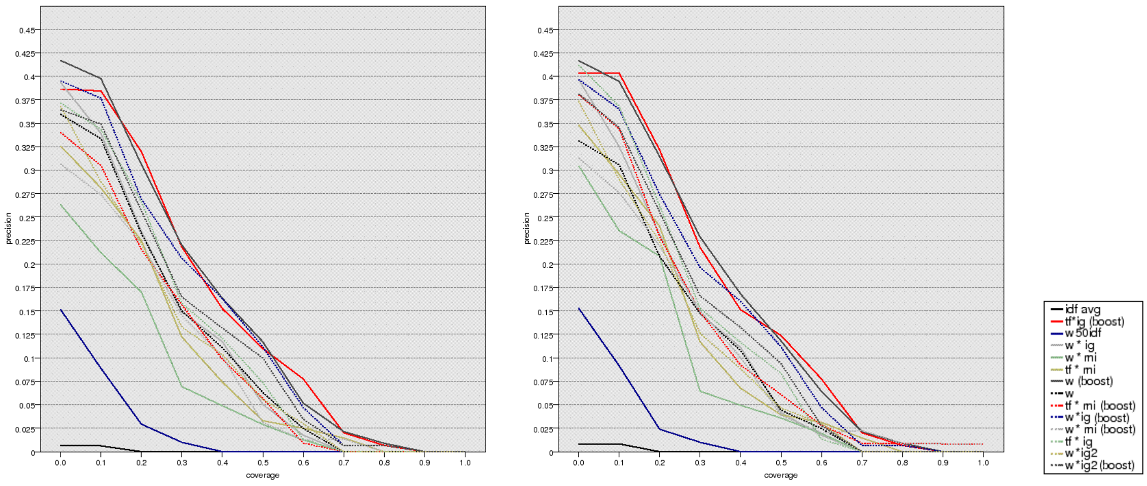
(*a*) (*b*)

Figure B.11: Chart for *precisionC* vs. *coverage* for 'Method 3' categorized by the user (a) or by Tess (b)



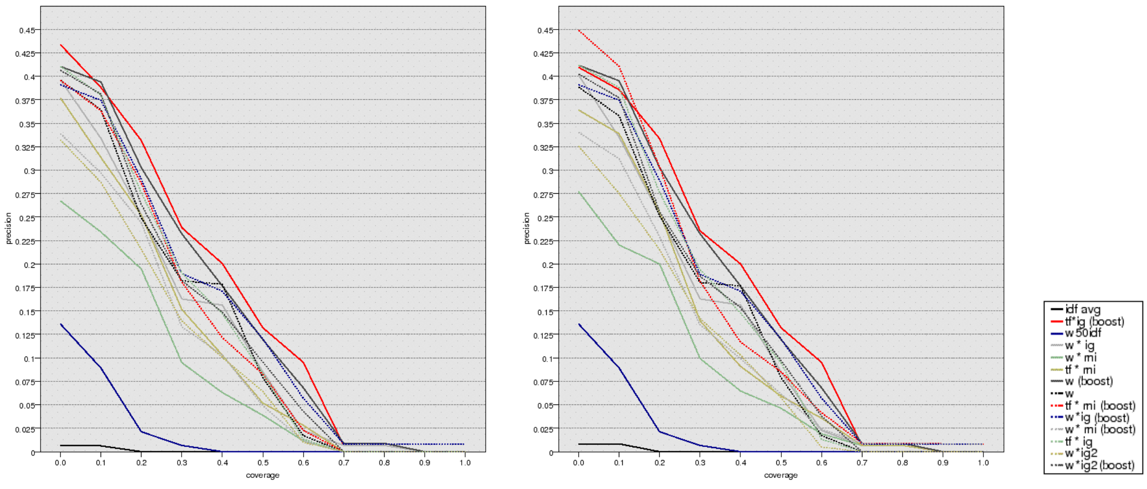
(*a*) (*b*)

Figure B.12: Chart for *precisionC* vs. *coverage* for 'Method 4' categorized by the user (a) or by Tess (b)



(*a*) (*b*)

Figure B.13: Chart for *precisionC* vs. *coverage* for ‘Method 5’ categorized by the user (a) or by Tess (b)



(*a*) (*b*)

Figure B.14: Chart for *precisionC* vs. *coverage* for ‘Method 6’ categorized by the user (a) or by Tess (b)

The two charts for *novelty* are presented next. Fig. B.15 displays the results when using the topic selected by *Digg*'s user, whereas Fig. B.16 corresponds to the *novelty* ratio when using the topic suggested by Tess categorizer.

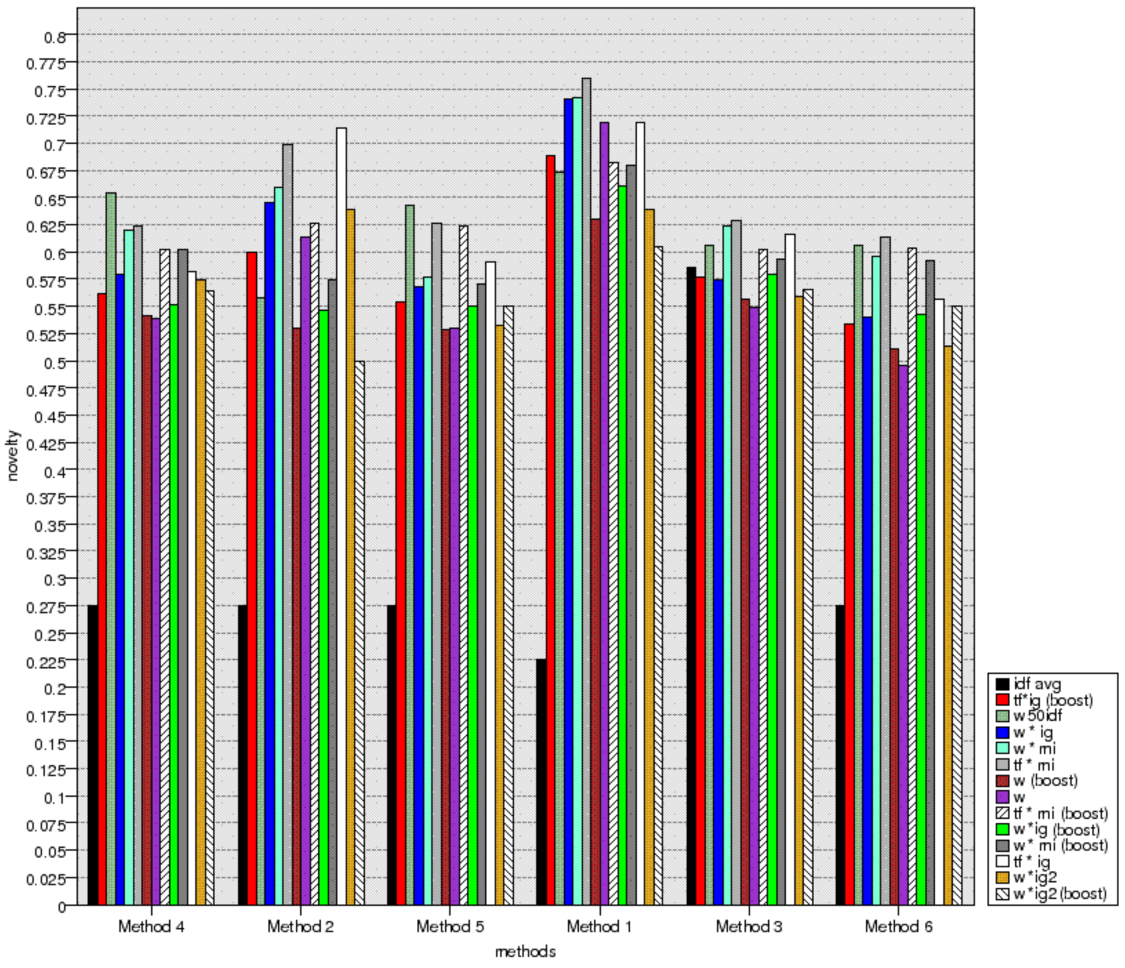


Figure B.15: Chart for *novelty* when using the topic proposed by the story's submitter

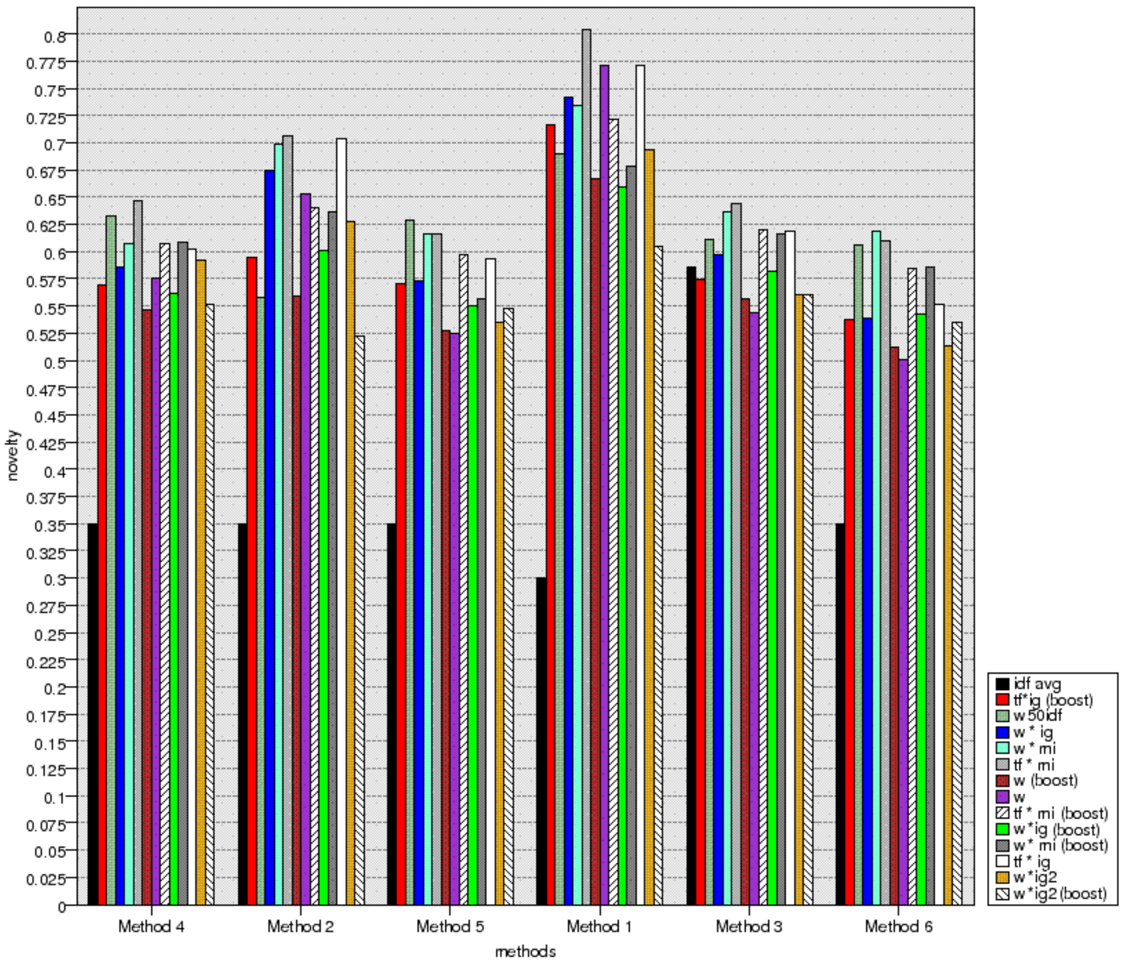
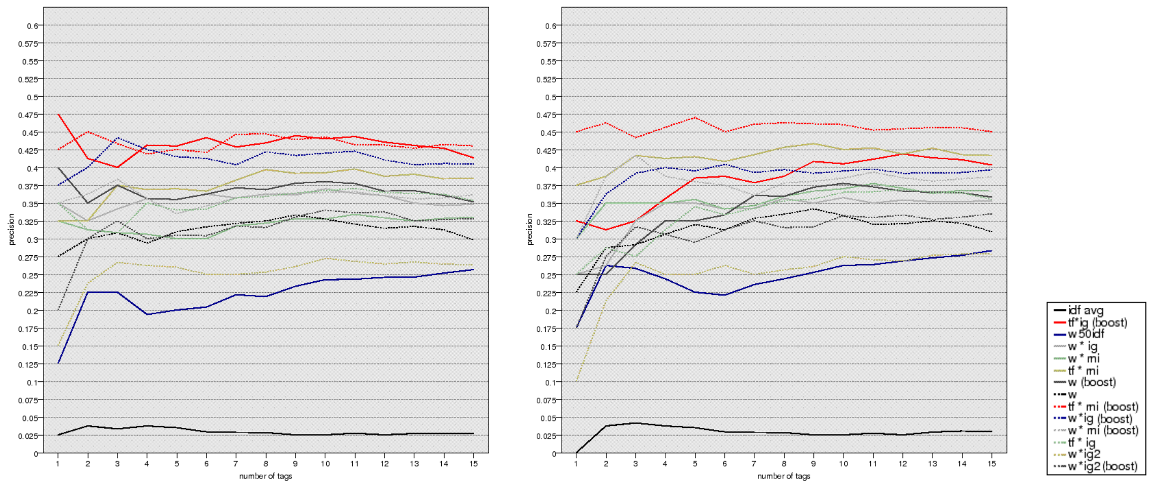


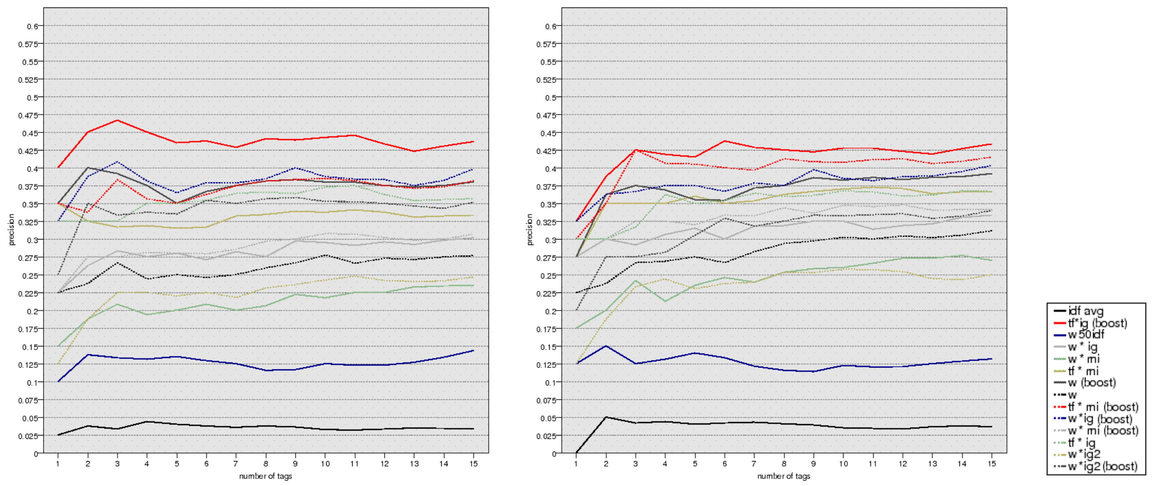
Figure B.16: Chart for *novelty* when using the topic proposed by the categorizer

Finally, the charts for *precision* vs. number of tags are presented in Figures B.17, B.18, B.19, B.20, B.21, B.22. Their disposition is similar to that of *precision* vs. *coverage* charts.



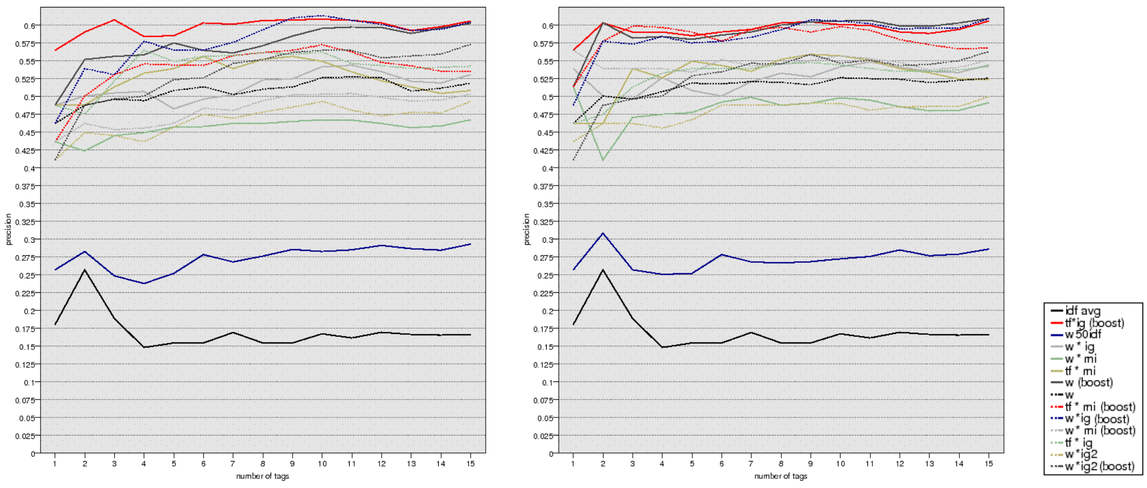
(*a*) (*b*)

Figure B.17: Chart for $precision1^{+}$ vs. number of suggested tags for ‘Method 1’ categorized by the user(a) or by Tess (b)



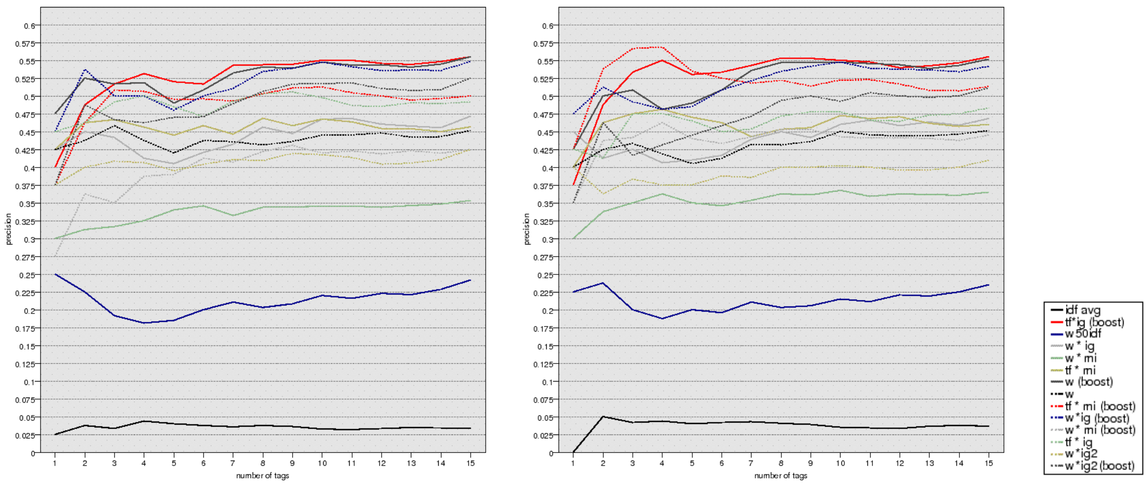
(*a*) (*b*)

Figure B.18: Chart for $precision1^{+}$ vs. number of suggested tags for ‘Method 2’ categorized by the user(a) or by Tess (b)



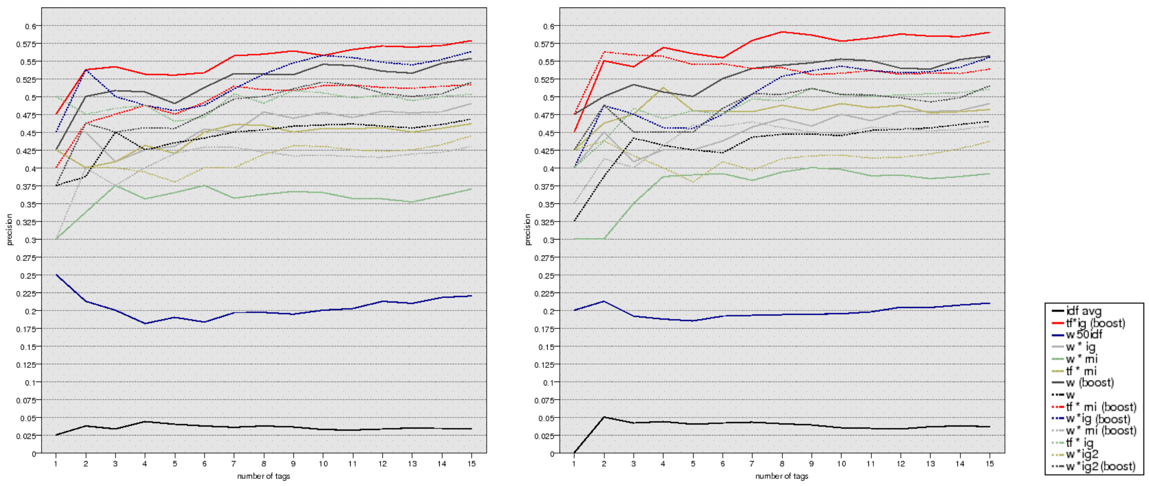
(*a*) (*b*)

Figure B.19: Chart for $precision1^{+}$ vs. number of suggested tags for 'Method 3' categorized by the user(a) or by Tess (b)



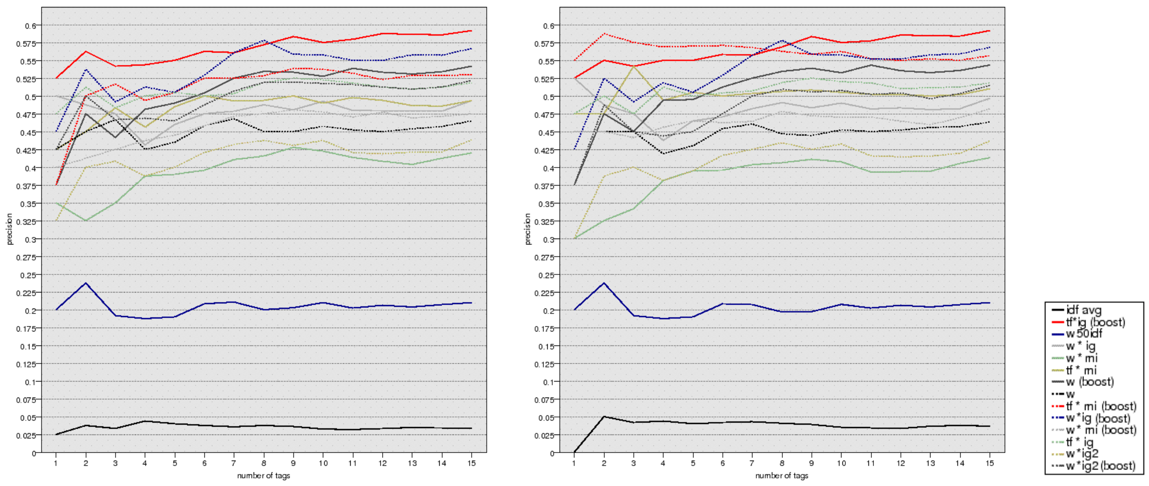
(*a*) (*b*)

Figure B.20: Chart for $precision1^{+}$ vs. number of suggested tags for 'Method 4' categorized by the user(a) or by Tess (b)



(*a*) (*b*)

Figure B.21: Chart for $precision1^{+}$ vs. number of suggested tags for ‘Method 5’ categorized by the user(a) or by Tess (b)



(*a*) (*b*)

Figure B.22: Chart for $precision1^{+}$ vs. number of suggested tags for ‘Method 6’ categorized by the user(a) or by Tess (b)

C Technologies and Libraries

This appendix provides a general overview on the languages, libraries and auxiliary tools used in this project.

C.1 Python

Python [1] is a popular multi-paradigm programming language. It is widely used in systems like *YouTube.com*, *Google* or *EVE Online*. It supports the functional, object oriented and imperative programming paradigms. Also it has garbage collection, which releases the programmer from the task of memory management. Usually, Python programs are interpreted, though there are some compilers available. The large community of Python users allows for the language to evolve. Several libraries are available and there are interpreters for every operating system. Thus, the task of writing complex applications or to port them to run on different systems is fairly simple.

This language was chosen to write some of the tools, mainly because of its ease of writing script-like code. Indeed, calling other programs and catching their output is rather straightforward. Contrarily to other languages for script writing, Python code is easier to maintain, as its readability is ensured by the syntax. Furthermore, since Python is a high-level language, programing gets simplified. The vast amount of helpful libraries is another compelling factor and increases Python's value.

C.2 SQLite

SQLite [2] is a simple database management system. It implements most features of the SQL-92 standard and guarantees the ACID (atomic, consistent, isolated and durable) properties of the transactions. Each database is stored in a single file, which makes it easier to move the data around. Besides, the library is relatively small and is available to many operating systems.

This library was used by the crawler, as it needed not to store much information and the queries on the data were somewhat simple. Since SQLite is written in C, a wrapper to Python, called pysqlite, was used. Such tool makes it possible to use the library as if it was written in Python.

C.3 Wget

GNU Wget [3] is a tool for retrieving files from web servers. It supports many popular Internet protocols for download, including HTTP, HTTPS and FTP. There are several built-in features to

[1] http://www.python.org/
[2] http://www.sqlite.org/
[3] http://www.gnu.org/software/wget/

help in the retrieval process, such as, recursive download, use of filename wildcards, conversion of absolute links to relative, support for proxies, among others. Furthermore, GNU Wget is free software and is available for many operating systems.

This tool was used by the crawler to simplify the downloading of files from websites, other than *Digg*'s. Since it can deal with several protocols and handle server-related issues, such as high response times or the number of times it should try to download a file before giving up, this tool can increase the crawler's robustness.

C.4 Java

Java [4] is a widely used programming language, created by Sun Microsystems. The language is imperative and is intended to write object-oriented code. The syntax is similar to that of C or C++, though, in Java, the programmer needs not to worry about memory management, as there is a built-in garbage collection mechanism. Usually, Java applications are compiled to bytecode and require a virtual machine to run. This intermediary version is, virtually, operating system independent. In fact, Java applications are, typically, highly portable.

This programming language was used in this project, mainly because the chosen framework for document indexing and processing was written in Java. There was a port to Python, the language that had been previously used to write the crawler, but it was not up to date and its documentation was not very good. The fact that Java applications are fairly easy to port to other operating systems, was considered as well.

C.5 Lucene

Apache Lucene [5] is a text search engine library, written in Java. Ports to other programming languages, such as C and C#, are also available. It is used in many popular applications, like *Eclipse* [6] and *Wikipedia*. Notable features include high-performance indexing, the existence of several search algorithms and some text processing capabilities, such as stemming, for instance.

The main reasons for choosing Lucene as the information retrieval library were its performance, the quality of its documentation and the indexing features, which transform the documents into data structures that can be easily manipulated by the algorithms.

[4]http://java.sun.com/
[5]http://lucene.apache.org/
[6]http://www.eclipse.org

C.6 PHP

PHP [7] is a programming language that is especially suited for the development of web applications. Hence, it is mainly used in server-side scripting. It supports the object-oriented paradigm and is an imperative language. PHP syntax is similar to Perl's. There are several libraries available for this language, which can be very helpful when programming complex applications. PHP code and HTML can coexist in the same page. Typically, the PHP code is embedded in the HTML and it can produce HTML code, making it easy to create dynamic webpages. Also it supports file reading and writing as well as database accesses.

The evaluation tool was written in PHP for it makes it is easy to adapt the content to each user, and both functional and interface code, that is, PHP and HTML code, can be placed in the same page. Also, the results can easily be stored for further analysis.

[7] http://www.php.net/