



Pós-Graduação em Ciência da Computação

“Tag Suggestion using Multiple Sources of Knowledge”

By

Ícaro Rafael da Silva Medeiros

M.Sc. Dissertation



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, JULY/2010



Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Ícaro Rafael da Silva Medeiros

“Tag Suggestion using Multiple Sources of Knowledge”

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

A M.Sc. Dissertation presented to the Federal University of Pernambuco in partial fulfillment of the requirements for the degree of M.Sc. in Computer Science.

Advisor: *Frederico Luiz Gonçalves de Freitas*
Co-Advisor: *Helena Sofia Pinto (INESC-ID Lisboa)*

RECIFE, JULY/2010

Medeiros, Ícaro Rafael da Silva

**Tag suggestion using multiple sources of knowledge /
Ícaro Rafael da Silva Medeiros - Recife: O Autor, 2010.
xxi, 102 folhas : il., fig., tab.**

**Dissertação (mestrado) Universidade Federal de
Pernambuco. CIn. Ciência da computação, 2010.**

Inclui bibliografia e apêndice.

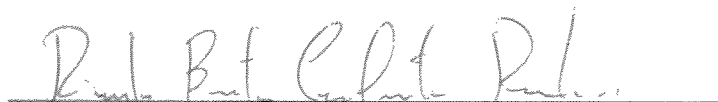
**1. Inteligência artificial. 2. Recuperação da informação. I.
Título.**

006.31

CDD (22. ed.)

MEI2010 – 0177

Dissertação de Mestrado apresentada por **Ícaro Rafael da Silva Medeiros** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Tag Suggestion using Multiple Sources of Knowledge**", orientada pelo **Prof. Frederico Luiz Gonçalves de Freitas** e aprovada pela Banca Examinadora formada pelos professores:



Prof. Ricardo Bastos Cavalcante Prudêncio
Centro de Informática / UFPE



Prof. Pável Pereira Calado
Instituto Superior Técnico - Portugal



Prof. Frederico Luiz Gonçalves de Freitas
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 19 de julho de 2010.



Prof. Nelson Souto Rosa

Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

*I dedicate this thesis to my family and friends.
Scientia vincere tenebras!*

Acknowledgements

To my family for all the support, funding, love and to bear living with me. Love you all!

I would like to thank Prof. Fred Freitas, my advisor, for all the lessons, patience and incentive. I will not forget the day when I was saying goodbye to you before going to Portugal and you said: “you remember me in my student times, very enthusiastic about research and innovation”.

To Prof. Helena Sofia Pinto and Prof. Pável Calado, my portuguese advisors, that have taught me how international, exigent, good, and state of the art research is. Thanks for all the wisdom, support, guidance, patience, and opportunities given.

Thanks to my IT colleagues Fernando Lins, Marcelo Siqueira (and all the SWORD — Semantic Web and Ontologies Research and Development — Knights), Mário Peixoto, Diogo Cabral, Marco Rosner, Ellison Leão, and Manoel Neto for the good R&D and programming advices.

Thanks to Hugo Estanislau, Heberth Braga, Thiago Araújo, and Lucas Cavalcanti, my roommates in Recife, “na característica”.

To Zózimo Neto, Daniel Martins, Maritza Moreira, and Frederico Moreira, thanks for all the great moments we had in Lisbon (nights) and when traveling through Europe.

To my closest friends, for all the support and care, even when I was far away. Special thanks to Larissa Cabús, Vanessa Cabral, Leonardo Almeida, Pablo Gomes, Diego Nunes, and Ítalo.

To AC/DC, Oasis, Wilco, and The Killers for the great concerts I was lucky to see in Lisbon.

To James Douglas Morrison, Friedrich Wilhelm Nietzsche, and Jean-Luc Godard, for the art and inspiration.

To everybody that made part of my life during these years. À todos, “aquele abraço”!

Eu nada amo mais profundamente do que a vida, e ainda mais quando a detesto. Se me inclino para a sabedoria e amiúde com excesso, é porque me lembra bastante a vida. Tem os seus olhos, o seu riso e até o seu dourado anzol. Que hei de fazer, se se parecem tanto as duas?

In my heart do I love only Life — and verily, most when I hate her! But that I am fond of Wisdom, and often too fond, is because she remindeth me very strongly of Life! She hath her eye, her laugh, and even her golden angle-rod: am I responsible for it that both are so alike?

—FRIEDRICH NIETZSCHE (Assim Falou Zarathustra, 1885)

Resumo

Nos sistemas de *tagging* social usuários atribuem *tags* (palavras-chave) a recursos (páginas Web, fotos, publicações, etc), criando uma estrutura conhecida como *folksonomia*, que possibilita uma melhora na navegação, organização e recuperação de informação. Atualmente, esses sistemas são muito populares na Web, portanto, melhorar sua qualidade e automatizar o processo de atribuição de tags é uma tarefa importante. Neste trabalho é proposto um sistema que automaticamente atribui tags a páginas, baseando-se em múltiplas fontes de conhecimento como o conteúdo textual, estrutura de hiperlinks e bases de conhecimento. A partir dessas fontes, vários atributos são extraídos para construir um classificador que decide que termos devem ser sugeridos como *tag*.

Experimentos usando um dataset com *tags* e páginas extraídas do Delicious, um importante sistema de *tagging* social, mostram que nossos métodos obtém bons resultados de precisão e cobertura, quando comparado com tags sugeridas por usuários. Além disso, uma comparação com trabalhos relacionados mostra que nosso sistema tem uma qualidade de sugestão comparável a abordagens estado da arte na área. Finalmente, uma avaliação com usuários foi feita para simular um ambiente real, o que também produziu bons resultados.

Palavras-chave: Sugestão de tags, folksonomias, mineração de texto, recuperação de informação, aprendizagem de máquina, descoberta de conhecimento.

Abstract

In social tagging systems users are allowed to assign tags (keywords) to resources (Web pages, photos, publications, etc), which can be used to enhance content navigation, organization and retrieval. Nowadays, these systems are very popular on the Web, so improving their quality and automating the process of tag assignment is an important task. In this thesis we propose a system that automatically assigns tags to pages based on multiple sources of knowledge like text, hyperlink structure and knowledge databases. From these sources a lot of features are extracted to build a classifier which decides terms to be suggested as tags.

Experiments on a Web tagging dataset extracted from Delicious, an important social tagging system, show that our method obtains good precision and recall results, when compared to the tags suggested by human users. Besides, a comparison with related works shows that our system has a suggestion quality comparable to state of the art approaches. Finally, a user-based evaluation was performed to simulate a real environment, which has also produced good results.

Keywords: Tag suggestion, tagging, folksonomy, information retrieval, machine learning, text mining.

Contents

Acronyms	1
1 Introduction	3
1.1 Goals	7
1.2 Structure	9
2 Related Work	11
2.1 Social Tagging Systems	11
2.1.1 Delicious	13
2.1.2 Classifying Social Tagging Systems	17
2.2 Folksonomies	18
2.2.1 Folksonomies and Semantics	20
2.2.2 Tagging Behavior	21
2.3 Information Retrieval	22
2.4 Machine Learning	25
2.4.1 Support Vector Machine	27
2.4.2 Classifier Evaluation	29
2.4.3 Combining Classifiers	31
2.5 Keyword Extraction	32
2.6 Tag Suggestion	33
2.6.1 Tag Suggestion Approaches	34
2.6.2 Machine Learning Approaches	35
3 ANTaReS	
A Tag Suggestion System	37
3.1 System Summary and Architecture	37
3.2 Datasets	39
3.3 Training Phase	40
3.4 Feature Extraction	41
3.4.1 Information Retrieval Features	42
Term Frequency	42
Inverse Document Frequency	42
Term Frequency-Inverse Document Frequency	42
3.4.2 HTML Features	42

	Title	43
	Keywords	43
	Description	43
	Emphasizing Tags	43
	Text in Anchors	43
3.4.3	Linguistic Features	44
	Part-of-speech Tag	44
	Average Length of the Sentence	44
	Sentence Co-occurrence of Terms	44
3.4.4	URL Features	45
	Term Part of the URL	45
3.4.5	Text Features	45
	First Occurrence in the Document	45
	Capital Letters	45
3.4.6	Inbound Pages Features	46
	Inbound Link Anchors	46
	Inbound Link Context	47
	Inbound Page Content	47
3.4.7	Wordnet Features	48
	Frequency of Occurrence	48
	Tag Count	48
	Overlap Count	48
3.5	Classifier Combination	49
3.6	Suggestion Phase	49
3.7	Implementation	51
3.8	User-based Evaluation System	54
4	Experiments	57
4.1	Extraction of the Datasets	57
4.1.1	Web Pages Dataset	58
4.1.2	Inbound Pages Dataset	59
4.1.3	Wordnet Dataset	60
4.2	Tag Source Analysis	61
4.3	System Evaluation	65
4.3.1	Impact of Features	65
4.3.2	Combination of Classifiers	69

4.3.3	Comparing with other Machine Learning Methods	71
4.3.4	Comparing with Related Work	71
4.4	User-based Evaluation	73
4.4.1	Results	75
4.5	Discussion	76
5	Conclusion	79
5.1	Future Work	80
5.2	Final Words	81
	Bibliography	83
	Appendices	91
A	Apache Lucene	91
A.1	Lucene Index	91
A.2	Important Classes for Index Creation and Access	92
Document	92
Field	92
Term	92
Analyzer	92
Directory	93
IndexWriter	93
IndexReader	93
TermFreqVector	94
B	WEKA	94
B.1	Building Feature Vectors	94
B.2	ARFF File Format	95
B.3	Training a Classifier	96
B.4	Saving and Loading Classifiers	97
B.5	Testing a Classifier	97
B.6	Using a Classifier	97
C	GATE	98
C.1	Main Classes Used in ANTaReS	98
GATE	98
Document	98
AnnotationSet	98

	Annotation	99
	AnnotationUtils	99
D	Detailed Experiment Results	99
D.1	One Web Page Feature Used in Isolation	99
D.2	Adding Web Page Features to the Set of Features	99
D.3	Removing a Single Feature from the Full Set of Web Page Features	100
D.4	Orders of Precedence in Majority Voting	101
E	Comparing ANTaReS with Heymann et. al (2008b)	102

List of Figures

1.1	Bibsonomy main page.	3
1.2	Delicious main page.	4
1.3	Example of binded words and synonyms in Delicious.	5
1.4	Example of binded words in Bibsonomy.	6
1.5	Example of a misplaced tag: “coffee” in a page about Java language. . .	6
2.1	Conceptual model of tagging systems (Marlow <i>et al.</i> , 2006).	12
2.2	Popular bookmarks on Delicious.	13
2.3	Exploring tags with Delicious.	14
2.4	Main menu of Delicious.	14
2.5	Bookmarks summary of a user in Delicious.	15
2.6	Saving a bookmark in Delicious.	16
2.7	Using a Firefox extension to use Delicious.	17
2.8	Graphic representation of the SCOT ontology.	20
2.9	Representing a text document using the Vector Space Model (VSM). . .	23
2.10	Measuring cosine similarity between documents.	24
2.11	Example of inverted index.	25
2.12	Example of how to construct a Support Vector Machine (SVM) classifier (Wikipedia, 2010).	27
2.13	Formal example of how to construct a SVM classifier (Wikipedia, 2010). .	29
2.14	Example of a confusion matrix, used to test classifiers (Wikipedia, 2009). .	30
2.15	Combining classifiers with different feature spaces.	31
2.16	Tag suggestion process.	33
3.1	System architecture of ANTaReS.	38
3.2	Example of inbound pages.	39
3.3	The process of training ANTaReS classifiers.	40
3.4	Example of inbound link anchors.	46
3.5	Example of an inbound link context.	47
3.6	Representing sparse examples from different sources in the same feature space and the “confusion” of the SVM algorithm.	50
3.7	The process of ANTaReS tag suggestion (and testing).	50
3.8	Packages and databases of ANTaReS.	52
4.1	Process of extraction of the Delicious tagging dataset.	58

4.2	Process of extraction of inbound pages.	60
4.3	Process of extraction of Wordnet related terms.	61
4.4	Example of tags versus information source analysis for page d_1	63
4.5	Precision results in the user-based evaluation.	75
4.6	Coverage results in the user-based evaluation.	76
4.7	Novelty results in the user-based evaluation.	76

List of Tables

4.1	Membership relations of tags described in Figure 4.4.	63
4.2	Analysis of the tags found in the three knowledge sources.	64
4.3	Web pages features used in isolation (Summary). Complete version in Table D.2.	66
4.4	Wordnet features used in isolation.	67
4.5	Adding each single Web page feature to the set of features.	67
4.6	Removing a single Inbound page feature from the full set of features. . .	68
4.7	Removing a single Wordnet feature from the full set of features.	69
4.8	Results for the classifiers focused in one source.	69
4.9	Results for the combined classifiers.	70
4.10	Experiment with different Machine Learning (ML) methods for classification.	72
4.11	Most popular tags in the dataset used in Song <i>et al.</i> (2008b).	72
4.12	Comparing ANTaReS with Song <i>et al.</i> (2008b) and Song <i>et al.</i> (2008a). .	73
4.13	Summary of user-based evaluation results	77
B.1	An example of an ARFF file.	95
D.2	Web pages features used in isolation (Complete).	100
D.3	Adding each single Web page feature to the set of features.	100
D.4	Removing a single Web page feature from the full set of features.	101
D.5	Results for different orders of precedence for majority vote.	101
E.6	Results for the experiment comparing ANTaReS with Heymann <i>et. al</i> (2008b).	102

Acronyms

STS Social Tagging System

IR Information Retrieval

ML Machine Learning

TF-IDF Term Frequency-Inverse Document Frequency

TF Term Frequency

IDF Inverse Document Frequency

SVM Support Vector Machine

VSM Vector Space Model

AI Artificial Intelligence

POS Part-Of-Speech

NLP Natural Language Processing

RBF Radial Basis Function

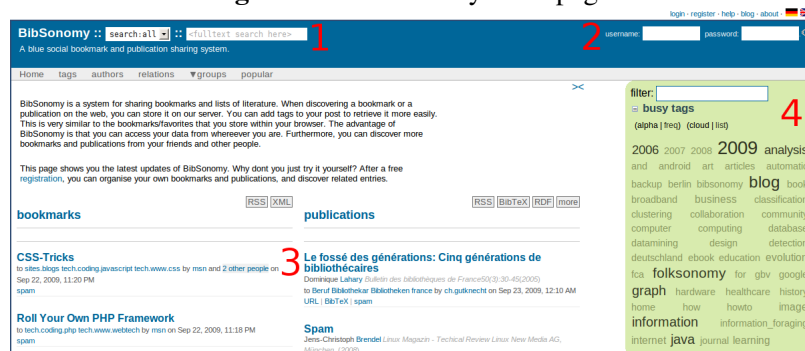
1

Introduction

Social Tagging Systems (STS) are popular applications on the Web, where users can assign tags to resources such as Web pages, photos, videos, music, artists and publications. These tags are freely chosen keywords, used to describe and organize content. Successful websites like Delicious, Flickr, Youtube, Last.FM and Bibsonomy¹ are examples of such systems.

An example of Social Tagging System (STS) is shown in Figure 1.1: the Bibsonomy system allows users to tag bookmarks and publications. In the figure we see points of access of important features like search (1), “login” area (2), recent bookmarks or publications uploaded (3), and popular tags, tag browsing, or filtering (4).

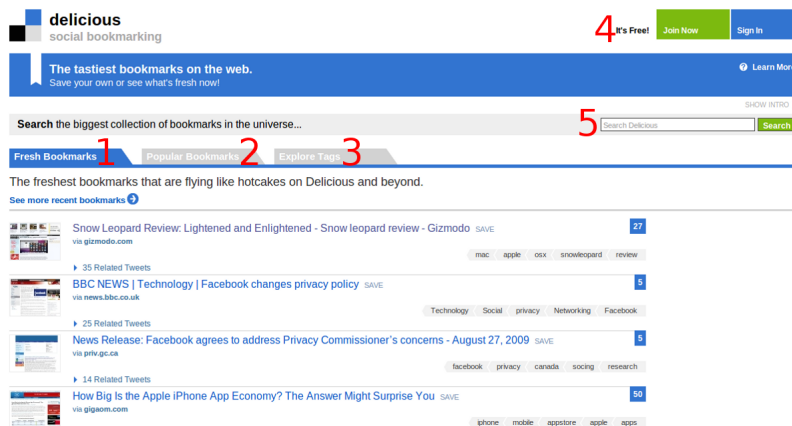
Figure 1.1 Bibsonomy main page.



Another example is Delicious, where users can upload Web pages and classify them using tags. This system also has a lot of interesting features like recent (1) and popular bookmarks (2) or tag browsing (3) (Figure 1.2). Also accessible through Delicious main page shown in Figure 1.2 is the “join” and “sign in” area (4) and the search feature (5).

¹www.delicious.com, www.flickr.com, www.youtube.com, www.last.fm and www.bibsonomy.org

Figure 1.2 Delicious main page.



One of the advantages of social tagging is that users can assign appropriate tags for a resource without a vocabulary restriction like when using a predefined lexicon or hierarchy (Mathes, 2004). It is argued that these hierarchical classification schemes such as dmoz Open Directory or Yahoo! directory² are too rigid and inflexible to classify Web data and a better approach is to allow content producers and content consumers (e.g., in blogs: bloggers and readers) to classify their data in a non-hierarchical way (Brooks and Montanez, 2005).

Therefore, tags are appropriate for the Web and user contributed resources where the distribution or type of content may change rapidly (Heymann *et al.*, 2008b). The use of tags as metadata, whose semantics emerges from its community, creates a bottom-up classification known as folksonomy (Mathes, 2004).

In fact, there are claims that the semantics emerging from a folksonomy is more accurate than when defined by a committee or a single person, and since content creation is decentralized, its description (classification) should be decentralized as well. Moreover, a shared pool of tagged resources enhances the metadata for all users, potentially distributing the workload for metadata creation amongst many contributors (Marlow *et al.*, 2006).

Some authors consider folksonomies as lightweight knowledge representation schemes. The main difference to traditional ontology engineering approaches is that users do not need to deal with any formal model. The widespread use of STSs shows that folksonomies are able to overcome (or reduce) the knowledge acquisition bottleneck, well known in knowledge-based systems (Hotho *et al.*, 2006a). However, to achieve this, folksonomies must have associated semantics, which is discussed in some works like (Mika, 2007) and

²Available at www.dmoz.org and dir.yahoo.com

(Jäschke *et al.*, 2008).

While in many of these applications the primary goal of these tags is to serve the needs of individual users, i.e. to organize and retrieve resources in personal bookmark collections, these tags also help other users to browse, categorize, organize and find items. Therefore, tags are useful for tasks such as information discovery, information sharing, and information extraction (Suchanek *et al.*, 2008).

Furthermore, tags can also be used in many other tasks, such as to measure the relevance of terms for Web search (Heymann *et al.*, 2008a), query expansion (Wang and Davison, 2008), automatic creation of metadata (Junichiro *et al.*, 2004), Web page annotation (Brooks and Montanez, 2006; Chirita *et al.*, 2007), and ontology learning (Mika, 2007).

However, one of the major advantages of tagging, can also be one of its drawbacks: the lack of structure and unrestricted vocabulary. Some problems in STSs include (Rader and Wash, 2006; Marlow *et al.*, 2006):

- Polysemy: when a word (tag) has multiple related meanings. For example, some dictionaries define *whistle* as the sound produced by a *whistle*. Therefore, *whistle* has two related meanings: the sound *whistle* and the device *whistle* that produces this sound (Leacock and Ravin, 2000);
- Homonymy: when a single word has different unrelated meanings. For example, “Java” can refer to the programming language, the island, or the Java coffee. In Figure 1.1 we see that, in Bibsonomy, “Java” is a frequent tag and must refer (in most cases) to the programming language but the folksonomy cannot represent this semantic association;
- Synonymy: when different words have the same meaning (e.g., “buy” and “purchase”). An informal example is “Web page” and “website”, often used to mean the same thing. The screenshot of top tags of Delicious in Figure 1.3 shows two synonyms: “howto” and “tutorial”, marked in red;

Figure 1.3 Example of binded words and synonyms in Delicious.

A screenshot of a list of tags from the Delicious bookmark service. The tags are arranged in four rows. The first row contains: design, blog, video, software, tools, music, programming. The second row contains: webdesign, reference, tutorial, art, web, howto, javascript, free, linux. The third row contains: web2.0, development, google, inspiration, photography, news, food. The fourth row contains: flash, css, blogs, education, business, technology, travel, shopping, books, mac, tips, politics, science, opensource, games, culture. The words 'tutorial' and 'howto' in the second row are each enclosed in a red rectangular box. The word 'web2.0' in the third row is enclosed in a blue rectangular box. The word 'opensource' in the fourth row is also enclosed in a blue rectangular box.

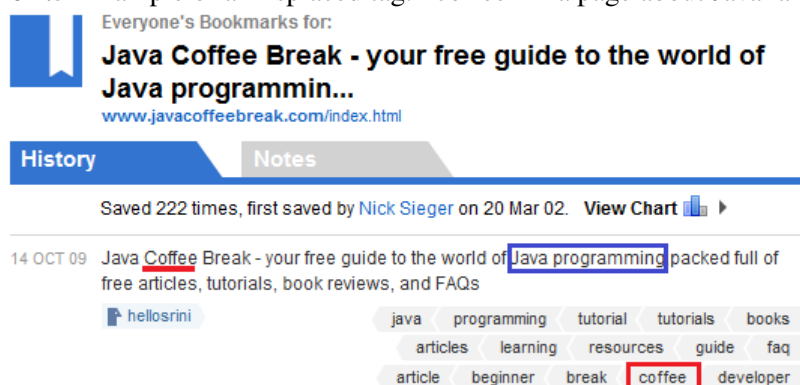
- **Bound words:** given that some **STSs** only work with simple words some users assign tags by binding words like in “semanticweb”, “semweb”, “semantic_web”, “webdevelopment”, or “webdev”. The screenshot of Bibsonomy in Figure 1.4 shows related tags to “semweb” tag with some examples about this issue. The screenshot of Delicious (Figure 1.3) also shows bound words like “web2.0” and “opensource”, marked in blue;

Figure 1.4 Example of binded words in Bibsonomy.



- **Misplaced tags:** users can also assign tags that do not classify the resource correctly. In Delicious (Figure 1.5) we found an example of a website called “Java Coffee Break” which in fact is about Java programming but the tag “coffee” was assigned to it, which leads to ambiguity between the two senses of “Java” (homonyms);

Figure 1.5 Example of a misplaced tag: “coffee” in a page about Java language.



- **Tags in different languages:** almost all **STSs** have no restriction upon language, so it is common to find different words that represent the same concept but in different languages;

- Different levels of abstraction: different expertise and purposes of users may result in tags that use various levels of abstraction to describe a resource. For example, a resource can be tagged as “cat”, “animal”, or “Persian cat”;
- Use of “selfish” tags like “toread”, “mustread”, or “reference” for personal organization.

These problems may impair the quality of the folksonomy and, therefore, all the advantages aforementioned. Moreover, tagging is still a manual process and some users may not be willing to engage in such a chore.

To improve the quality of folksonomies and to help users in the process of tag assignment, some websites include a tag suggestion mechanism. This mechanism usually works by, given a resource, suggesting the tags that were most frequently assigned to it by other users. However, this process is only effective if the resource has already been tagged by other users. Also, it somewhat impairs the emergence of novel tags, given that users will tend to choose tags that were already used.

1.1 Goals

Tag suggestion is an important feature of [STSs](#) and can lead to vocabulary convergence, folksonomy enrichment, tag disambiguation, among other advantages ([Heymann et al., 2008a](#)). Therefore, tag suggestion can mitigate the aforementioned disadvantages that happen due to folksonomy lack of structure and unrestricted vocabulary.

For example, suppose we want to disambiguate page A and page B associated with tag “Java” meaning respectively “language” and “coffee”. A tag suggestion system can recommend the tags “programming”, “language”, and “development” to the first page. The tags “coffee” and “island” can be associated with the second page, reducing ambiguity.

In systems like Delicious where tags are associated with pages several times synonyms can “disappear” due to suggestion when considering only the tags most frequently associated. So if one synonym is more used than others the vocabulary will converge towards it.

Another advantage regarding synonyms is recall increasing. For example, if the tag suggestion system can associate “tutorial” and “howto”, in resources assigned with “howto” users will be suggested to use “tutorial”, making the resource also retrievable

by using the term “tutorial” in a query. Similarly, other problems mentioned like binded words, misplaced tags, among others, can be reduced with suggestion.

The main goal of our work is to design and develop a tag suggestion system for Web pages, called ANTaReS — which stands for A Novel TAg Recommendation System. Some of the requirements for ANTaReS include:

- To suggest relevant tags to Web pages regardless the existing folksonomy, i.e. information about previously assigned tags to this resource or other information about the folksonomy are not be available. ANTaReS must be able to suggest new unforeseen tags, which is different from suggestion in Delicious, where the most popular tags assigned to a resource are suggested;
- To be adaptable to any STS, i.e. the system can not rely on any STS-specific feature;
- The system must be evaluated using a real tagging dataset as a baseline to validate our approach of suggestion. A comparison with related work and a user-based evaluation must also be performed.

To accomplish this and fulfill the requirements, some specific sub-goals were necessary during project development:

- To do an overview of the context in which tag suggestion systems are included, i.e. to study how STSs and folksonomies work, how to extract information from them, and the main research concerns about the topic. This analysis gives us a better understanding of how to design our software to fit STSs;
- Other objective is to study and analyze the approaches for tag suggestion proposed so far in literature. Therefore we could base our design and implementation choices on previous failures or successes found in these works. To understand, reuse, and adapt successful solutions is mandatory;
- An important task is to understand tag suggestion by analyzing different kinds of input data in order to infer their tag “richness” (i.e. to find the quantity of good tags that can be extracted from them). To accomplish this, a real tagging dataset was compared to some sources often used for extracting information in Web pages;
- To provide a detailed explanation of ANTaReS architecture, implementation, main processes, and crucial features;

- To validate ANTaReS with standard measures of classification systems, comparing our suggestions to tags assigned by users in a real folksonomy extracted from Web and discussing the results;
- To validate our system by comparing it to state of the art related work in tag suggestion;
- To perform and discuss results of a user-based evaluation, where students and researchers are invited to evaluate our suggestions.

Our system is different from previous approaches such as (Sood *et al.*, 2007; Heymann *et al.*, 2008b; Lipczak *et al.*, 2009; Wang and Davison, 2008; Song *et al.*, 2008a) because we use a significant large number of features to describe terms and several sources of knowledge are considered to discover relevant tags. Other different approaches are also cited through the text.

In short, in this thesis we describe ANTaReS, a tag suggestion system for Web pages. To describe it, we walk through the steps of development from early analysis of related work to discussing results and future improvements. In all steps we explain the rationale of our design choices and compare our system to previous approaches.

1.2 Structure

This thesis is divided in 5 chapters. Chapter 1 introduces the concepts about Social Tagging System (STS), folksonomies, and tag suggestion. In Chapter 2, we provide a detailed discussion about STSs and folksonomies. Information Retrieval (IR) and Machine Learning (ML) techniques frequently used in similar text mining tasks such as tag suggestion and specifically used in ANTaReS are also subject of this chapter. Finally, tag suggestion approaches are presented.

Chapter 3 starts by providing an overview of ANTaReS main features and its architecture. In the following sections, important parts of the system such as the dataset extraction, feature extraction, and classifier building are explained and discussed in detail. Then, in Chapter 4, we describe the experiments made to evaluate our approach. The results are also presented and discussed.

Finally, in Chapter 5 we draw some conclusions and the future work is defined. After the bibliography we also include some appendices, which contain information about programming libraries used and detailed experiment results.

2

Related Work

In this chapter we define what tag suggestion is and review the most relevant research work in this area. First we describe what Social Tagging Systems (STS) are and folksonomies, the context in which tag suggestion systems are applied. We also review Information Retrieval ([IR](#)) and Machine Learning ([ML](#)) techniques that are being used in ANTaReS and tag suggestion systems in general.

2.1 Social Tagging Systems

Resource sharing systems are applications on the Web where users can upload and share content like photos (Flickr), videos (Youtube), bookmarks (Delicious), etc. In order to organize users' uploads, these systems encourage users to classify their submissions through a tagging system where keywords are used to describe uploaded content. A difference between other keyword-based organization systems is that [STSs](#) do not rely on controlled vocabularies ([Marlow *et al.*, 2006](#)).

In addition, as described in [Marlow *et al.* \(2006\)](#), each tag serves as a link to resources tagged with the same keyword, so assigning popular tags makes resources more popular. This way resources can be easily retrieved using tags, and the resource sharing system can be better organized. Tags are specially important in resources otherwise hardly retrieved — when there is little text surrounding information, like photos and videos.

Recently these systems gained attention as elements of social interaction have been introduced, connecting individual activities (such as uploading a Web page to Delicious) to a rich network of shared tags, resources, and users ([Marlow *et al.*, 2006](#)). When these social interactions can be observed repeatedly in the same environment we can say that a social network (e.g. in [STSs](#) a network of users) is created. Social networks on the Web are increasing since the beginning when blogs and Wikis first appeared. Because of

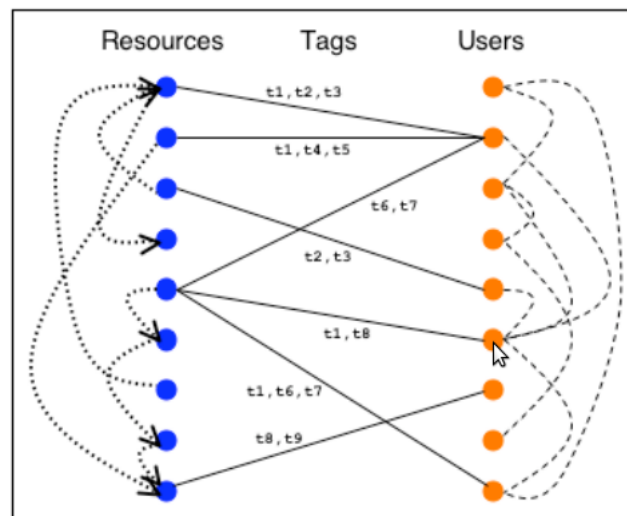
the network created, resource sharing systems that use tags are named **Social Tagging Systems**.

Because of their lack of predefined taxonomic structure, **STSs** rely on shared and emergent social structures and behaviors, as well as related conceptual and linguistic structures of the user community (Marlow *et al.*, 2006). This freedom is an advantage for users, making **STSs** popular for the ease of use.

These applications are quite popular: Flickr has a database containing more than 3 billion photos and receives 40,000 photo uploads per second¹ and 1,000,000 photos per day (Song *et al.*, 2008b). Another example of popularity is Delicious, that gets roughly 150,000 bookmarks posts per day (Song *et al.*, 2008b).

In Marlow *et al.* (2006) a model of **STSs** is presented, Figure 2.1: Tags are edge labels in a graph that connect users and resources. Users can share social relations (like friendship or “following uploaded bookmarks”) through the system (as in social networks). Resources also can have a relation (regarding hyperlink structure, for example).

Figure 2.1 Conceptual model of tagging systems (Marlow *et al.*, 2006).



In order to better describe how a **STS** works and its main features, we present an overview of Delicious, one of the most popular collaborative tagging systems. After this we present models of **STSs** that create a classification of them, according to certain attributes.

¹Information found in <http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr>, extracted on 2009.

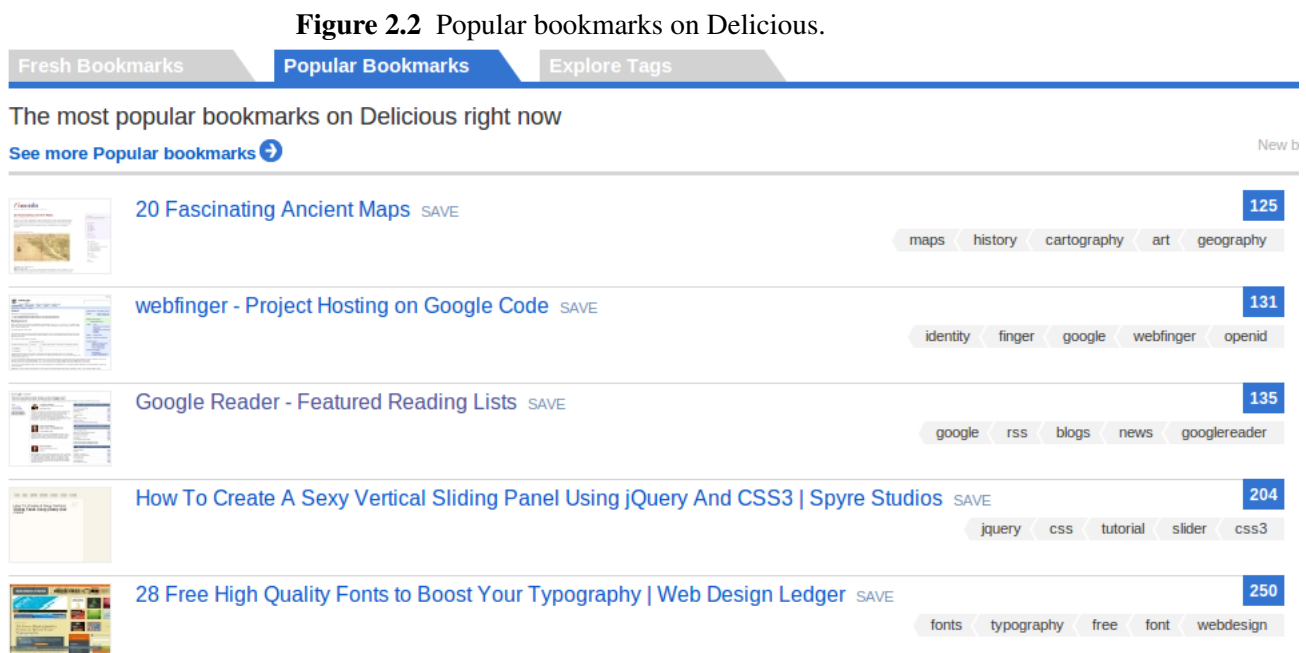
2.1.1 Delicious

Delicious is a social bookmarking site where users upload bookmarks (Web pages) they find interesting and apply tags to describe these pages thus creating a social network for sharing relevant content. Using the website users can, for example, explore bookmarks using tags (tag browsing), explore bookmarked friend's pages (network browsing), or follow bookmarks with a particular tag using RSS feeds (tag subscription).

The Web pages posted in Delicious' bookmarks are relevant and some huge datasets were tested in Heymann *et al.* (2008a) for tag assignment accuracy, i.e. if the tags users were using were relevant to classify content. They concluded that Delicious tags are indeed relevant and can even enhance Web search, as some tag information does not exist in text and hyperlink structure.

The main page of Delicious, Figure 1.2, highlights fresh and popular bookmarks and the “explore tags” view. Fresh bookmarks are pages which were added by users recently. This view shows pages with good upload frequency, therefore in the way to become popular bookmarks.

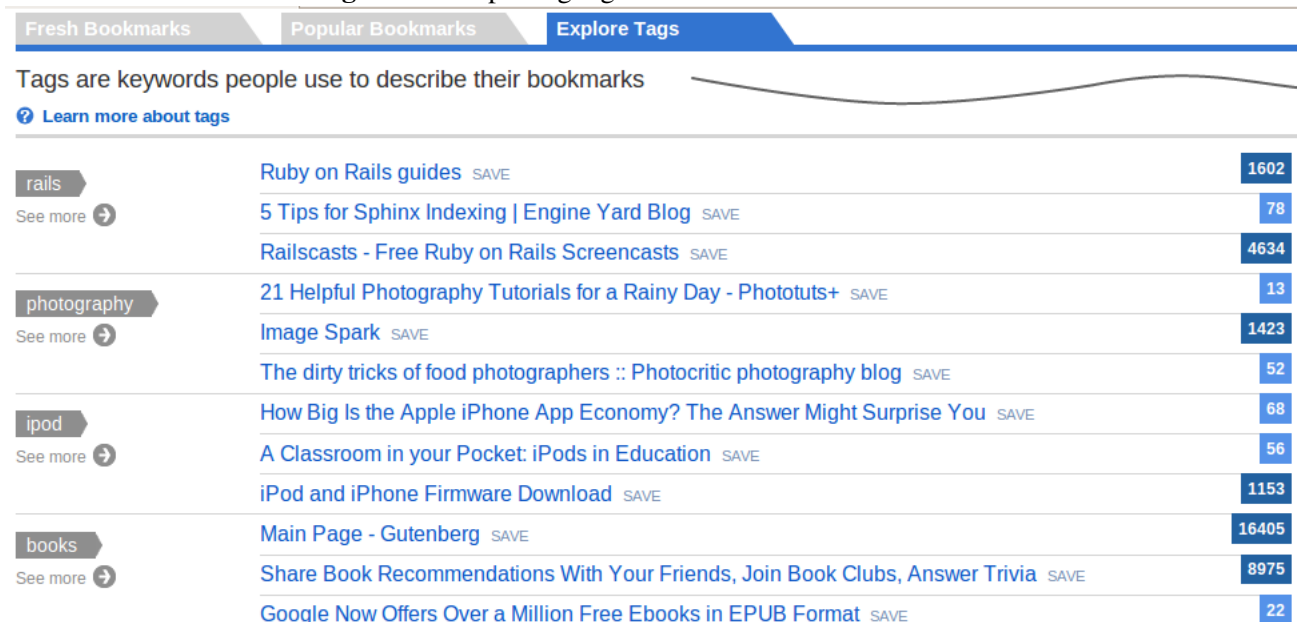
Popular bookmarks are pages with a more solid success story on Delicious, with lots of users sharing them, Figure 2.2. This feature can be an interesting source of information for trend detection and analysis.



Also accessible from the main page is the “explore tags” feature, Figure 2.3, that

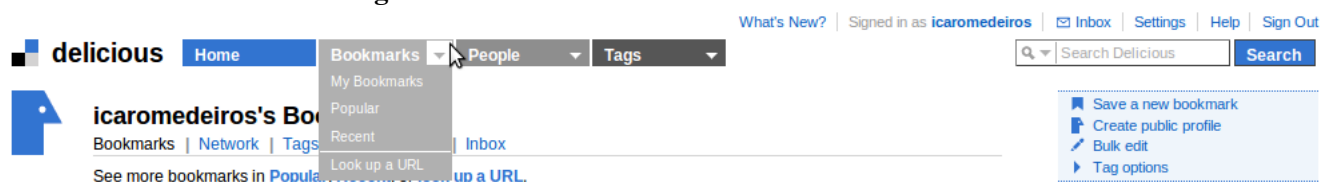
makes possible to browse Delicious through tags, enabling users to view recent bookmarks using specific tags, search for bookmarks using multiple tags, and explore related tags.

Figure 2.3 Exploring tags with Delicious.

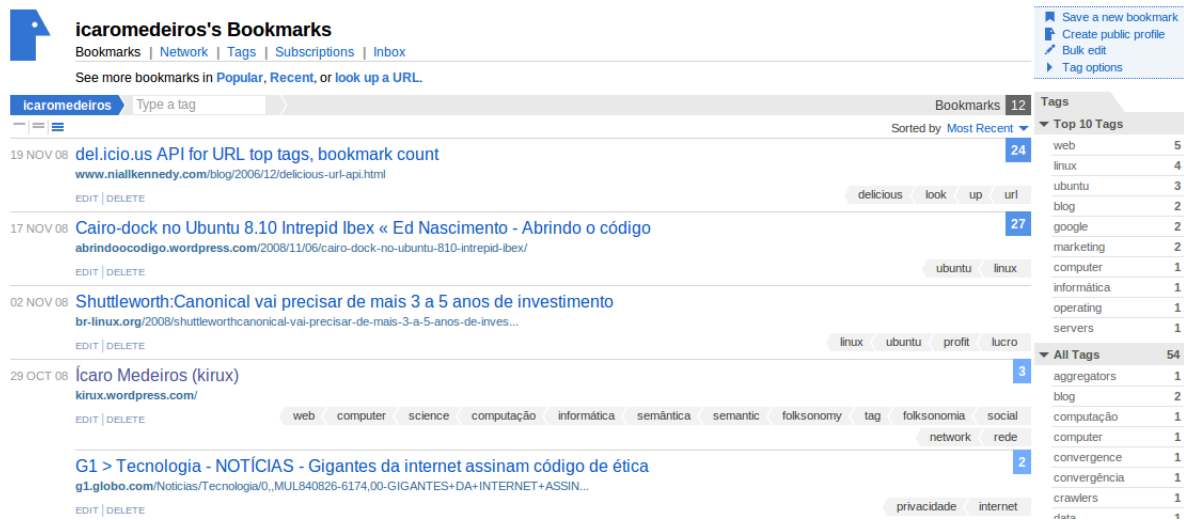


Through Delicious main menu, in the left top of Figure 2.4, some features, disposed hierarchically, can be accessed, such as:

Figure 2.4 Main menu of Delicious.



- Bookmarks
 - My bookmarks: This option shows the user's bookmarks, Figure 2.5. The bookmarks appear sorted by upload time and with information like the URL, number of people that uploaded it and the tags the user assigned to it. A summary of top and all tags assigned by the user is also shown. Another option is to search the personomy — the personal folksonomy, i.e. the collection of tags of a user;
 - Popular: This is a similar view as the one in Figure 2.2, showing Web pages frequently uploaded;

Figure 2.5 Bookmarks summary of a user in Delicious.

- Recent: Shows recent uploaded bookmarks, different from fresh bookmarks in Figure 1.2 because this page shows bookmarks being uploaded in real time and fresh bookmarks show new pages that tend to become popular;
- Look up a URL: In this view users can see a summary of tag usage for a specific URL.
- People
 - My Network: This view shows the bookmarks recently uploaded by the network of friends, their tags, frequency, etc. The network can also be managed by adding new friends, dividing the network into groups or declaring yourself a fan of some user;
 - Go to a user: In this view it is possible to search for a friend and see all his/her recent bookmarks.
- Tags
 - My tags: In this view a tag cloud is shown containing all the tags assigned by a user. Its display size is proportional to the frequency of use. Also, it is possible to do some organization tasks such as renaming tags, deleting tags, or managing tag bundles (to create manually assigned clusters).

Finally, the user menu in Delicious, on the right top of Figure 2.4, provides access to functions such as settings of your account, your inbox — with, for instance, messages containing bookmarks sent to you by friends, among others.

One of the main advantages of Delicious and STSs in general is that content sharing and tag assignment are easy jobs for most types of users. To show how easy is to upload a bookmark and assign tags in Delicious we show two ways of upload: through the main website and through a browser (Firefox) extension.

When logged in Delicious there is always an option on the right side of the page to “Save a new bookmark”, Figure 2.4, leading to Delicious save page², as shown in Figure 2.6.

Figure 2.6 Saving a bookmark in Delicious.

The figure consists of two screenshots of the Delicious 'Save a new bookmark' interface. The top screenshot shows the initial step where a URL is entered. The bottom screenshot shows the full form with fields for URL, Title, Notes, Tags, and a 'SEND' button, along with a 'Mark as Private' checkbox and a 'Save' button.

Save a new bookmark
Start by entering a URL

Did you know? Saving bookmarks to Delicious is much easier with our [bookmarking tools](#).

URL

Next

Save a new bookmark
Now add tags and notes

URL Required

TITLE Required

NOTES

TAGS 1000 characters left

SEND

☐ Mark as Private

Save **Cancel**

Tags **Send** **NEW!**

Sort: Alpha | Frequency

Recommended

google internet

Popular

chrome browser ie ie6 development webdev plugin

The official Firefox extension for Delicious (called Delicious Bookmarks³) provides a simple way to upload bookmarks, manage your tags and recently bookmarked websites using the browser interface as seen in Figure 2.7.

Considering specific Delicious advantages we can mention:

- It is a good environment for sharing, organizing, discovering and finding interesting bookmarks;
- Users can access their bookmarks from any computer at anytime;

²<http://delicious.com/save>

³<https://addons.mozilla.org/en-US/firefox/addon/3615>

Figure 2.7 Using a Firefox extension to use Delicious.

- It is a popular system with lots of users sharing and classifying content;
- As a popular [STS](#), there are many tools to help using Delicious features;
- It is an interesting source for trend detection or to follow which Web pages gained attention for being concerned with certain topics.

Delicious drawbacks include all the general problems related to [STSs](#) like polysemy, synonymy, homonymy, binded words, misplaced tags, among others, as stated in Chapter 1.

2.1.2 Classifying Social Tagging Systems

Some authors distinguish between different types of [STSs](#), concerning some attributes. We enlist the most important dimensions that classify [STSs](#) ([Marlow et al., 2006](#)):

- *Tagging rights*: Some systems only permit the content authors to tag resources (e.g. Flickr). Other systems do not have this restriction so users can freely classify content previously uploaded (e.g. Delicious);
- *Tagging support*: Some systems do not present other tags that users assigned to a resource (blind tagging), while other systems show tags previously assigned

(viewable tagging). Finally, some systems (Delicious, for example) suggest possible tags to the user (suggestive tagging);

- *Aggregation*: Systems may differ on how they count tags. For example, on Youtube and Flickr tags used more than one time for the same resource are counted as one (set-model), while Delicious can count multiple associations of a tag to one resource (bag-model);
- *Type of object*: This dimension regards the types of the resources being tagged. For example Delicious has a wider scope (Web pages) while other STSs are more specific: on Youtube users just tag videos.

STSs can also be evaluated using features concerning the incentives users have to tag. Marlow *et al.* (2006) claim that users may have multiple motivations to tag resources, the most important are:

- *Future retrieval*: The most important factor for users to classify content. One might like to retrieve the same (or similar) resources in the future;
- *Sharing*: This motivation has a social component because users may want to feel part of a community, and using tags can be a way of finding people with similar interests. *Attracting attention* is a motivation very similar to this one;
- *Opinion expression*: Some users may use subjective tags to resources to express an opinion about its content (e.g. “funny” or “bad”).

The research about STSs has been aiming four major questions: describing folksonomies formally, relationships between folksonomies and semantics, describing users’ tagging behavior, and tag suggestion. These issues will be covered in the next sections.

2.2 Folksonomies

The assignment of tags to resources creates a bottom-up classification called folksonomy (Hotho *et al.*, 2006a). The term was coined by Vander Wal⁴ as a blend of the terms “folk” and “taxonomy”. Some works in literature contest this term because folksonomies does not have any hierarchical classification as in taxonomies (Golder and Huberman, 2006). In this section we present some formal definitions and models of folksonomies.

⁴<http://vanderwal.net/folksonomy.html>

[Hotho et al. \(2006a\)](#) defines a folksonomy as a tuple $\mathbb{F} := (U, T, R, Y, \succ)$ where U , T , and R are finite sets that represent **users**, **tags**, and **resources**, respectively. Y defines a relation of tag assignment between them, i. e., $Y \subseteq U \times T \times R$. Finally, \succ is a user-specific hierarchical (is-a) relation between tags, i. e., $\succ \subseteq U \times T \times T$. Following this definition a personomy — folksonomy of a specific user — is also defined.

According to [Mika \(2007\)](#), a folksonomy can be defined using three disjoint sets $A = \{a_1, \dots, a_k\}$, $C = \{c_1, \dots, c_l\}$, $I = \{i_1, \dots, i_m\}$, corresponding the set of **actors** (users), the set of **concepts** (tags) and the set of **instances** (resources). The folksonomy is built when users assign **tags** to resources, thus creating a set of connections $T \subseteq A \times C \times I$. Thus the folksonomy can be seen as a tripartite graph with three-way hyper edges $H(T) = (V, E)$ where $V = A \cup C \cup I$ and $E = \{\{a, c, i\} | (a, c, i) \in T\}$. It could be noticed that these definitions are almost equal.

Some ontologies regarding folksonomies and tagging were proposed to provide a formal, machine processable and shared representation enabling reuse across **STSs** and a formal semantics to tags. [Gruber \(2007\)](#) defines a folksonomy where the Tagging class is the core concept, defined as

Tagging(object, tag, tagger, source, + or -)

where `tagger`, `tag`, and `object` are based on the actors, concepts and instances in [Mika \(2007\)](#). `Source` represents the **STS** in which this tag assignment is stored and the last parameter is an attempt to flag the assignment as good or bad, preventing spam and misclassifications in the ontology.

Later works extend this definition, refining and modeling it in a formal language. [Echarte et al. \(2007\)](#) creates an ontology capable of representing any folksonomy and also defines an algorithm to transform folksonomies into instances of the ontology and to update the resulting ontology over time. The ontology is written in OWL and has the advantage of being able to associate different variations of the same concept in which different tags were used (e.g. “Rails”, “ror”, and “Rubyonrails”).

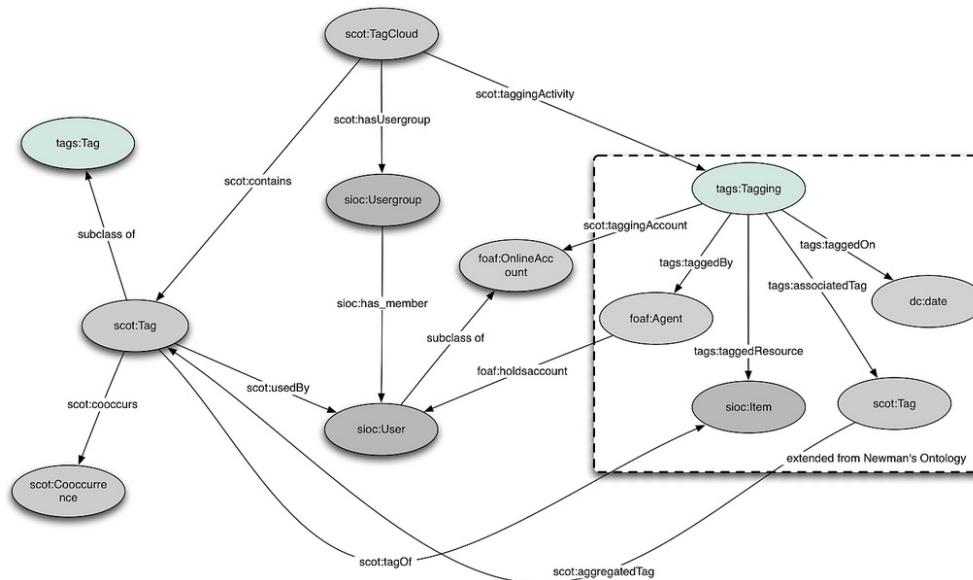
[Kim et al. \(2008\)](#) extend concepts stated in [Gruber \(2007\)](#) and [Newman et al. \(2005\)](#) and reuses vocabulary of existing and consolidated vocabularies like FOAF⁵ and SIOC⁶ to create the SCOT (Semantic Cloud of Tags) ontology. The first vocabulary concerns about people and interests and the later one is a popular ontology used to describe online

⁵Friend Of A Friend: <http://www.foaf-project.org/>

⁶Semantically-Interlinked Online Communities <http://sioc-project.org/>

communities. The ontology is written in OWL and briefly summarized in Figure 2.8. The work also contains a comparison with previous ontologies of folksonomies and tagging.

Figure 2.8 Graphic representation of the SCOT ontology.



2.2.1 Folksonomies and Semantics

One of the problems of folksonomies is the lack of formal and explicit semantics of tags. As they are a large dataset of lightweight annotations, they became interesting to researchers concerned about extracting machine-processable semantics from low-level structures (Cattuto *et al.*, 2008).

This can be achieved associating folksonomies with formal knowledge representation structures like ontologies with a folksonomy-ontology association (Jäschke *et al.*, 2008), a semantic enrichment of folksonomies (Angeletou *et al.*, 2008), or even with ontology learning using folksonomies as data source (Mika, 2007).

Most of these techniques use a bottom-up approach, trying to first extract tag relatedness and similarity. This relatedness carries semantic information, thus is the link with ontologies (Cattuto *et al.*, 2008). Measures of tag relatedness include clustering and graph techniques (Mika, 2007; Hotho *et al.*, 2006b), tag co-occurrence (Cattuto *et al.*, 2007), or a mix of statistical approaches (Jäschke *et al.*, 2008). Cattuto *et al.* (2008) summarizes these techniques of relatedness and compare them quantitatively with consolidated semantic similarity measures in structures like WordNet (Fellbaum, 1998).

Through mining association rules in folksonomies, Schmitz *et al.* (2006) discover

supertag relations and communities. Differently, [Tanasescu and Streibel \(2007\)](#) proposes a system where users can also assign tags to tags, thus creating explicit or implicit relations between tags that can contextualize ambiguous tags by creating semantic associations.

2.2.2 Tagging Behavior

Other different research area in folksonomy studies how users behave when tagging and using [STSs](#). User behavior is analyzed in several works which try to understand how users interact with the [STS](#), creating models of behavior, comparing them with real folksonomies, and discussing the implications of behavior patterns in the resulting folksonomy and the design of future [STSs](#).

For example, [Golder and Huberman \(2006\)](#) analyzes how the number of different tags used by specific users increases as they upload further bookmarks to Delicious and if there exists a typical number of uploads to a single resource after which the relative proportions of tags stabilize, i.e. the vocabulary for a resource has converged. The main reasons for the vocabulary stability, as claimed by authors, is that users imitate each other, the community is cohesive and users have the same vocabulary, or the stable content of pages limits the number of reasonable tags which could be used.

Other works which discuss imitation process include [Cattuto *et al.* \(2007\)](#) and [Suchanek *et al.* \(2008\)](#). This latter states that the high imitation rate found in Delicious can be explained by the mechanism of tag suggestion. [Heymann *et al.* \(2008b\)](#) analyzes the imitation rate in an attempt to define the predictability of tags.

[Marlow *et al.* \(2006\)](#) discusses tag usage in Flickr, stating that most users have few distinct tags in their vocabulary while few users have large sets of tags. This analysis indicates that socially related users tend to use a more similar vocabulary than disconnected users.

[Marlow *et al.* \(2006\)](#) comments that there are clearly distinct behaviors concerning tag usage through time: some users add new tags consistently as photos are uploaded, other users present a declined tag growth over time, and others have a small vocabulary that presents a sudden growth. This latter behavior might indicate, for example, a personal motivation (e.g, to organize photos as the collection grows) or a social one (e.g., to attract attention using tags).

Also concerning personal and social motivations, [Rader and Wash \(2006\)](#) states that there is very little inter-user agreement in Delicious, and that users, consciously or not, practice what they call “selfish tagging”.

In addition to imitation analysis, [Dellschaft and Staab \(2008\)](#) model folksonomies

also concerning background knowledge of users, i.e. if the users tend to assign keywords frequently found in a corpus of related pages of the resource being tagged, that this is a suitable approximation. Therefore, besides imitating previous tag assignment for the resource, users can choose a word from their active vocabulary that is related to the resource content.

Before getting into tag suggestion itself we present in the next sections research areas in which tag recommendation algorithms usually rely their systems on, namely Information Retrieval, Machine Learning, and a former research area very similar to tag suggestion: Keyword Extraction.

2.3 Information Retrieval

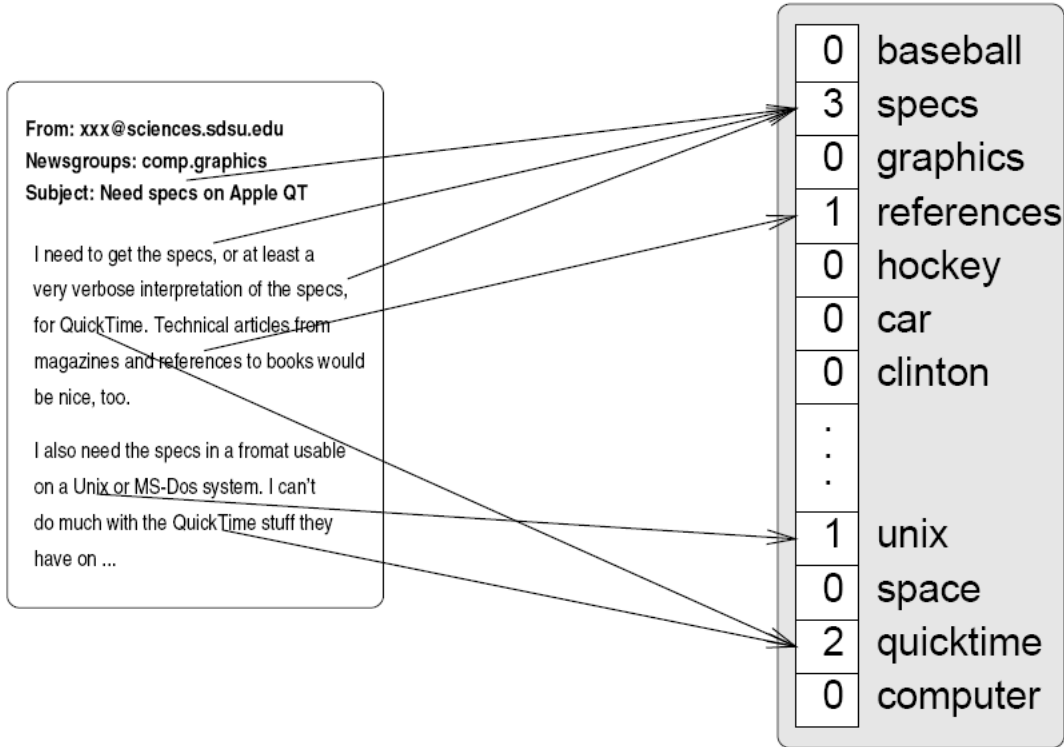
According to [Baeza-Yates and Ribeiro-Neto \(1999\)](#), Information Retrieval deals with the representation, storage, organization, and access of information systems. One main concern of **IR** is search: document search, information search within text documents, as well as search on the Web — the most known application.

In order to represent a text document many **IR** models were proposed. However, the Vector Space Model (**VSM**) has proven to be the most successful ([Baeza-Yates and Ribeiro-Neto, 1999](#)). In the **VSM** a text is seen as a vector where each dimension represents a term with an associated weight to indicate the relevance of this term for the document.

Most of the times, a preprocessing phase excludes non-informative words (called stop words), such as prepositions, articles, and connectives. These words are filtered, thus they are not indexed. Another filtering step is to reduce words to their root form (or stem), a process called stemming (e.g. the stemmed form of “studies” and “student” is “studi”).

Formally, a document d_j is represented as a term weight vector $d_j = \{w_{1j}, \dots, w_{|\tau|j}\}$ where τ is the dictionary, the set of terms that occur in the document collection. An example is given in Figure 2.9, where a simple frequency occurrence weight is used to model a text.

The most used weighting scheme is the Term Frequency-Inverse Document Frequency (**TF-IDF**), based on the combination of two weights: the Term Frequency (**TF**) and the Inverse Document Frequency (**IDF**) ([Baeza-Yates and Ribeiro-Neto, 1999](#)). Term Frequency TF_{ij} is the (normalized) number of times a term t_i appears in the document d_j . The idea is that a term which is recurrent in the text is likely to be important. **TF** weight is given by Equation (2.1):

Figure 2.9 Representing a text document using the **VSM**.

$$TF_{ij} = \frac{n_{ij}}{\sum_k n_{kj}} \quad (2.1)$$

In Equation (2.1), n_{ij} is the frequency of the term i in document j and $\sum_k n_{kj}$ is the sum of term frequencies in document d_j .

The Inverse Document Frequency is the inverse of the number of documents in which the term occurs. It is used to distinguish common words from rare words in the corpus. For example, in a corpus about Artificial Intelligence (AI) terms like “intelligence” or “intelligent” are expected to appear in a lot of documents and are not good to differentiate documents. The IDF_i of a term t_i is given by Equation (2.2):

$$IDF_i = \log \frac{|D|}{|\{d : t_i \in D\}|} \quad (2.2)$$

In Equation (2.2), $|D|$ is the number of documents in the collection and $|\{d : t_i \in D\}|$ is the number of documents where the term t_i appears. Finally, the Term Frequency-Inverse Document Frequency (**TF-IDF**) is simply the product of **TF** and **IDF**. In this work we use a normalized **TF-IDF** so that all values fall in the $[0, 1]$ interval. The main reason is that

TF-IDF tends to overestimate terms in large documents (Salton and Buckley, 1988). The **TF-IDF** formula with the cosine normalization is given by the Equation (2.3):

$$\text{TF-IDF}_{ij} = \frac{TF_{ij} \times IDF_i}{\sqrt{\sum_k (TF_{kj} \times IDF_k)^2}} \quad (2.3)$$

In (2.3) the denominator $\sqrt{\sum_k (TF_{kj} \times IDF_k)^2}$ is the Euclidean norm of document vector d_j . A summary of different types of term weighting schemes is done in Salton and Buckley (1988).

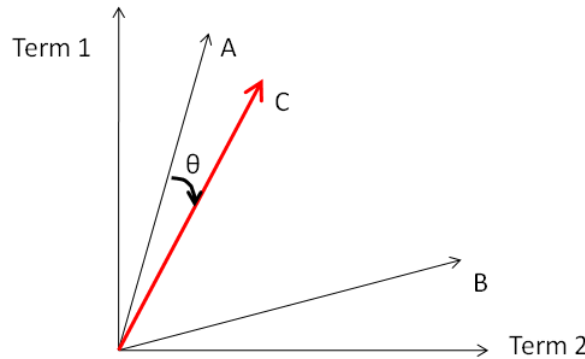
We are going to use extensively the **VSM** and the weights presented in this section to represent Web pages in the process of modeling them to build our classifier.

For example, some tag suggestion works suggest tags previously assigned to similar documents in the folksonomy. To measure document similarity the documents are represented as term vectors, using weights as **TF-IDF**, and the cosine similarity between documents A and B is calculated. Similar documents are those that have a small angle between them, as seen in Equation (2.4) (Baeza-Yates and Ribeiro-Neto, 1999):

$$\cos\theta = \frac{A \cdot B}{||A|| ||B||} \quad (2.4)$$

This is depicted in Figure 2.10. Consider three documents A , B , and C that contain only two terms and are represented as vector of term weights. Document C is more similar to A because it is closer — i.e. the angle separating the documents is smaller — to A than B in the vector space.

Figure 2.10 Measuring cosine similarity between documents.



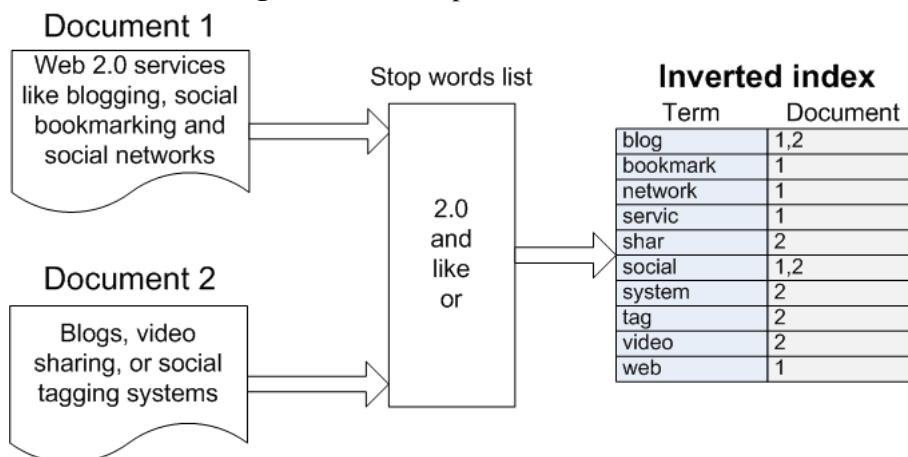
To store information about term weights and to make future retrieval fast, **IR** systems generally represent text in a data structure called index. Optimized methods for indexing

are responsible for the success of Web search engines (Brin and Page, 1998), that can handle billions of Web pages and return answers to search queries in milliseconds.

A good example of text indexing technique is the inverted index. An inverted index is a word-oriented mechanism composed of two elements: the vocabulary and the occurrences. The vocabulary is the set of words in the text — or in the corpus. For each such word a list of all the documents where the word appears is stored (Baeza-Yates and Ribeiro-Neto, 1999). Given that this is the inverse of the common relationship, in which documents list terms, this structure is called inverted index.

An example of inverted index is depicted in Figure 2.11. It contains information about terms and their occurrence in documents, where the stop-words are filtered and the index terms are stemmed. It is clear that this makes text search easy. For example, a search for “social bookmarking” will match the documents 1 and 2, where at least one of these words appears. But document 1 has the two words, therefore, the document must be more relevant and this information can be used to rank the search result.

Figure 2.11 Example of inverted index.



2.4 Machine Learning

The research area of Machine Learning (ML), a sub-field of AI, is concerned with the question of how to build computer programs that automatically improve (or “learn”) with experience (Mitchell, 1997). These programs eventually eliminate the need for detailed programming (Samuel, 1959).

Formally, a program with ML can be defined as (Mitchell, 1997):

A computer program is said to learn from experience E with respect to some

class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

In this work we use supervised learning, a major type of **ML** method, which consists in, given a set of training data, consisting in pairs of input objects and output objects, deduce the function that relates these input/output objects. In our case, given a collection of Web pages, a set of negative examples (terms that are not tags) and a set of positive examples (tags), a binary classifier is created that, given new unforeseen pages, deduces whether a term is suitable to be a tag or not. The classifier is then used to suggest tags.

Some concepts and definitions of **ML** which are going to be used (mostly in Section 3.4 and Section 3.3) are listed below:

- **Feature** (or attribute): Describes a characteristic or an aspect of an example. In tag suggestion, it could be a numeric description of the weight of a term in a Web page using the simple term frequency of Figure 2.9;
- **Example**: It is a tuple of feature values that describes an object. For example, in a tag suggestion system with only the **TF** and **IDF** of a term as features, we could have an example $e_1 = \{0.3118, 0.0189, true\}$ representing the values of **TF**, **IDF**, and class label (it is a tag), respectively;
- **Corpus**: The set of examples containing feature values and class labels;
- **Feature space**: Given a number n of numeric features and a set of examples, a feature space is a vector space in \mathbb{R}^n used to represent example vectors (f_1, \dots, f_n) of feature values;
- **Classifier**: The classifier is created during the process of training to recognize patterns and build a function, in our case, to see whether a term is good to be used as a tag or not;
- **Class**: Special feature used in supervised learning to label examples. As our suggester is a binary classifier, the class attribute is a boolean value that represents if the term is a tag or not. The task of a classifier is to infer this label.

Based on the definition of text classification in [Sebastiani \(2005\)](#) we can formally define tag suggestion as the task of approximating the unknown target function $\Phi: \vec{t}_i \rightarrow \{T, F\}$ (which describes how a term t_i with feature values (f_1, \dots, f_m) should be classified as a tag or not, given a supposed expert authority) by means of the function $\hat{\Phi}: \vec{t}_i \rightarrow \{T, F\}$,

called a classifier. If $\Phi(t_i) = T$, so t_i is a positive example, while $\Phi(t_i) = F$ is a negative example.

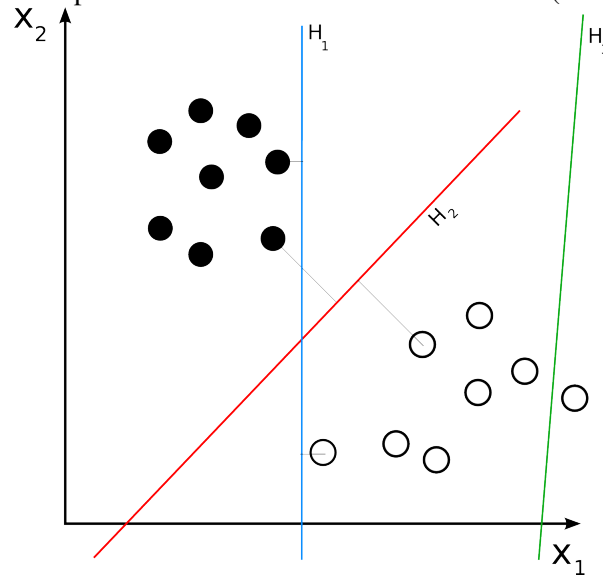
A lot of methods to create this $\hat{\Phi}$ function have been proposed such as neural networks, decision trees, and K-NN (Mitchell, 1997). The technique used in this work is the Support Vector Machine (SVM) (Joachims, 1998).

2.4.1 Support Vector Machine

Support Vector Machine (SVM) is a method (or a set of methods) of supervised learning, used for tasks like classification. The basic form of SVM consists in the creation of a hyperplane (or a set of hyperplanes) that divides the feature space into partitions so that each partition only contains examples of a specific class. In a binary classifier, for example, this hyperplane separates positive and negative examples with a maximum-margin (Joachims, 1998).

As seen in Figure 2.12, there are examples (dots) with two features (thus plotted in \mathbb{R}^2) representing two classes (white and black) and three hyperplanes (in \mathbb{R}^2 hyperplanes are lines) H_1 , H_2 , and H_3 . The hyperplane H_3 does not separate the two classes, while H_1 is very close to two examples in different classes, therefore it has a small margin. The hyperplane H_2 is the one that provides the maximum-margin.

Figure 2.12 Example of how to construct a SVM classifier (Wikipedia, 2010).



The SVM model can be formalized as follows (Burges, 1998). Consider a training corpus $C = \{\vec{x}_i, y_i\}$ where $i = 1, \dots, l$ (number of examples), $\vec{x}_i \in \mathbb{R}_n$ is a feature vector

and $y_i \in \{1, -1\}$ is the class label. Suppose a hyperplane H that separates positive and negative examples satisfying the equation $\vec{w} \cdot \vec{x} + b = 0$ where the vector \vec{w} is normal to the hyperplane, $\frac{|b|}{\|\vec{w}\|}$ is the perpendicular distance from the hyperplane to the origin, and $\|\vec{w}\|$ is the Euclidean norm of \vec{w} .

Let d_+ and d_- be the shortest distances from H to the closest positive (e_+) and negative example (e_-) respectively. The margin is $m = d_+ + d_-$. We want to choose \vec{w} and b in order to maximize the margin m , the distance between the parallel hyperplanes H_+ and H_- containing e_+ and e_- , both parallel to H . As the examples must not fall into the margin we add some restraints:

$$\vec{w} \cdot \vec{x} + b \geq +1 \text{ for } y_i = +1 \quad (2.5)$$

$$\vec{w} \cdot \vec{x} + b \leq -1 \text{ for } y_i = -1. \quad (2.6)$$

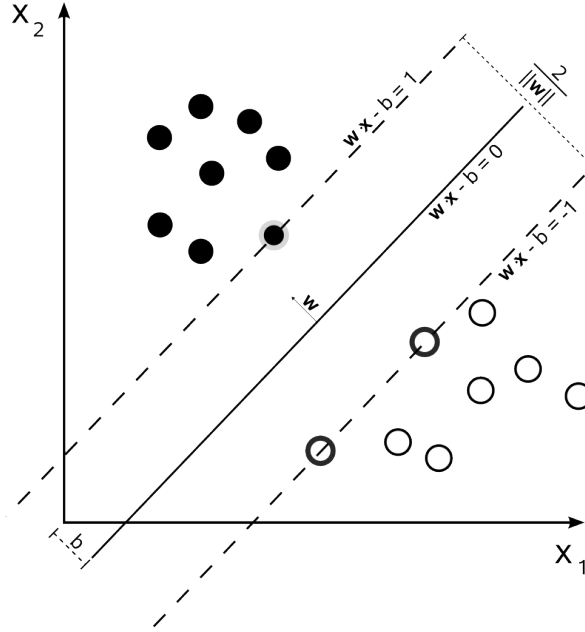
Combining the two equations we have the restriction:

$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0, \forall i \quad (2.7)$$

In the \mathbb{R}_2 scenario of Figure 2.13 where the examples can be separated by a line (thus the training data is linearly separable) we find that $d_+ = d_- = \frac{1}{\|\vec{w}\|}$, thus the margin is $m = \frac{2}{\|\vec{w}\|}$ and the problem is reduced to minimize $\|\vec{w}\|$ subject to the restriction in (2.7). The training points like e_+ and e_- present in one of the hyperplanes H_+ or H_- and for which the Equation (2.7) holds are called support vectors, indicated in Figure 2.13 by extra circles.

SVM is extensively used in tasks involving text, such as text classification (Joachims, 1998) where it was found to be the most successful method (Sebastiani, 2005) and tag suggestion (Heymann *et al.*, 2008b; Wang and Davison, 2008). **SVM** can also deal with high-dimensionality and sparseness of data, very common in tasks involving text. This is the main reason for our choice of **SVM** instead of other methods.

SVM algorithms use a compact representation of the feature space, by the use of kernel functions. Popular kernel types include linear, polynomial, sigmoid and Radial Basis Function (**RBF**). The type of kernel is an important parameter in SVM classifiers.

Figure 2.13 Formal example of how to construct a SVM classifier (Wikipedia, 2010).

2.4.2 Classifier Evaluation

To evaluate the effectiveness of classifiers, at first the corpus is divided into two sets of examples: the training and test set. The first one is used to build the classifier and the second to test it. As the class labels of examples are known they are compared with the labels given by the classifier output.

This division of training and test set is done in several ways to guarantee that the classifier is not dependent of the training data used and that the test set is chosen randomly. The most used method is the k-fold cross-validation (Mitchell, 1997), which works as follows:

1. The corpus C is divided into $C = \{C_1, \dots, C_k\}$ non-overlapping subsets (usually $k = 10$);
2. K iterations $i = 1, \dots, k$ are made in which:
 - (a) C_i is the test set;
 - (b) The training set $T = \{C_j \cup \dots \cup C_k \mid C_j \neq C_i\}$ is the union of the other subsets where $j = 1, \dots, k$;
3. The global error of the classifier is the average of the error result for each subset C_i .

Other method of validation is called Holdout and consists in creating a subset of fixed size with examples chosen randomly from the corpus to create a test set, and the remaining examples are the training data. Usually the percentage used is 50%-50%.

The error is estimated comparing classifiers outputs and corpus data. This can be visualized using a confusion matrix, where the rows represent the class labels predicted and the columns represent the actual value of class labels. For example, for the class *tag* the confusion matrix, Figure 2.14, contains:

- True Negatives (TN): examples classified as *not tags* correctly;
- False Positives (FP): examples classified as *tags* which are in fact *not tags*;
- False Negative (FN): examples classified as *not tags* which are in fact *tags*;
- True Positive (TP): examples classified as *tags* correctly;

Figure 2.14 Example of a confusion matrix, used to test classifiers ([Wikipedia, 2009](#)).

		actual value		
		<i>p</i>	<i>n</i>	total
prediction outcome	<i>p'</i>	True Positive	False Positive	<i>P'</i>
	<i>n'</i>	False Negative	True Negative	<i>N'</i>
total		<i>P</i>	<i>N</i>	

With these confusion matrix values, measures like precision, recall, and F-1 can be calculated. These measures are standardized and used in both [IR](#) and [ML](#) ([Baeza-Yates and Ribeiro-Neto, 1999](#)). Precision *P* represents the percentage of examples which were correctly labeled as positive. The measure is useful to test the confidence of the classifier when labeling an example as positive and it is given by Equation (2.8):

$$P = \frac{TP}{TP + FP} \quad (2.8)$$

Recall measures from the whole set of positive examples how many examples the classifier correctly labeled as positives, i.e. recall measures at what extent the classifier “misses” positive examples in the classification. It is measured by the division of true positives by the sum of true positives and false negatives (i.e. the denominator contains all positive examples, correctly or incorrectly classified) and is seen in Equation (2.9):

$$R = \frac{TP}{TP + FN} \quad (2.9)$$

The F-1 measure combines the precision and recall measures presented in (2.8) and (2.9), calculating the harmonic mean between those measures. It is given by the Equation (2.10):

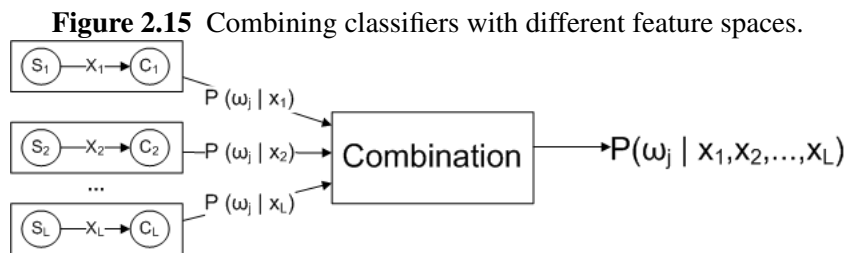
$$F1 = \frac{2 \times P \times R}{P + R} \quad (2.10)$$

2.4.3 Combining Classifiers

Combining classifiers is useful when predictions made by one method (like SVM) can be enhanced combining its output with other classifier with a different technique (like KNN and others) over the same feature space. This classifier may have a different perspective on the same problem.

Another type of combination, the one used in this work, is when different classifiers explore different features spaces (e.g. a security system that uses one classifier for each sensor — retina, face recognition, and fingerprint) and the final decision is a combination of the classifier results. In our case there is a classifier for each source of knowledge.

This process is depicted in Figure 2.15: for each sensor s_i , an example x_i is submitted to a classifier C_i that gives the posterior probability of the example being classified under class ω_j , where $i = 1, 2, \dots, L$. Considering $j = 1, 2, \dots, C$, the result of the combined classifier is the class ω_j which has the highest combined posterior probability $P(\omega_j | x_1, x_2, \dots, x_L)$.



The problem of combining classifier outputs when acting in different feature spaces is well studied. Kittler *et al.* (1998) creates a framework for comparing different types of combination rules — ways of calculating $P(\omega_j|x_1, x_2, \dots, x_L)$, with the major advantages and drawbacks for each rule. In ANTaReS we tested two combination rules: majority vote and the product rule.

The majority vote rule relies only on the class labels returned by each classifier. The final chosen class is the one that obtained the majority of votes from all classifiers. Usually an order of precedence for the classifiers is provided to solve the case of ties. In our case the order is defined by the percentage of tags found in that information source (as detailed in the analysis of Section 4.2).

The product rule works as follows:

1. Suppose there are C classes $\{1, \dots, C\}$, L classifiers with example vectors $\{x_1, \dots, x_L\}$, where each vector represent one of the L feature spaces;
2. Assign the example x_i to class ω_j if

$$[P(\omega_j)]^{-(L-1)} \left(\prod_{i=1}^L P(\omega_j|x_i) \right) > [P(\omega_k)]^{-(L-1)} \left(\prod_{i=1}^L P(\omega_j|x_i) \right), \quad (2.11)$$

where $k = 1, \dots, C$ and $k \neq j$

In Equation (2.11) $P(\omega_j)$ is the prior probability of an example being classified in ω_j class and is estimated using the number of training examples that are labeled in this class. $P(\omega_j|x_i)$ is a value given by each ω_j classifier: it is the (posterior) probability that given an example x_i the classifier will label it using class ω_j .

In our case the formula can be simplified as there are only two classes: tag or non-tag. Thus, the example is a tag if:

$$\frac{\prod_{i=1}^L P(\omega_{TAG}|x_i)}{P(\omega_{TAG})} > \frac{\prod_{i=1}^L P(\omega_{NON-TAG}|x_i)}{P(\omega_{NON-TAG})}, \quad (2.12)$$

and non-tag otherwise.

2.5 Keyword Extraction

Before talking about tag suggestion we take a step back and talk about a research area which has a lot in common with tag suggestion: Keyword Extraction. Automatic keyword

extraction is the task of choosing representative terms from a text (Giarlo, 2005) and it was studied before the advent of STSs.

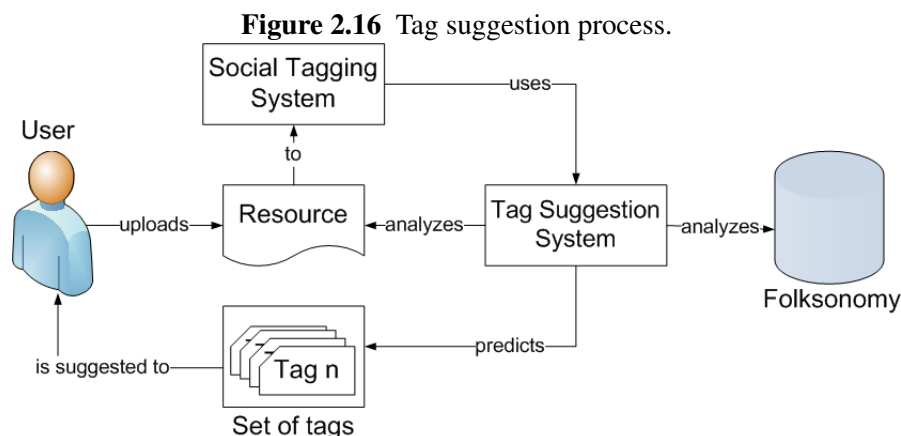
Examples of proposals in this area are Matsuo and Ishizuka (2003) and Tonella *et al.* (2003), where features such as term frequency or term co-occurrence are combined in a formula and used to rank the terms in order of importance as relevant keywords. In the case of Tonella *et al.* (2003), the extracted keywords are later used to cluster Web pages.

Keyword extraction is also often used in online advertisement, often using ML techniques. Wu and Bolivar (2008), Yih *et al.* (2006) and Lacerda *et al.* (2006) have used logistic regression and genetic algorithms, respectively, to extract keywords from a Web page. These keywords are afterwards compared to paid advertisements, to determine if these should be placed on the given page. Given that this classification should be performed very fast the ML techniques used are rather shallow and only use the text as information source.

2.6 Tag Suggestion

Tag suggestion is a mechanism present in most STSs which consists in recommending tags for resources being uploaded by a user. It is an important feature as sometimes users are not aware of relevant keywords or are not willing to engage in such task. Here we review some of the approaches proposed to date.

In the tag suggestion process, the STS uses data of the folksonomy (previous assignments) or of the resource itself (like text statistics) to predict a set of tags and suggest them to users, as depicted in Figure 2.16.



In Delicious, for example, if a resource (URL) which is being submitted already exists in the dataset, the system sorts the most popular tags for this item and recommends them

to the user, as seen in the Delicious save page, Figure 2.6. Although useful, this approach does not work when the resource is being published for the first time and in systems where every resource is unique (like Youtube or Flickr).

It is said that tag suggestion speeds up vocabulary convergence (Heymann *et al.*, 2008a; Golder and Huberman, 2006), one of the main problems in folksonomies. Other advantages of tag suggestion in STSs include (Heymann *et al.*, 2008a):

- Increase recall of tag queries/feeds: users can subscribe to tags they are interested in, using RSS feeds. Tag suggestion increases the number of tags available in those feeds, so more resources (possibly interesting) will now be found in this subscription. Similarly the recall of tag queries is increased;
- Inter-user agreement: when users have similar interests but different vocabularies, resource classification and organization might be misleading. Suggestion helps users to achieve a vocabulary consensus;
- Tag Disambiguation: ambiguous and polysemous tags might mislead users and are easily found in STSs like Delicious, as seen in Chapter 1. Adding additional non-ambiguous tags might help to contextualize tags and help disambiguate them, specially in STSs in which the aggregation of tags follows a bag-model (Section 2.1.2);
- Bootstrapping (cold start): the way users assign tags is determined by previous experience with tags. With a low tag usage, for example at the beginning of a STS, users won't assign tags, so a suggestion system could previously and automatically assign tags to resources.

2.6.1 Tag Suggestion Approaches

In this subsection we present the most important works for tag suggestion. These mostly differ from our work since there are no ML techniques applied. Works using ML are the subject of the next subsection.

Chirita *et al.* (2007) uses a formula to measure term weights combining features like term frequency, TF-IDF and HTML features such as presence in anchors, meta tags, among others. Also, personalized tags are suggested comparing Web pages with desktop documents, using the cosine similarity — Equation (2.4). Terms with the highest weights are then extracted from similar documents for recommendation.

In [Mishne \(2006\)](#) and [Suchanek *et al.* \(2008\)](#), a collaborative filtering approach is used, where tags assigned to a given set of pages are also suggested to new, similar pages. Collaborative filtering is extensively used in recommender systems ([Herlocker *et al.*, 2004](#)), for example, to suggest products to users based on similar user interests. Tag suggestion can be seen as a recommendation task, given that tags are recommended (suggested) to users.

In these collaborative filtering approaches documents are represented using the **VSM** ([Baeza-Yates and Ribeiro-Neto, 1999](#)) discussed in Section 2.3. Therefore documents are seen as vectors of term weights and the similarity is measured using Equation (2.4).

Some systems, like [Symeonidis *et al.* \(2008\)](#) and [Jäschke *et al.* \(2007\)](#), use the graph definition in [Mika \(2007\)](#) and seen in Section 2.2 for tag suggestion systems. For example, if users A and B share the same interests, i.e. they tagged similar resources, they are considered similar users. Therefore, tags are suggested to user A if user B assigned them to the same (or similar) resource. The major drawback of the approaches discussed so far, however, is the fact that they can only recommend tags already in the folksonomy.

With the same goal as our approach, [Oliveira *et al.* \(2008\)](#) describe a tag suggestion solution for the Digg website⁷. The authors rely on a heuristic combination of statistical features taken from the words in the resources to be tagged to determine if each term could be used as a tag.

They can suggest new unforeseen tags, however, their best results required the use of a set of features that is very specific to the dataset, namely the description and thematic categories of the documents being tagged. These features are not usually present in other **STSs** and, therefore, their approach cannot be trivially ported to other environments.

2.6.2 Machine Learning Approaches

Unlike these approaches, however, **ML** techniques can suggest new unforeseen tags and have been used in many tag suggestion works. These works, which are more similar to our approach, will be examined in this section.

The recommendation system in [Sood *et al.* \(2007\)](#) uses case based reasoning to find keywords previously associated with blog posts to recommend as tags for new blog posts. The system considers several features such as the frequency of occurrence of the tag in previous posts, number of times a term appears in the document, blog popularity, among others. However the limitation of tags already in the folksonomy is still present.

⁷<http://digg.com/>

A system that combines content-based and graph-based suggestion was presented in [Lipczak et al. \(2009\)](#). Using a training corpus, weights were assigned to each source recommender and combined using a technique called “leading precision rescorer”.

[Heymann et al. \(2008b\)](#) used a SVM classifier to predict tags using features such as page text, anchor text, surrounding hosts, and other tags applied to the URL. Association rules between tags were also used and were said to have a crucial influence on precision of suggestions, being able to give insights about relationships between tags.

With the original purpose of expanding queries, a tag suggestion system based on page contents is used in [Wang and Davison \(2008\)](#). They train a set of binary SVM classifiers, each one focused on a single tag. Thus, tags are seen as categories and resources are then classified as belonging or not to a tag/category with a Delicious dataset of 503 webpages.

In [Song et al. \(2008a\)](#) the authors are concerned about efficiency and create a system for real-time tag suggestion where the time required for evaluating a document is 1 second in average. They have used a combination of clustering, folksonomy graph, and mixture model techniques to build a classifier.

Another work of Song et. al ([Song et al., 2008b](#)) presents a framework for tag suggestion using a multi-class sparse Gaussian process classification. In this case, tag suggestion is seen as a problem of multi-label classification, where each resource can be classified under several different categories, each category corresponding to a tag.

Authors claim that the classifier is built with very few training instances compared to other methods, making training time linear and classification time constant. In this case of using few instances the results are better than classifiers using SVM. The authors also claim that the system is capable of recommending relevant tags to images and videos, by using surrounding textual information.

3

ANTaReS

A Tag Suggestion System

ANTaReS is a tag suggestion system for Web pages. Given a resource, e.g. a Web page, a set of keywords is presented for the users and they can assign them to the page. The set of keywords is given by a classifier that analyzes term features to suggest tags.

As stated in Section 1.1, ANTaReS has some requirements, such as:

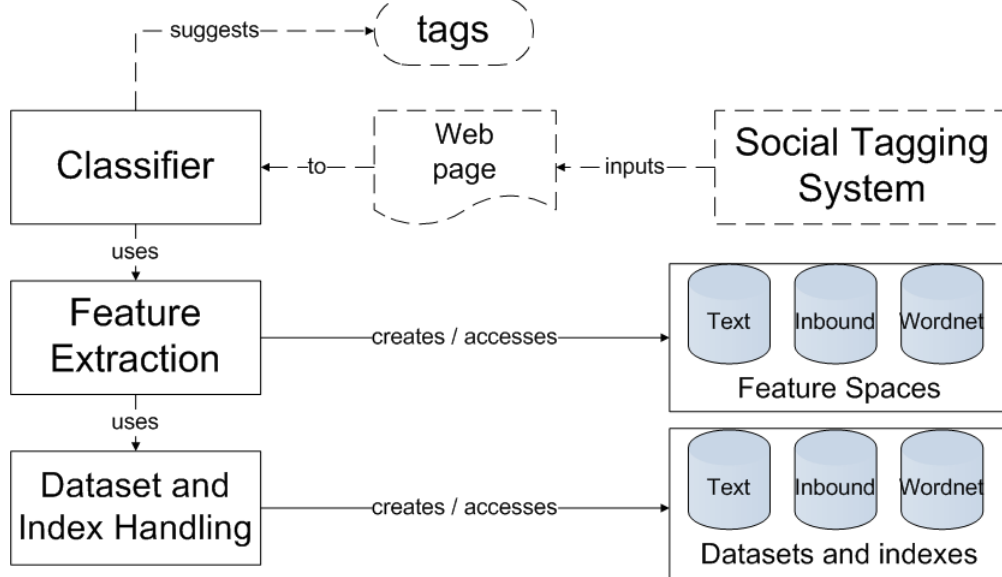
- To recommend tags to pages without any information of an existing folksonomy. ANTaReS must also be able to suggest new unforeseen tags;
- To be designed in a way that facilitates ANTaReS to be plugged to any Social Tagging System (STS) that uses Web pages as resource. For this, STS-specific features cannot be part of the classifier;
- To validate the methods implemented in the system using a real tagging data as baseline and to perform an user-based evaluation.

In this chapter we describe how these requirements were satisfied by describing ANTaReS, explaining its architecture, the datasets used, how the classifier was trained, how the features were extracted, how the classifiers are combined, how the suggestion is performed, and some details about code packages and implementation.

3.1 System Summary and Architecture

The main objective of ANTaReS was to build a classifier that, given a Web page, suggests a set of tags. ANTaReS is composed by three main subsystems and some important databases, depicted in Figure 3.1:

Figure 3.1 System architecture of ANTaReS.



- A subsystem for **dataset handling**, that creates and accesses the corpus with Web pages and associated tags;
- The **datasets** hold information about candidate terms, extracted from several sources and described in Section 3.2;
- Using the Information Retrieval ([IR](#)) method of indexing, the Web pages and the inbound pages are represented as inverted **indexes**, described in Section 2.3. These structures help to measure features values for the classifiers;
- A subsystem for **feature extraction**, responsible for accessing the dataset handling package to create and access the **feature spaces**, databases that relate candidate terms and their feature values, used to train the classifier;
- The **classifier** subsystem: the main package in the system, it is the component to be added to a [STS](#) so that, given a Web page, a set of tags is suggested. It provides the mechanisms to classify terms as tags.

The elements of Figure 3.1 in dotted lines are outside of ANTaReS scope, in a [STS](#) that calls the classifier of ANTaReS for tag suggestion. With the [STS](#) part in ANTaReS architecture, we emphasize that the system must be easily plugged to a [STS](#).

3.2 Datasets

The classifier in ANTaReS works by examining terms extracted from three sources of information:

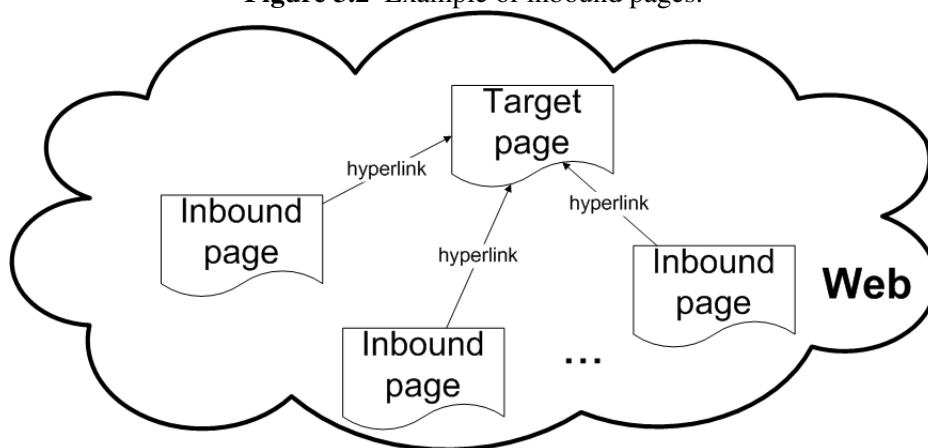
1. The target Web page to which tags will be suggested;
2. Inbound pages, i.e. Web pages that have a link to the target page;
3. Terms related with words in the target page, extracted using Wordnet.

For each of these sources, we construct a dataset with a list of candidate terms and whether they are tags or not. This information is essential to construct the classifier and to evaluate our suggestions. Furthermore, we analyze these sources to get insights for their usefulness for tag suggestions in Section 4.2.

The Web pages dataset contains terms from the target Web page itself and is fully described in Section 4.1.1. These HTML pages are also saved in an index — see Section 2.3.

The inbound pages dataset contains terms found in inbound pages. An inbound page is a page that has one or more hyperlinks to a target Web page in the Web pages dataset, as depicted in Figure 3.2. These hyperlinks are also called inbound links or inlinks. As for the Web pages dataset, the inbound pages are also represented in an index. The process of extraction is explained in Section 4.1.2.

Figure 3.2 Example of inbound pages.



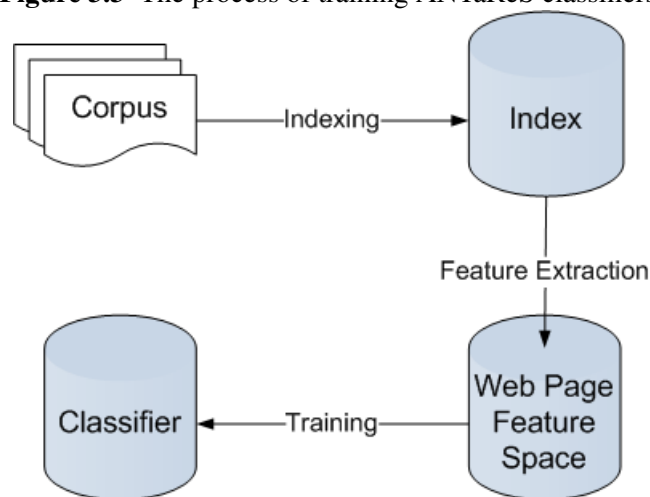
Finally, the Wordnet dataset contains words that hold semantic relations — like synonymy — with relevant terms in the target Web page. More detail about this dataset is presented in Section 4.1.3.

3.3 Training Phase

As seen in Section 2.4, in supervised learning we create a classifier by presenting several examples with feature vectors and class values. This classifier learns the patterns that differentiate examples of each class, in our case, terms suitable to be tags or not.

This process is the training phase, as summarized in Figure 3.3: the Web pages dataset is indexed, the term features are extracted — which will be described in Section 3.4, and the training data is given to the classifier. At the end, a classifier that works with Web page terms and features is created. The process is similar for the other sources.

Figure 3.3 The process of training ANTaReS classifiers.



The Machine Learning (ML) classification technique used to create the classifiers in training was the Support Vector Machine (SVM), explained in Section 2.4.1. In the experiments of Section 4.3 we show a comparison between SVM and other ML methods to confirm it was the better choice. SVM is being used extensively in related tasks like text classification (Joachims, 1998; Sebastiani, 2005) and tag suggestion itself (Heymann *et al.*, 2008b; Wang and Davison, 2008).

The proportion of positive and negative examples is variable for each classifier. For Web page terms the words chosen as tags by the Delicious users were used as positive examples and an equal number of randomly chosen words in the page were used as negative examples.

As all the inbound features — Section 3.4.6 — are extracted using TF-IDF values, we use a fixed X and extract the X terms with highest and lowest TF-IDF values. For the features explained in Section 3.4.6, inbound link anchors, inbound context, and inbound

content, the fixed X is 20, 50 and 100¹, respectively.

Finally, for the Wordnet terms and features, we use all the terms returned from the queries, with no distinction of positive and negative example or relying on values to extract a subset of terms.

An important part of training is the extraction of relevant and information rich features that measure the weight of terms regarding some aspects, e.g. Term Frequency (**TF**) and Inverse Document Frequency (**IDF**). These features are the topic of the next section.

3.4 Feature Extraction

As stated in Section 2.4, a feature describes a characteristic or an aspect of an example, like a numeric description of the weight of a term in a Web page with **TF**. Choosing the appropriate set of features is paramount to the success of our approach. Here we describe how the features were computed and the rationale for their choice.

Most of the features used here are also used in previous work in keyword extraction like (Giarlo, 2005; Yih *et al.*, 2006; Wu and Bolivar, 2008; Matsuo and Ishizuka, 2003; Lacerda *et al.*, 2006) and tag suggestion (Song *et al.*, 2008b; Wang and Davison, 2008; Heymann *et al.*, 2008b; Suchanek *et al.*, 2008). Wordnet features as we measure them, Section 3.4.7, were not found in literature, although some papers use thesaurus knowledge (Chirita *et al.*, 2007).

Also, some features present in literature were not used, such as features using graph knowledge (Jäschke *et al.*, 2007; Song *et al.*, 2008a), association rules — i.e. associated tags are suggested when the other is (Heymann *et al.*, 2008b; Qu *et al.*, 2008), and **STS**-specific features like tag popularity and tags previously associated to a resource (Xu *et al.*, 2006).

All the features used to represent the terms can be organized into groups, according to the type of information they provide and the source used to measure it. We now describe each group and the individual features that compose them. The first groups concern Web page terms features and will be seen in Sections 3.4.1 to 3.4.5. After this, inbound features and Wordnet features are described in Section 3.4.6 and Section 3.4.7, respectively.

¹These values were obtained experimenting the trade-off between the increase of the classifier precision versus the time consumed by adding more examples.

3.4.1 Information Retrieval Features

Measures of term relevance such as **TF-IDF** have been traditionally used in many **IR** tasks ([Baeza-Yates and Ribeiro-Neto, 1999](#)) to create a **VSM** and represent documents, in Web search systems, for example. Here, we adopt some of these measures, as seen in the next sections. All these term statistics in this subsection, also used to other types of features, were computed using the Lucene index, [Appendix A](#), by means of the **IR** equations described in [Section 2.3](#).

Term Frequency

Term Frequency (**TF**), Equation [\(2.1\)](#), is the (normalized) number of times a term appears in the document. The idea is that a term which is recurrent in the text is likely to be important. It is used alone in papers like ([Matsuo and Ishizuka, 2003](#); [Wu and Bolivar, 2008](#)) or in combination with **IDF** when measuring **TF-IDF**.

Inverse Document Frequency

Inverse Document Frequency (**IDF**), Equation [\(2.2\)](#), is the inverse of the number of documents in which the term occurs divided by the number of documents in the collection. It is used to distinguish common words from rare words in the corpus. It is rarely used alone, most of the times it appears in combination with **TF** when measuring **TF-IDF**;

Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (**TF-IDF**) is measured by the product of the **TF** and **IDF**, Equation [\(2.3\)](#), normalized so that its values fall in the $[0, 1]$ interval. All other features measured by means of **TF-IDF** of a term in some specific part of the document — such as the title, [Section 3.4.2](#) — are also normalized.

This is the most important term weight in **IR** and the most used feature for keyword extraction ([Lacerda et al., 2006](#); [Yih et al., 2006](#); [Qu et al., 2008](#)) and tag suggestion ([Song et al., 2008b](#); [Chirita et al., 2007](#); [Heymann et al., 2008b](#); [Suchanek et al., 2008](#)).

3.4.2 HTML Features

Since we work with Web pages, HTML information is largely available. These HTML markups can give important insights about page structure, emphasizing regions of the document, and context information.

All HTML features have a value in the interval $[0, 1]$, corresponding to the normalized **TF-IDF** value of the word as shown in Equation (2.3), measured only within each corresponding HTML tag or group of tags. For example, a value for the title feature is the **TF-IDF** of the term considering only words in that HTML tag.

Title

The title of the Web page — in the `<title>` tags — describes the subject of the page and usually contains important terms regarding the page topic. It is used for tag suggestion in (Wu and Bolivar, 2008; Yih *et al.*, 2006).

Keywords

Terms within the `<meta name='`keywords`'>` meta tag may contain many relevant tags, given that these terms were chosen by the creator of the page with a similar purpose, i.e. to summarize the page topics using keywords. Wu and Bolivar (2008) and Yih *et al.* (2006) use this feature in keyword extraction.

Description

The `<meta name='`description`'>` meta tag is also chosen by authors to describe the page in some short sentences and might contain relevant tags. This feature is also considered in Wu and Bolivar (2008) and Yih *et al.* (2006) for keyword extraction.

Emphasizing Tags

Terms that occur within tags such as ``, `<big>`, ``, ``, `<h1>`, and `<h2>` are emphasized when viewing a Web page as they represent formatting options like boldface, section titles, among others. Therefore, we can expect them to be important terms to that page. This information is used in Web search (Brin and Page, 1998) and keyword extraction (Wu and Bolivar, 2008).

Text in Anchors

Text within the `<a>` tags, which point to another page through a link, is another well-know HTML source of information (Brin and Page, 1998) and, therefore, may contain relevant terms as well. It is explored for keyword extraction in (Wu and Bolivar, 2008; Yih *et al.*, 2006; Qu *et al.*, 2008).

3.4.3 Linguistic Features

Natural language information can also be used to determine the likelihood of a word being a good keyword (Hulth, 2004). All linguistic features were extracted using the GATE library, described in Appendix C.

Part-of-speech Tag

This feature represents the grammatical class of a word. The values are **1** if the term is a noun, **2** if it is an adjective, **3** if it is a verb, and **4** if it is any other. If the same word represents several classes in a text we choose the most frequent one.

In our dataset, most of the tags are nouns and adjectives, in this order. Rarely tags are verbs, adverbs, or other grammatical classes. Papers that explore this information in tag suggestion and keyword extraction systems include (Yih *et al.*, 2006; Hulth, 2004, 2003).

Average Length of the Sentence

It was used successfully in Wu and Bolivar (2008) and measures the average length of the sentences in which the term occurs, described in Equation (3.1):

$$ALS_i = \frac{\sum_k s_{ik}}{|\{k : s_{ik}\}|} . \quad (3.1)$$

In (3.1) s_{ik} is the length of a sentence k which contains term i and $|\{k : s_{ik}\}|$ is the number of sentences that contain term i .

Sentence Co-occurrence of Terms

It measures the sum of the number of times terms i and k appear together in a sentence, divided by the number of sentences where term i appears. Term k is a term selected as a positive training example for the classifier. It corresponds to the formula of Equation (3.2):

$$CO_i = \frac{\sum_k s_{ik}}{s_i} . \quad (3.2)$$

The idea behind Equation (3.2) is that words that co-occur often are probably part of a composite term (e.g., “Eiffel Tower”, or “White House”) and thus, if they co-occur with

other tags, they are likely to also be valid tags. This information was used as a feature in (Matsuo and Ishizuka, 2003).

3.4.4 URL Features

URL information may be important, given that many content management systems, blogging systems, and Web development frameworks use the URL to indicate the domain, category, and title of the pages.

Term Part of the URL

This is the only URL feature extracted and its value is simply 1 if the term appears in the URL and -1 if not. An example of work that employs this is Yih *et al.* (2006).

3.4.5 Text Features

This group of features includes text features which do not fit the previous ones.

First Occurrence in the Document

Terms that appear in the beginning of the Web page may be more important than the ones in footnotes. Thus, we use the position of the first occurrence of the term in the document as a feature. Thus, we use the value as measured in Equation (3.3):

$$FO_{ij} = \frac{f_{ij}}{l_j} . \quad (3.3)$$

In (3.3) f_{ij} is the position of the first occurrence of the term t_i in document j and l_j is the position of the last term in document j . Papers exploring these include (Qu *et al.*, 2008; Wu and Bolivar, 2008; Yih *et al.*, 2006; Hulth, 2004).

Capital Letters

Terms starting with a capital letter may indicate an important named entity such as a person, organization, or location. The feature value is 1 for terms starting with capital letters and -1 for the remaining terms. Wu and Bolivar (2008); Yih *et al.* (2006) are examples of keyword extraction systems using this feature.

3.4.6 Inbound Pages Features

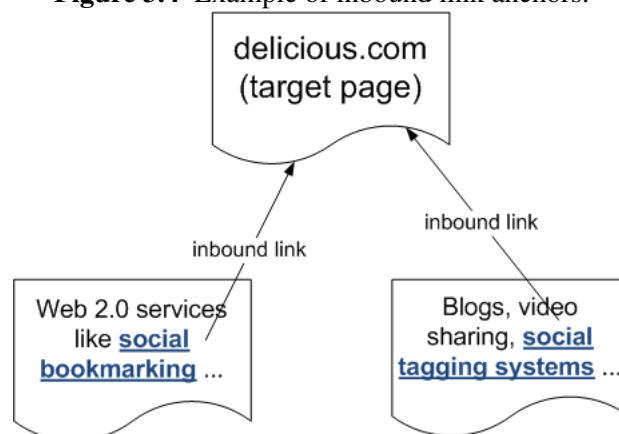
Keywords found in inbound pages, i.e. pages that link to target pages, are used in IR systems like Web search (Brin and Page, 1998). These pages might contain important information for suggestion, as seen in Section 4.2. Thus, features using this information were also used in ANTARES as in other tag suggestion systems such as Heymann *et al.* (2008b).

These are the first mentioned features where the information used is not extracted from the Web page itself. The feature in this section were extracted from terms in the pages of the Inbound Pages Dataset described in Section 4.1.2.

Inbound Link Anchors

The first and most obvious feature that we could extract from an inbound page is the content of the anchor text that is used in the hyperlink to the target page. Inbound link anchors are depicted in Figure 3.4, where the anchors are marked in underlined blue text and can have important keywords that maybe are not in the Web page — in this case, the target page was the Delicious page.

Figure 3.4 Example of inbound link anchors.



The value of the feature is the **TF-IDF** of the term when considered all terms in inbound link anchors for all inbound pages regarding a specific target page, i.e. **TF** and **IDF** are measured considering only terms in this collection of anchor terms where the terms are represented as a single document when measuring **TF-IDF**.

Kinsella *et al.* (2008) proposes a system solely based on this feature to tag suggestion and implies that anchor text was a primitive way of tagging Web pages before the advent

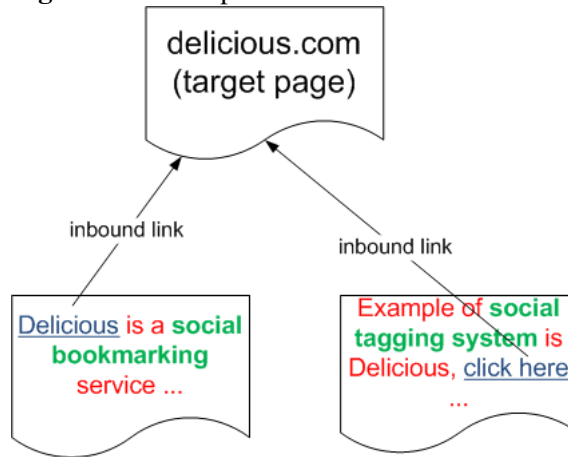
of STS, i.e. terms in anchors can be considered tags for the linked page. This feature is also used in Heymann *et al.* (2008b).

Inbound Link Context

While the anchor itself has a strong influence in the semantics of the hyperlink, the sentences that surround the inlink can also contain relevant keywords for the target Web page. Therefore, we define a context window of 25 terms² and extract the terms between the 25th term before the inbound link and the 25th term after the link.

The terms used for this feature are depicted in Figure 3.5, where the green terms can be considered good keywords for Delicious site. The hyperlink is in underlined blue text and the terms in red are also part of the context but no so important as the green ones.

Figure 3.5 Example of an inbound link context.



The value of this feature is the TF-IDF of the term considering only terms in this context sentences for all inbound pages for a target page, as in the previous feature.

Inbound Page Content

Besides the anchor text and the context of the inbound link, other parts of inbound pages might have important keywords. Therefore, we also get the TF-IDF of terms in inbound pages regardless the term's location in these pages.

²The rationale for 25 terms is that the sentences of our inbound corpus have 48.5 terms in average.

3.4.7 Wordnet Features

For the candidate terms collected from WordNet, described in Section 4.1.3, we used a specific set of features. These values were extracted using the JAWS library³. The idea is to use semantic relations to relevant terms in the target page and assign weights to the related terms based on these relations and use them as feature values.

Frequency of Occurrence

This feature value is the frequency with which a term appears as related to the terms from the target page, in all relations (synonyms, hyponyms, etc.). For example, in Figure 4.3, the term “related1” has frequency $F = 1$ and the term “related2” has frequency $F = 2$ because it was found by the synonymy relation with “term1” and by the hypernymy relation with “termN”. This feature is measure using Equation (3.4).

$$F_i = \frac{\sum f_i}{|R|} \quad (3.4)$$

In (3.4), Q are the terms from the target page used to query WordNet and R is the total set of related terms.

Tag Count

Tag count is a measure given by WordNet, counting the number of times the term — with a given meaning — was found in texts in the corpus used when building the WordNet thesaurus with the same meaning.

Overlap Count

This features measures the number of relations in which the related term appears related with the target page terms, divided by the number of target page terms used to query WordNet. This value can be found using the Equation (3.5).

$$O_i = \frac{\sum r_i}{|Q|} \quad (3.5)$$

In (3.5), Q is the set of query terms and $r_i = 1$ if $f_i > 0$.

³Java API for WordNet Searching, available at <http://lyle.smu.edu/~tspell/jaws/index.html>.

3.5 Classifier Combination

When testing the classifier only with Web page features we achieved high precision values but recall values only up to 42%, as described in the experiments of Section 4.3. So we were missing a lot of interesting tags that were not found in the Web page, as confirmed by our tag source analysis in Section 4.2. The idea was to use contextual information and to look for tags outside the page.

When we added Wordnet features to the classifier only with Web page features, a lot of examples only had WordNet features or the other way around. Therefore, the classifier was getting “confused”, or the results were bad, like precision of 10%, or the algorithm was not even converging during training — i.e. the classifier could not be created.

This happened because the feature spaces for each source are too different. Using all the sources and one feature space, the vectors are too sparse, because they might have null or low values for some features — dimensions in the feature space — for examples that only have features in one source.

An example of vector sparseness and SVM classifier “confusion” is depicted in Figure 3.6 that shows a feature space for examples using a Web page and a Wordnet feature. The examples in the axes represent examples that only have feature values for one source. We can see that there is no way of creating a hyperplane to divide the space, the attempts made in H_1 , H_2 , and H_3 can not divide the feature space into positive and negative examples and the SVM algorithm will not converge.

Therefore, we created three classifiers, each one specialized in a source of knowledge: the Web page itself, inbound pages, and related terms extracted from Wordnet. With these classifiers we could combine its output to give the final classification using rules like the product rule, explained in Section 2.4.3.

3.6 Suggestion Phase

After the classifier is trained, when we submit a Web page to the classifier, the terms from each source are extracted, analyzed, classified, and combined. The tag suggestion process — used, for example, when testing the classifier and in real suggestion scenarios — is depicted in Figure 3.7:

1. It starts by examining terms extracted from each source — candidate terms extraction;

Figure 3.6 Representing sparse examples from different sources in the same feature space and the “confusion” of the SVM algorithm.

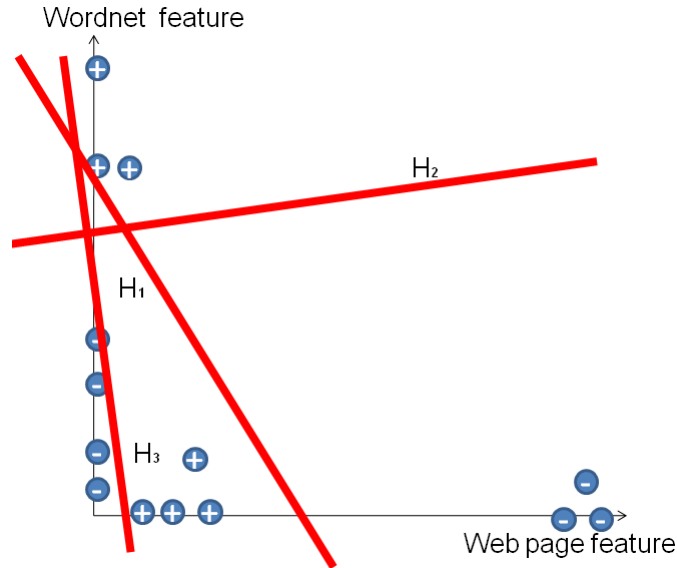
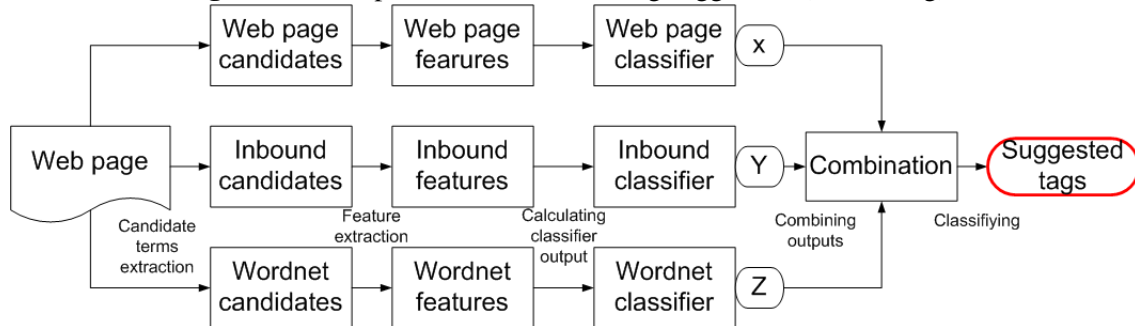


Figure 3.7 The process of ANTARES tag suggestion (and testing).



2. Then feature values are calculated to represent each term — feature extraction, based on the features explained in Section 3.4;
3. Afterwards, the terms are submitted to the classifiers specialized in each source and the classifier outputs are calculated;
4. Finally, the results of all classifiers are combined (combining outputs) to decide if the term is suitable to be a tag or not (classifying), with a formula like the product rule formula (2.11).

However, in some of the experiments in Section 4.3, some classifier with combinations of sources or using only one source are tested.

3.7 Implementation

ANTaReS was developed in Java programming language — except for the user-based evaluation, described in the next section, totaling 110 classes, divided in 30 packages, and 11649 lines of code.

Some required software components for ANTaReS are common in systems that use [IR](#) and [ML](#) techniques, therefore, many solutions are available for reuse. We reused some libraries regarding these and other requirements, as listed below:

- Apache Lucene: It is a text search engine written in Java. It was used to index Web pages in the [VSM](#) and to measure term weights like [TF](#) and [IDF](#), extensively used in feature extraction, Section [3.4](#). The library is described in Appendix [A](#);
- WEKA: A [ML](#) and data mining library that provided to ANTaReS an implementation of [SVM](#) and facilities to create the classifier, create the training and test sets, evaluate classifiers, among other functions. The library is described in Appendix [B](#);
- GATE: Library for Natural Language Processing ([NLP](#)) used to extract linguistic features of Section [3.4.3](#) and described in Appendix [C](#);
- JAWS: The JAVA API for Wordnet Searching⁴ was used to access Wordnet database and extract semantic relations between terms, used in Section [3.4.7](#);
- HTMLParser⁵: Library with facilities to parse HTML and, for example, to detect text in specific HTML tags — necessary to extract HTML features, presented in Section [3.4.2](#).

ANTaReS' code is divided in cohesive packages that handle specific and well defined tasks. The most important packages and databases of ANTaReS are depicted in Figure [3.8](#). The **main packages** provide the most important functions of the system such as dataset handling and indexing, feature extraction, and classifier training. Each package (rectangular box) in the picture represents a Java package in the code. The goals of each of these packages, from bottom to top, are listed below:

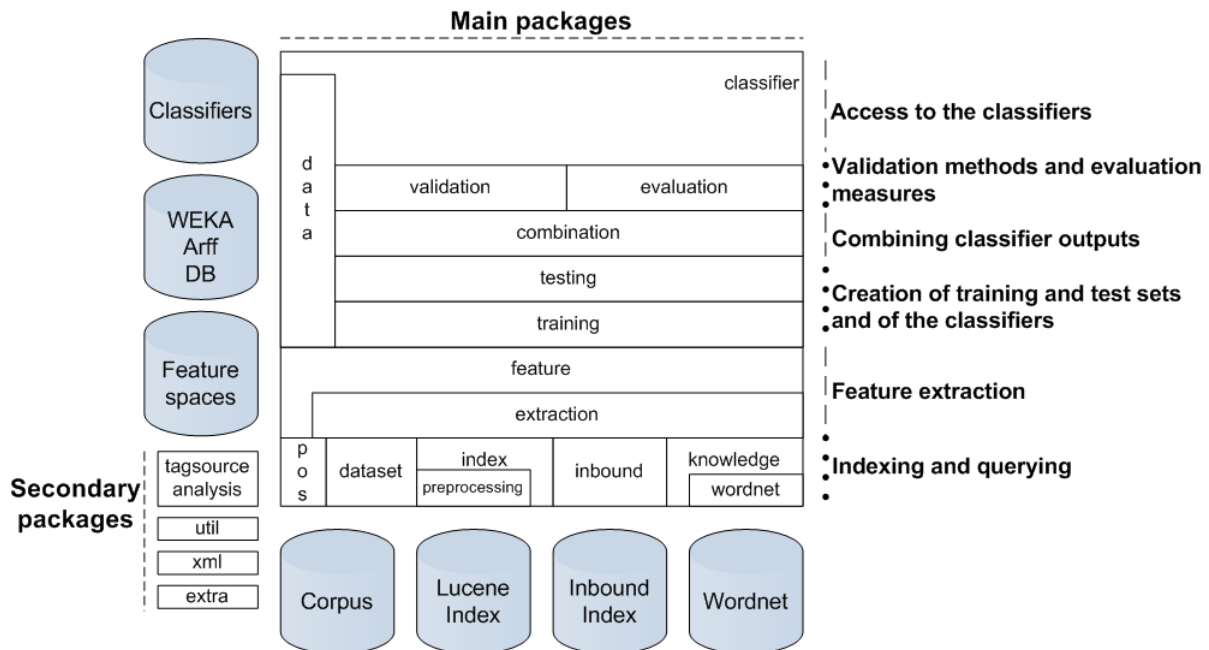
- Indexing and querying
 - dataset: It is responsible for accessing Delicious Web service to extract the dataset of URLs and tags — saved in an XML file⁶ and described in

⁴JAWS is available in <http://lyle.smu.edu/~tspell/jaws/index.html>.

⁵This library is available in <http://htmlparser.sourceforge.net>.

⁶This file can be seen in http://ontol.inesc-id.pt/delicious_dataset.xml.

Figure 3.8 Packages and databases of ANTaReS.



Section 4.1.1. Other function of the classes in this package is to download the HTML pages of these URLs, thus creating the Corpus database;

- **pos**: This package contains the classes responsible for handling the GATE library classes, Appendix C, and to access the Corpus database in order to extract linguistic features of Section 3.4.3;
- **index**: It uses the Corpus to create an index of the pages with Lucene library classes, Appendix A. It also accesses this index to calculate IR measures like TF-IDF, used in most of the features in Section 3.4;
- **preprocessing**: A sub-package of **index**, responsible for HTML parsing and preprocessing such as HTML tags removal, stop words removal and stemming;
- **inbound**: It treats the inbound pages extraction, Section 4.1.2, access to the Inbound Index, and it is responsible for inbound features extraction, described in Section 3.4.6;
- **knowledge**: Package responsible for abstract querying on knowledge sources, enabling the system to use different knowledge sources with little effort;
- **wordnet**: Sub-package of **knowledge** responsible for concrete querying

to Wordnet database to get the values of Wordnet features, described in Section 3.4.7.

- Feature Extraction
 - `feature`: Package with classes that handle the mechanics of how the features are measured such as `Document` (that encompasses document terms, document tags, methods to extract features of all document terms, and access to feature vectors for each term), `FeatureList` (that contains the features being used in this execution), and methods to feature normalization;
 - `extraction`: Sub-package of `feature` responsible to access the lower packages to extract features. Features and classes responsible for extracting them are mapped outside the code using an XML file, so it is possible to create different implementations of feature extraction without changing the code.
 - Classifier packages — the `classifier` package and sub-packages
 - Handling training and test sets and creating the classifiers
 - * `data`: Responsible for creating the Feature Spaces database for each source — using the `feature` package — and the conversion to the WEKA ARFF database, described in Appendix B.2. It also handles the access to ARFF files, representing levels of fold — in k-fold cross-validation, explained in Section 2.4.2, document, and term features, i.e. an ARFF file might contain the feature vectors for a whole fold, a specific document, or a specific term in a document. Finally, this package is also responsible for saving and loading `weka.classifier.Classifier` serialized classes — see Appendix B.4;
 - * `training`: It uses the WEKA ARFF database to gather training instances and create classifiers, which are serialized versions of `Classifier`, stored in `Classifiers` database;
 - * `testing`: Responsible for testing classifiers, with the creation of confusion matrices — as explained in Section 2.4.2.
 - Combining classifier outputs
-

- * `combination`: Package which is responsible for combining the outputs of several classifiers, using the methods like majority vote and product rule, as seen in Section 2.4.3.
- Validation methods and evaluation measures
 - * `validation`: Here the corpus division for validation is implemented, containing methods such as holdout and k-fold cross validation, as described in Section 2.4.2;
 - * `evaluation`: Set of classes responsible for measuring precision, recall, and F-1 values, as seen in Section 2.4.2.
- Access to the classifier
 - * `classifier`: It handles the classifiers and examples, wrapping WEKA classes such as `Classifier` and `Instance` — see Appendix B.1 and Appendix B.6.

The **secondary packages** in Figure 3.8 are responsible for other system functions such as the tag source analysis, examined in Section 4.2, and auxiliary classes handling data structures and other utilities, as listed:

- `tagsourceanalysis`: Package responsible for the tag source analysis of Section 4.2 based on the Feature Spaces database;
- `util`: It contains several utilities classes like data structures such as sets and specialized maps, string functions, logging, and access to Google Search API⁷ used to create the inbound dataset of Section 4.1.2;
- `xml`: Package for handling XML utilities such as parsing and saving;
- `extra`: Set of classes responsible for extra features like the creation of a database for the user-based evaluation in Section 4.4 and document statistics.

3.8 User-based Evaluation System

To perform the user-based evaluation, described in detail in Section 4.4, we reused similar solution that performs an evaluation of a tag suggestion system considering user's opinion (Oliveira *et al.*, 2008). The system is divided in two subsystems:

⁷<http://code.google.com/apis/ajaxsearch/>

- A Web application that gathers the opinions of the users regarding our tag suggestions and save this information in a database. The system was written in PHP;
- An evaluation system that uses the data generated for the Web application and calculates the measures specifically designed to evaluate the suggestions, as described in Section 4.4. This system is implemented in Python.

4

Experiments

To evaluate ANTaReS and test the accuracy of our methods, a set of experiments were designed and performed. This chapter describes how these experiments were made, presents their results, and discuss some conclusions derived from them.

First, we explain how the datasets used in the experiments were extracted and summarize their main characteristics. After, we perform an analysis on the datasets to infer the quantity of tags that can be found using them.

Then we describe an experiment comparing tags suggested by ANTaReS with tags assigned by users in a real Social Tagging System ([STS](#)) to infer the system's accuracy. Finally, a user-based evaluation is performed and factors other than accuracy were measured, such as user agreement on vocabulary and tag novelty.

4.1 Extraction of the Datasets

In this section we present how the datasets used in ANTaReS were extracted, their summary, and main characteristics. We begin by explaining how the Web pages dataset was extracted from Delicious. After, the process of creating the inbound dataset is explained, i.e. how we extract pages that have links to pages in the previous dataset. Finally we explain how Wordnet is queried using terms in the dataset Web pages to get semantically related terms.

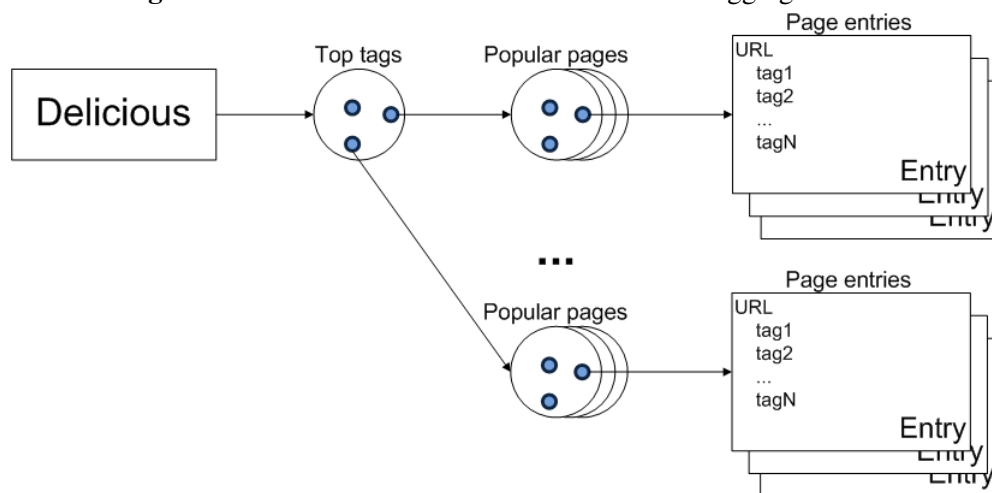
All datasets were preprocessed to remove HTML tags, stop words, and the terms were also stemmed. Except Wordnet dataset which is created directly in a feature space, all datasets have been indexed using Lucene library, described in [Appendix A](#).

4.1.1 Web Pages Dataset

For the analysis in Section 4.2, to train a classifier with Web page features, and to test the classifier we used a dataset extracted from the Delicious website¹. The extraction was made in February 2009, resulting in an XML file with a set of URLs, their corresponding list of tags and tag weights².

All data was extracted using the following steps, summarized in Figure 4.1:

Figure 4.1 Process of extraction of the Delicious tagging dataset.



1. The most popular tags shown in Delicious³ at the date of extraction were extracted, yielding a total of 199 tags;
2. For each popular tag, the most popular pages⁴ using it were retrieved, yielding an average of 7 pages per tag;
3. For each such page, the most popular tags used to describe it were extracted, using the “Look up a URL” option, described in Section 2.1.1, yielding an average of 9.8 tags per page;

¹The dataset is available at http://ontol.inesc-id.pt/delicious_dataset.xml.

²This weight is the number of times this tag was used to this URL, but the information is not used in ANTaReS because it is a Delicious-specific feature.

³The most popular tags of Delicious are found in a tag cloud in the URL <http://delicious.com/tag>.

⁴Popular pages that have a specific tag assigned can be found using a URL like <http://delicious.com/tag/howto>, that represents the popular pages associated with the tag “howto”.

All these queries to Delicious were done by consuming the Delicious Web service and Delicious feeds⁵. The queries were done using URLs such as <http://feeds.delicious.com/v2/json/popular/design>, which retrieves a JSON⁶ text file with a list of popular pages using the tag “design”.

As a result of this process, a corpus of 1,394 Web pages and 13,667 tags — with 2,201 unique tags — was collected. The Lucene Index has 47MB and the Corpus contains 66MB of HTML files, with 734,360 words, yielding 526.8 words per document in average.

Since only the most popular tags for popular resources are extracted, and since these tag assignments are made from registered users, the quality of this dataset is assumed to be high as the extracted tags represent terms that the community agreed on using to describe the resources. We can consider that the vocabulary of this dataset is closer to convergence and can therefore be used as a reliable baseline standard for comparison and analysis of a suggestion system. In Section 4.3 this dataset is used to test our tag suggestion system (i.e. the classifier).

Besides, in this extraction we got the terms that occur in the Web pages, in order to perform the analysis in Section 4.2 and train the classifier specialized in Web page terms and features.

4.1.2 Inbound Pages Dataset

The inbound pages dataset is used to the analysis in Section 4.2 and to train the classifier which uses terms and features from inbound pages, described in Section 3.4.6.

From the definition of inbound page given in Section 3.2 — a page that has a hyperlink to a target page, in our case, a page in the Web Pages Dataset — and Figure 3.2, it is obvious that we can possibly have thousands or millions of pages pointing to a target page. But we limit the number of inbound pages to 10 for each target page, for space and performance constraints.

To get the inbound pages URLs for a target page we queried the Google Search API⁷, which results in a JSON file with inbound pages URLs. The search for inbound pages can be tested in Google using the prefix `link:` in a simple Google search⁸.

⁵Instructions for using the service API and for querying feeds can be found in <http://delicious.com/help/api> and <http://delicious.com/help/json>, respectively

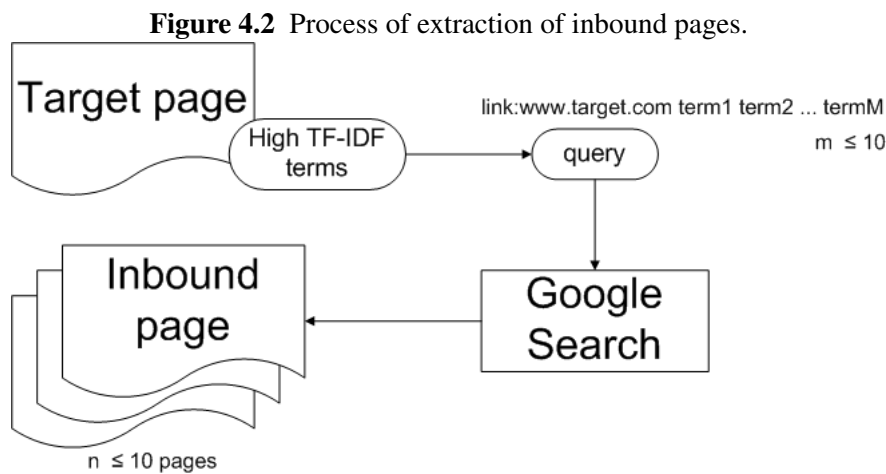
⁶JavaScript Object Notation is a data-interchange format popular in Web services which is composed by associative arrays, i.e. key-value pairs.

⁷<http://code.google.com/apis/ajaxsearch/>

⁸For example, using the URL <http://www.google.com/search?q=link:www.delicious.com>, the pages that link to Delicious website are shown.

But, using solely the target page URL resulted in poor results: for inbound pages of Delicious site, for example, pages in German, Arab, and Russian were retrieved, with few relevant pages related to Delicious.

In order to bypass this problem we used keywords to help the search: the 10 terms with the highest **TF-IDF** values in the target pages were extracted and added to the query, yielding better results. The whole process of inbound pages extraction is depicted in Figure 4.2.



In the end, this process resulted in an Inbound Pages Dataset of 7,173 pages (5.1 inbound pages per target page in average), totaling 646.9MB of HTML files and a Lucene index of 133MB.

4.1.3 Wordnet Dataset

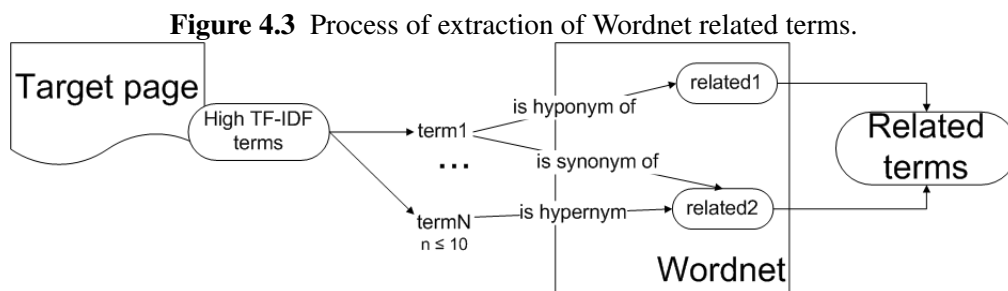
As the other datasets, the Wordnet dataset is used for the analysis of Section 4.2 and to train a classifier. It contains terms semantically related to relevant terms in the target page. Examples of relations — extracted from the Wordnet (Fellbaum, 1998) lexical database⁹ — used to build this dataset include:

- Synonymy: Words with these relation represent the same concept, e.g. “buy” and “purchase”;
- Hypernymy: It is a relation that happens when a word represents a broader concept than the other. For example, “dog” is a hypernym of “bulldog” and “poodle”, i.e. types of dog include bulldog and poodle;

⁹<http://wordnet.princeton.edu/>

- **Hyponymy:** It is the inverse of hypernymy, i.e. a hyponym is a word whose semantics is within the range of other, e.g. “house” and “library” are hyponyms of “building”. Therefore, library and house are buildings, or, library and house are types (kinds) of buildings;
- **Meronymy:** It represents a word that names a part of a larger whole, e.g. “door” and “window” are meronyms of “house”;
- **Metonymy:** It is the inverse of meronymy, i.e. it represents the whole-part relation. For example, “house” is a metonym of “floor”.

The extraction of this dataset consisted in selecting, for each target page, the 10 terms with the highest TF-IDF value. Then, for each term, we queried the WordNet thesaurus to obtain related terms, as depicted in Figure 4.3. All the related terms were considered candidate terms for suggestion — and thus added to the classifier, yielding a total of 26.2 terms per page, on average.



4.2 Tag Source Analysis

Since we aim at discovering tags besides those already present in the folksonomy, two research questions that naturally arise are:

1. Are there good tags within the textual content of the Web page being tagged?
2. If so, is the textual content of the Web page enough to find an acceptable number of good tags?

To answer these questions, we analyzed a dataset of Web pages and their respective tags, extracted from the Delicious tagging system. In this section, we describe the analysis performed.

We achieved high precision results — as stated in experiments of Section 4.3 — when testing a classifier using only Web page content — a dataset described in Section 4.1.1, mostly using information retrieval and HTML features. Therefore, the tags were correctly classified but the recall was low, a lot of interesting tags were “missing”. To increase recall other sources of knowledge where missing tags could be found were analyzed.

The first attempt used contextual information to extract tags by looking at hyperlink structure, a well-known way to find keywords to pages used in Web search engines (Brin and Page, 1998). The content of inbound pages was extracted as seen in Section 4.1.2 and compared to tags suggested by users. However, we found that this was not enough.

Another attempt was to consider semantics. The idea is to get a relevant term — where the relevance was measured with TF-IDF — in the target Web page and find terms with semantic relations with these relevant terms. For this we query WordNet (Fellbaum, 1998) and use the result, a set of related terms, to improve tag suggestions. The way Wordnet terms were extracted is described in Section 4.1.3.

Another knowledge source that could have provided semantic related terms by semantic relations and it was tested was the YAGO¹⁰ ontology (Suchanek *et al.*, 2007). But there was found that it was not effective because this ontology is based on Wordnet — thus being redundant with the previous source — and Wikipedia relations such as `diedIn`, `hasCapital`, and `locatedIn` — which were not very informative for tag suggestions

Therefore, to better understand the task of tag suggestion, we performed a quantitative analysis to get detailed insight on the different sources and associated tags, namely to see where we can find tags and what sources are more relevant. Thus, we were interested in computing the intersection between each set of candidate terms from a source and the set of tags proposed by Delicious users.

To exemplify, consider a target page and let $TAGS_i$ be the set of tags assigned to it by the users, T_i the set of candidate terms in the text of the target page, I_i the set of candidate terms extracted from the inbound pages, and W_i the set of candidate terms taken from WordNet. We want to compute the intersection between each set of candidate terms and the set of tags, as depicted in Figure 4.4.

In Figure 4.4, the sets $FT_1 = TAGS_1 \cap T_1$, $FI_1 = TAGS_1 \cap I_1$, and $FW_1 = TAGS_1 \cap W_1$ represent the tags found in a specific information source, corresponding to sets T , I , and W . Therefore, we find relations like $t_2 \in TAGS_i$ and $t_2 \in \{TAGS_i, FW_i\}$, summarized in

¹⁰YAGO stands for Yet Another Great Ontology and a demonstration can be found at <http://www.mpi-inf.mpg.de/yago-naga/yago/demo.html>.

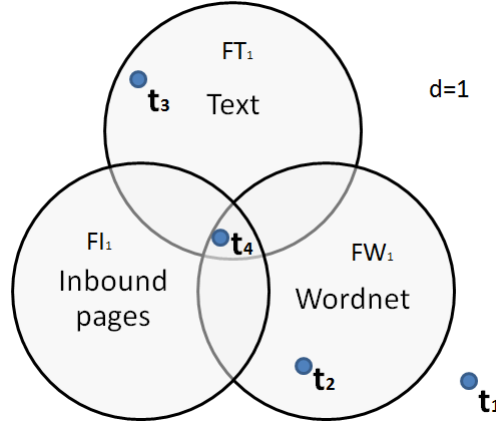
Figure 4.4 Example of tags versus information source analysis for page d_1 .

Table 4.1.

Table 4.1 Membership relations of tags described in Figure 4.4.

Tag	$\in TAGS_i$	$\in FT_i$	$\in FI_i$	$\in FW_i$
t_1	X			
t_2	X			X
t_3	X	X		
t_4	X	X	X	X

Using these sets and considering the whole dataset of term documents from each source and tags, we are interested in measuring:

- Tags found in a single source — measures the percentage of tags found in a specific source, for example for T_i we have:

$$\frac{\sum_i |FT_i|}{\sum_i |TAGS_i|} \quad (4.1)$$

- Tags found only in one source — measures the percentage of tags found only by one source, e.g. for I_i :

$$\frac{\sum_i |FI_i - FT_i - FW_i|}{\sum_i |TAGS_i|} \quad (4.2)$$

- Tags found by the intersection of at least two sources:

$$\frac{\sum_i |FT_i \cap FI_i| + |FT_i \cap FW_i| + |FI_i \cap FW_i|}{\sum_i |TAGS_i|} \quad (4.3)$$

Using equations (4.1), (4.2), and (4.3) for the example in Figure 4.4 we have:

- Tags found in a single source
 - Text: $2/4 = 50\%$
 - Inbound: $1/4 = 25\%$
 - Wordnet: $2/4 = 50\%$
- Tags found only in one source
 - Text: $1/4 = 25\%$
 - Inbound: $0/4 = 0\%$
 - Wordnet: $1/4 = 25\%$
- Tags found by the intersection of at least two sources: $1/4 = 25\%$

In Table 4.2 we summarize the results of the analysis for the extracted datasets. It is clear that the most relevant source is the actual Web page text, since it contains about 33% of the tags not found in any other source. It is closely followed by WordNet, which contains exclusively about 29% of the tags. Inbound pages show lower values — when used alone this source contains only about 8% of the tags.

Table 4.2 Analysis of the tags found in the three knowledge sources.

Tag found in ...	Web page text	Inbound pages	WordNet	$T \cup I \cup W$
Only 1 source	33.24%	7.85%	28.56%	—
At least 1 source	63.13%	19.27%	52.80%	—
At least 2 sources	—	—	—	28.55%

In any case, it is interesting to see that the combination of more than one source of information always contains more tags than each source in isolation. Thus, they can be seen as complementary, which justifies the use of sources other than Web page text and their combination in the final classification result.

4.3 System Evaluation

To test our classifiers, we designed an experiment with the 1394 pages dataset described in Section 4.1.1, that holds the relations of page and assigned tags extracted from Delicious. These documents were divided into training and test sets, using the 10-fold cross validation method, explained in Section 2.4.2. Except in Section 4.3.3 and when explicitly stated, the classifier used was the LibSVM in the WEKA library, with the default Radial Basis Function (RBF) kernel (Appendix B).

Precision, recall, and F-1 — the standard measures used to evaluate Information Retrieval (IR) and Machine Learning (ML) systems and described in Section 2.4.2 — were used in this experiment to evaluate the classification results. In all experiments we show the average of the results of these values considering all the 10 validation folds.

In tag suggestion, precision is said to be better than recall, because, for users, it is less likely to miss an absent tag and more likely to be annoyed at an inappropriate tag suggested (Wang and Davison, 2008).

Wang and Davison (2008) also notice that a method does achieve high precision when during training there are few positive examples and a lot of negative examples. This was taken into consideration when deciding the proportion of negative and positive examples of our classifiers, as seen in Section 3.3.

In the first experiments we test the impact of different sets of features in classifiers using one source and then we test the combination of classifiers. After, we compare different Machine Learning methods and quantitatively compare our work with related tag suggestion systems.

4.3.1 Impact of Features

In this section we test the impact of the features presented in Section 3.4 on the accuracy of our classifiers. This is important to understand the nature of the features, how valuable they are, and how can we adjust the feature extraction to obtain the best results.

Furthermore, we can use these information to improve the classifiers with methods such as feature selection or feature weighting (Wang *et al.*, 2004). For example, Yih *et al.* (2006) does an analysis on how feature weighting can enhance keyword extraction.

We start by evaluating each feature in isolation, in each source. By doing this, we can measure what features are the most informative for the classifier.

In Table 4.3, we see the results of percentage values for precision, recall, and F-1 measures for some of the Web page features, obtained when each term feature vector

is represented with a single feature. The best results in each column are indicated in boldface. Some of the tables in this chapter only show a summary of the data, and a complete version of the results can be found in Appendix D.

We note that, in this experiment and in the others where only one source is tested, we measured recall using only the tags present in the specific source — e.g., in Table 4.3 only tags in the Web page are considered, since only those can be retrieved when using this information source alone.

This “recall rule” is used similarly in the next results. For example, in Table 4.4, all terms have the “frequency of occurrence” feature, therefore, this feature alone is able to correctly classify all the tags in the dataset as such, i.e. a recall of 100%.

In Table 4.3, it can be seen that simple features, such as whether the words occur in the *Title* already yield high precision values, of about 90%. These features correctly classify tags that have good values in them, such as a high **TF-IDF** in the title, which explains the high precision. However, recall is very low in most cases, with values of about 10% through 30%, as these features are not able to correctly classify a good number of tags, as many tags do not appear in the title.

Also, the features that have the highest recall values and a reasonable precision value, such as **IDF** or *First Occurrence*, have the best result considering F-1 measure. The full table showing the result for each features is presented in Table D.2.

Table 4.3 Web pages features used in isolation (Summary). Complete version in Table D.2.

Feature	P	R	F-1
Title	94.05	33.82	49.75
Inverse Document Frequency (IDF)	51.31	99.78	67.77
First Occurrence	74.07	63.61	68.44

The inbound features used in isolation were not usable, since the classifier predicts that all examples are not suitable to be used as tags. Therefore, the denominator of equations (2.8) and (2.9) is zero and there is no value for precision, recall and F-1. This happens because these features alone are not enough to build a classifier that separates the feature space into positive and negative examples.

Wordnet features performed very well, with high precision and recall values, as seen in Table 4.4, except for the *overlap count* feature, which yielded very low recall. The improvement of F-1 values over Web page features happens because this classifier has a richer information source, i.e. semantic relations between terms are better than statistical measures such as **TF-IDF**.

The high recall values occur because the tags found in this source are closely related to the terms we use to query Wordnet, as seen in Section 4.1.3, therefore, they are classified correctly. In fact, for “Frequency of Occurrence” all tags that occur in Wordnet query results are correctly classified, yielding 100% of recall.

Table 4.4 Wordnet features used in isolation.

Feature	P	R	F-1
Frequency of Occurrence	86.44	100.00	92.72
Overlap Count	87.29	7.34	13.53
Tag Count	93.25	73.85	82.41

To further examine how the combination of features can influence the results, each feature was added in turn to the set used by each classifier for the Web page features. Using few features, precision values are high, while recall values remain low.

Adding more features brings an increase in recall values, while, on the other hand, seems to lead to a decrease in precision. This trade-off is not unexpected, since more features bring more information but also increase the probability of misclassifications. At the end, it was achieved a F-1 value of 75%, using all features. The full report on this experiment is found in Table 4.5.

Table 4.5 Adding each single Web page feature to the set of features.

Added features	P	R	F-1
Title	94.05	33.82	49.75
+Keywords	90.18	46.41	61.28
+Description	86.50	50.72	63.95
+Emphasizing Tags	85.35	52.31	64.87
+Anchor	87.45	53.55	66.72
+TF	87.12	53.85	66.56
+IDF	83.84	61.03	70.64
+TF-IDF	83.84	60.98	70.61
+First Occurrence	82.82	66.44	73.73
+Capital Letters	85.31	64.23	73.28
+Part of the URL	85.52	64.66	73.64
+Part-of-Speech Tag	85.29	67.34	75.26
+Average Sentence Length	83.42	69.35	75.74
+Sentence Co-Occurrence (all)	83.25	69.06	75.49

This test was not made for the other source features as they only have three features each and this is redundant with the next experiment.

Table 4.6 Removing a single Inbound page feature from the full set of features.

Excluded feature	P	R	F-1
NONE (i.e. all features)	90.25	8.56	15.53
-Inbound Link Anchor	95.24	4.38	8.37
-Inbound Link Context	100.00	3.75	7.23
-Inbound Page Content	45.53	1.90	3.65

With a similar purpose of testing combination of feature sets we experimented the results by removing each feature from the full set of features. For Web page features we observe that removing only one feature from the full set has little impact on the results. This behavior indicates that there is some redundancy in the information they carry, therefore, the results are stable and there are minor differences. The full information on this experiment is found in Table D.4.

The same test was also made for the other sources classifiers. For the Inbound features, the result is shown in Table 4.6. The results are not good, the recall is too low, meaning that the classifier misclassify a lot of “real” tags. In a classifier that does not converge when using only one feature, as seen for the Inbound features classifier, each feature is very important, therefore, it was expected that removing one feature would result in bad results.

This is also explained by the fact that the tags are present in the inbound pages but not with good **TF-IDF** terms, i.e. they are not so relevant in the inbound pages. The best result is achieved using all the features and will be the set of features chosen when we refer to the Inbound pages classifier.

The result for this analysis in Wordnet features is shown in Table 4.7. It is clear that the *Frequency of Occurrence* is the best feature, as when we remove it from the set of features, the F-1 value decreases.

It is also interesting to notice that in all combinations that involve this feature, i.e. when removing one of the other two features, the result is the same when testing all the features together. Also, this result is the same when considering only *Frequency of Occurrence* alone, as seen in Table 4.4, which indicates that this feature “dominates” the other features results. A classifier with this single feature will be used when we refer to the Wordnet classifier, as the results are the same.

In addition to these experiments, we also tested all possible combinations of Web page features. The combination that achieved the highest F-1 values was obtained using all features except **TF** and *Sentence Co-Occurrence*. The values obtained were 83.65%, 69.69% and 76.09%, in precision, recall, and F-1, respectively. This combination of

Table 4.7 Removing a single Wordnet feature from the full set of features.

Excluded feature	P	R	F-1
NONE (i.e. all features)	86.44	100	92.72
-Frequency of Occurrence	92.63	75.58	82.23
-Overlap Count	86.44	100	92.72
-Tag Count	86.44	100	92.72

features was used in all the following experiments when we refer to the Web page classifier.

4.3.2 Combination of Classifiers

In the following set of experiments, we combine the classifiers from different sources to see the best combinations and the best methods to combine them — i.e to discover the combination rules, explained in Section 2.4.3, that achieve the highest results.

First, we review the best results for each classifier one source in Table 4.8. With this we can compare the results and see whether the combination is useful or not. Differently from the previous section, in all recall values from now on we considered the whole set of tags regardless if they are present in the source or not, i.e. the “real” recall.

Table 4.8 Results for the classifiers focused in one source.

Classifier	Precision	Recall	F-1
Web page	83.65	43.28	57.05
Inbound	90.25	6.54	11.28
Wordnet	86.44	52.82	65.54

As can be seen the Web page and WordNet classifiers are very precise, although recall stays at about 43% and 53%, respectively. Surprisingly the Web page classifier was worse than the Wordnet classifier, which achieved the best F-1 value. Inbound classifier is even more precise than the other classifiers, but it suffers with a very low recall of 6.5% and therefore the F-1 value is also very low, 11.3%.

The rationale of these results were commented in the previous section and are reviewed here. The Inbound classifier has few information to achieve high recall values, in fact a lot of tags are not even present in inbound pages, as seen in the tag source analysis of Section 4.2. The difference between Web page and Wordnet classifier can be explained by the fact that Wordnet features relate terms semantically, providing richer information

than Web page features.

Comparing recall results with the tag source analysis in Section 4.2, we see that the recall value for the Wordnet classifier is almost equal to the number of tags found in them — Table 4.2, which means that almost all tags found in these source are correctly classified.

This does not happen with the Web page classifier, where the recall is 43%, although 63% of the tags can be found in it, which means there are a lot of misclassifications. The Inbound classifier also shows this behavior, where the recall is almost one third of the number of tags found in it — 19.3%. Again, the statistical nature of the features is the reason for this.

To enhance this results, we combined the classifiers and experimented all possible combinations of classifiers, using two combination methods — majority vote and product rule, explained in Section 2.4.3.

But first, we consider a detail about majority vote. In this method, precedence is important, i.e. in case of ties in the votes one classifier will give the final decision. We have tested all the orders of precedence for all the combinations possible, as presented in Table D.5. The best result follows the order $\{Wordnet, Text, Inbound\}$ and its is the one used hereafter.

Finally, the result with the combined classifiers is shown in Table 4.9, testing the majority vote and the product rule for all classifiers combinations. The rows represent the results for each combination method and the columns show different combinations. Each cell shows the values for precision, recall, and F-1, respectively.

Table 4.9 Results for the combined classifiers.

	Classifiers			
	Web page + Inbound	Web page + Wordnet	Inbound + Wordnet	ALL
Majority Vote	90.13 75.12 81.93	83.91 95.41 89.28	86.08 54.85 66.97	79.82 85.17 82.40
Product Rule	85.19 75.50 80.04	83.91 95.41 89.28	82.60 56.42 67.01	79.82 85.17 82.40

In comparison with Table 4.8, all the combinations in Table 4.9 performed very well and are better than using each source in isolation. It can be also noticed that the combinations involving the Inbound classifier do not perform well as the others. For example, the best F-1 result uses the *Web page + Wordnet* classifier and when the Inbound classifier is included for the combination with all the classifiers, precision and recall values decrease.

The results for the two combination methods are quite similar, in fact, the best result and the result for all classifiers are equal for the two methods. This can be explained by

the fact that posterior probabilities in Equation (2.11), estimated by the classifier, are two close to 0 or 1 and thus, do not interfere with the final decision, and the classification using product rule becomes equal to the one given by the majority vote method.

4.3.3 Comparing with other Machine Learning Methods

In order to assure that our choice of using a Support Vector Machine (SVM) classifier instead of other ML methods was correct, we tested other types of classifiers such as Naive Bayes (John and Langley, 1995), Multilayer Perceptron (Mitchell, 1997), K-Nearest Neighbor (Aha and Kibler, 1991), and different implementations of SVM: the SVM with Sequential Minimal Optimization (Platt, 1998) and the SVM implementation in SVMLight library¹¹.

We also have tested different kernel types for the SVM classifiers. Except for the SVMLight classifier, all other methods are implemented in WEKA classes, described in Appendix B.3. The results shown in Table 4.10 are for the product rule involving the *Web page + Wordnet* classifier, as presented in Table 4.9 only for LibSVM with RBF kernel.

In Table 4.10, we see that the results are very alike. All classifiers have the F-1 value ranging from 85 to 90%, except for the LibSVM with sigmoid kernel and the SMO with polynomial kernel, which achieved the worse results.

The classifier we were testing so far, LibSVM with RBF kernel, outperformed the other classifiers, except for the SVMLight. Obviously, we can also use the SVMLight classifier as our “final” classifier. It is also worth notice that the SVM classifiers are better than other methods.

4.3.4 Comparing with Related Work

A comparison with related work was also performed. Ideally, we would have a standard baseline dataset to compare the systems as in other research areas like text classification (Sebastiani, 2005). However, in this case, the authors of Song *et al.* (2008b) provided us with the dataset they extracted from Delicious and used in their experiments.

To create their dataset, they crawled Delicious from Oct 15 2007 to Jan 10 2008 and 50 tags were randomly sampled from the tag lists. Then, for each tag, they have retrieved the content of bookmarks and associated tags, yielding 45,715 unique items with 93,215 words and 8,792 tags. Table 4.11 presents the most frequent tags in the dataset.

¹¹Using the jnismvlight <http://www.mpi-inf.mpg.de/~mtb/> library to port the SVMLight library <http://svmlight.joachims.org/>, written in C, to Java. We used the polynomial kernel and default options.

Table 4.10 Experiment with different ML methods for classification.

Method	P	R	F-1
LibSVM with RBF kernel	83.91	95.41	89.28
LibSVM with linear kernel	79.95	93.71	86.27
LibSVM with polynomial kernel	82.94	94.92	88.51
LibSVM with sigmoid kernel	62.15	90.78	73.07
Naive Bayes	81.95	91.80	86.59
Multilayer perceptron	80.10	94.43	86.61
KNN	78.58	94.00	85.59
SMO with polynomial kernel	82.12	75.40	78.61
SMO with RBF kernel	80.14	93.68	86.37
SVMLight	87.22	94.27	90.59

Table 4.11 Most popular tags in the dataset used in Song *et al.* (2008b).

Tag Name	Frequency
internet	1743
technology	1543
java	1522
software	1473
web	1429
photography	1375
news	1328
music	1291
business	1115
travel	1092

To be able to compare the systems, we have used the parsed version of the URLs of the dataset, indexed the pages using Lucene and applied our techniques to perform the comparison, including our LibSVM classifier with RBF kernel using Web page and Wordnet best features, as presented in Section 4.3.2. Creating this new classifier with this dataset was very straightforward, which means that ANTaReS is able to work with different sets of data with simple adaptations.

Using a 50/50% ratio of training and testing data, as they did in Song *et al.* (2008b), we obtained the results shown in Table 4.12. In this work, they compare their results with Song *et al.* (2008a), so we are going to compare ANTaReS with these two works. Here we use the default LibSVM classifier with RBF kernel, although we shown that

SVMLight outperforms it.

Table 4.12 Comparing ANTaReS with [Song et al. \(2008b\)](#) and [Song et al. \(2008a\)](#)

Work	Precision	Recall	F-1
ANTaReS	68.58	43.19	52.95
Song et al. (2008a)	43.52	62.31	52.77
Song et al. (2008b)	47.38	66.16	54.23

From Table 4.12, we see that ANTaReS is in the same level as these works, which are some of the best papers in tag suggestion in terms of results. Comparing F-1 values, ANTaReS is slightly better (0.24%) than [Song et al. \(2008a\)](#) and slightly worse than [Song et al. \(2008b\)](#), with 1.28% of difference in F-1. Surprisingly, recall of ANTaReS is lower in comparison with these works, even though we use more sources of information than them.

However, precision of ANTaReS is significantly better than [Song et al. \(2008a\)](#) and [Song et al. \(2008b\)](#), which is important as we mentioned that precision is said to be better than recall for tag suggestion, because users do not usually miss an absent tag but get annoyed when an inappropriate tag is recommended ([Wang and Davison, 2008](#)).

Overall, this is a satisfying result as we shown that ANTaReS is comparable to state of the art methods for tag suggestion in terms of accuracy.

We have also compared our work with [Heymann et al. \(2008b\)](#), but as we did not have access to the dataset, we implemented a similar classifier and tested on our dataset. As this is a more indirect comparison, we only explain the experiment in Appendix E.

4.4 User-based Evaluation

It is important to test the suggestion with real users as they are the final target of the system. Therefore, we invited professors, researchers, and students that voluntarily tested our system. The classifier used was the LibSVM with RBF kernel.

For these experiments, we used a randomly chosen subset of our dataset, containing 44 Web pages. Each user evaluated 11 pages and each page was evaluated by 3 users. The evaluation was performed as follows:

1. Each user was asked to carefully read the page, paying attention to the page's topics;

2. After reading the page, the user was asked to manually assign 3 to 5 tags he/she believes can correctly describe it;
3. Afterwards, we show the user a set of 1 through 8 words, recommended by ANTaReS as tags to the current page;
4. The user is then asked to mark which of these suggestions are relevant tags.

To get our results, we used specific measures, based on precision and recall, using the tags submitted by the users as the reference set. For example, precision P is measured using the proportion of tags users considered relevant R_T — i.e. marked as relevant in step 4 — and all tags T shown to users in step 3, as shown in Equation (4.4):

$$P = \frac{|R_T|}{|T|} \quad (4.4)$$

Since each document was evaluated by 3 different users, it is also interesting to measure their agreement, or disagreement, when assigning tags. To this purpose, we considered three variations of the precision measure in (4.4):

- *precision1⁺*: A term is considered a correct tag — i.e. the term belongs to R_T — if it was assigned, or marked, as a tag by at least one user;
- *precision2⁺*: A term is considered a correct tag if it was marked as a tag by at least two users;
- *precision3*: A term is considered a correct tag if it was assigned by all users reviewing this document.

Another user-oriented evaluation measure we used is coverage C , similar to standard recall: here we compute the overlap of tags U assigned by users in step 2 when compared with our suggestions T in step 3, by means of Equation (4.5):

$$C = \frac{|T \cap U|}{|U|} \quad (4.5)$$

Finally, we wanted to know if the system reveals new relevant tags that users were unaware of. It is important to have a good novelty ratio which means the system suggests tags users could not recall or think of by themselves. To determine this, we applied the *novelty* measure, which is defined in Equation (4.6):

$$N = \frac{|R_T - U|}{|R_T \cup U|} \quad (4.6)$$

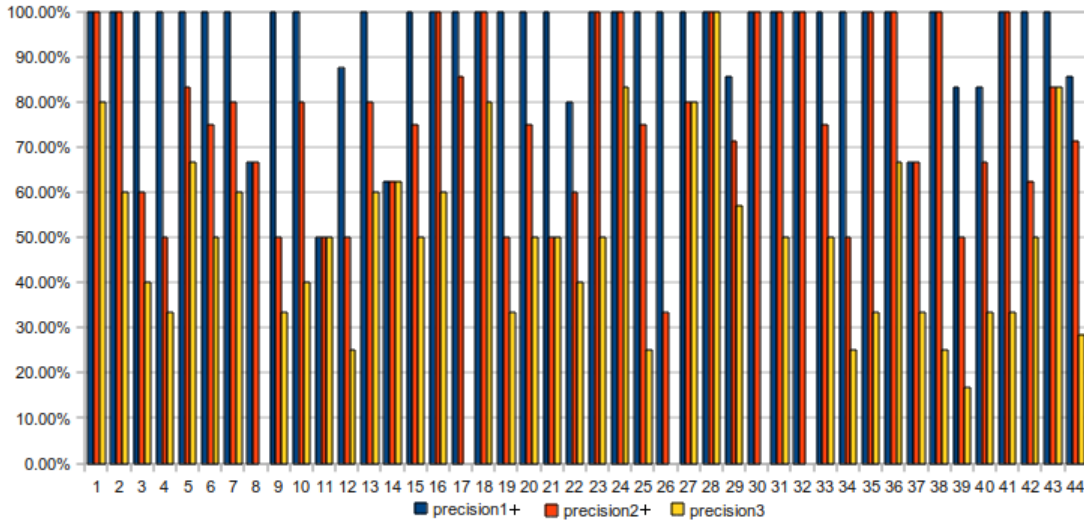
In Equation (4.6), R_T is a set containing the tags at least one user considered relevant and U is the set with the tags assigned by at least one user. In the following section, we present the results obtained in this experiment.

4.4.1 Results

The results of the user evaluation are shown Figure 4.5, Figure 4.6, and Figure 4.7, corresponding to the precision, coverage, and novelty measures, respectively. In all the graphics, the xx axis represents the Web pages and the yy axis represents the precision, coverage, and novelty results.

In Figure 4.5, we observe that precision results reach very high $precision1^+$ values for almost all pages, with an average value of 94%. Although somewhat lower, with an average of 77%, $precision2^+$ also shows very good results. The values of $precision3$ are clearly lower, indicating that user agreement is difficult to reach, even in an experiment with only 3 users. However, roughly half of the tags suggested (44%) are considered good, on average.

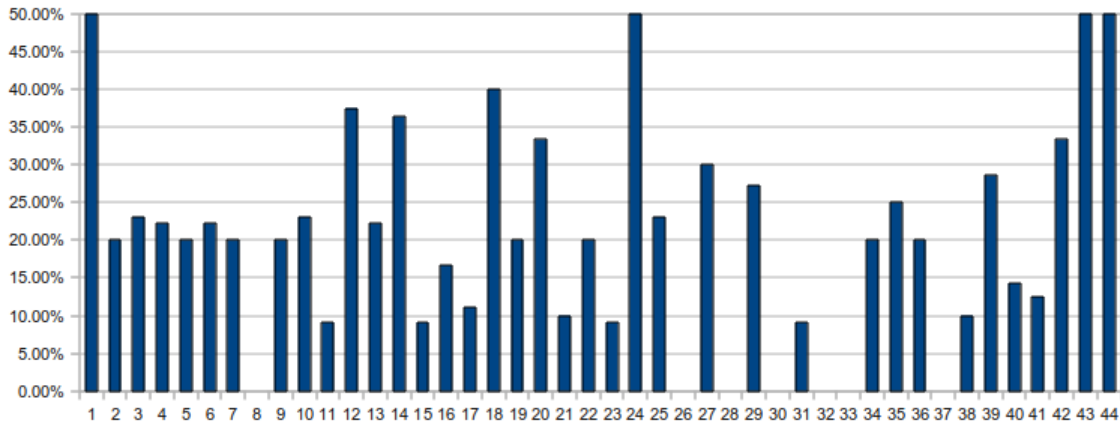
Figure 4.5 Precision results in the user-based evaluation.



Coverage values in Figure 4.6, on the other hand, are much lower, yielding an average of 20%. This is not surprising, since, as expected, the users and ANTaReS use different vocabularies — many of the tags suggested by ANTaReS were synonyms to those

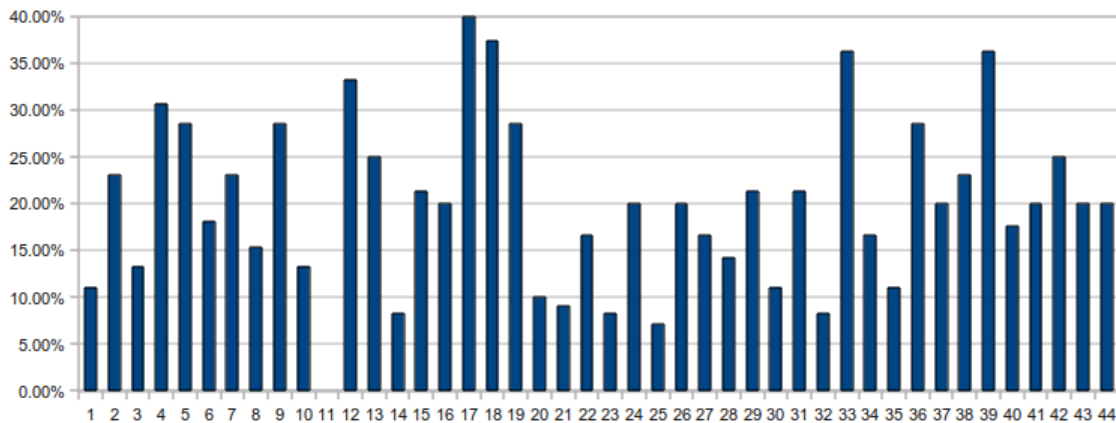
suggested by the users. Also, users assign a lot of binded words, such as “Website”, whereas our system suggests “Web” and “site” separately.

Figure 4.6 Coverage results in the user-based evaluation.



Finally, in Figure 4.7, novelty values were around 20%. This means that, on average, for every 10 relevant tags suggested by ANTaReS, 2 were not thought of by the users.

Figure 4.7 Novelty results in the user-based evaluation.



In Table 4.13, we summarize the results of the user-based evaluation using some statistical measures. The maximum values of each row are indicated in boldface.

4.5 Discussion

In this chapter we shown the experiments used to test ANTaReS’ accuracy. In this section we briefly discuss the overall results and the goals achieved in each experiment.

Table 4.13 Summary of user-based evaluation results

Measure / Experiment	<i>precision1</i> ⁺	<i>precision2</i> ⁺	<i>precision3</i>	coverage	novelty
Minimum value	50.00	33.33	0.00	0.00	8.33
Maximum value	100.00	100.00	100.00	50.00	40.00
Average	94.35	76.56	44.28	20.41	19.97
Standard deviation	12.05	19.72	24.57	14.36	9.06

At first, in Section 4.1, we described how the datasets used in ANTaReS were created and their main characteristics. We have also performed an analysis to check the best sources of information to find usable tags for suggestion, in Section 4.2.

This was important to understand the nature of the sources we had in hand, thus helping in the design of features for them. Besides, with this analysis we discovered an “upper limit” for ANTaReS, i.e. how many tags we can find for the classifier using these sources. In short, Web page text and Wordnet sources were the best in this analysis.

Then we have performed some experiments that were able to provide information of some idiosyncrasies of our dataset, classifier, and specially of the classifier features. Therefore, we could enhance our classifier results using the knowledge obtained through them.

To achieve this we tested features used in isolation, adding features in turn to the set of features, and removing a feature from the full set of features. Through these experiments we have learned which features perform well, “confuse” the classifier, or the ones that add little value to the final result.

After this we tested how combining results of classifiers for different sources of information could enhance the results (Section 4.3.2). Testing some combinations we reached the point to indicate which was our “best classifier”, the combined classifier using Web page text and Wordnet features. Results in this experiment were very satisfying as ANTaReS reached almost 90% in F-1 measure.

A different experiment was to try different ML methods for the classifier other than the LibSVM with the default RBF kernel used so far (Section 4.3.3). In general, SVM classifiers performed best, with slight differences regarding different types, implementations and libraries. This result justified our choice of using SVM algorithms in our classifier.

The greatest test for ANTaReS was to compare it with state of the art algorithms used in tag suggestion, which was done in Section 4.3.4. To perform this we used a dataset

provided by the authors of [Song *et al.* \(2008b\)](#) and tested our classifier on it. We achieved results that put our work in a good position when comparing with the state of the art methods found in [Song *et al.* \(2008a\)](#) and [Song *et al.* \(2008b\)](#) as we outperform the first and have a small difference to the latter.

Finally, we validated our system with an user experiment, where people were invited to evaluate our suggestions, thus simulating real use. The results were also good in this experiment, in which half of the suggested tags were considered good by at least one user and 20% of them were new and relevant to users.

5

Conclusion

In Social Tagging System, popular applications on the Web, users contribute to information retrieval and organization of resources like Web pages by assigning tags (keywords) to them. However, this is generally a manual process and users can be helped by a tag suggestion system that recommends relevant keywords to them.

In this work we developed a tag suggestion system called ANTaReS. Reviewing the goals stated in Section 1.1, the system should:

1. *Suggest tags regardless the presence of an existing folksonomy;*
2. *Be adaptable do any STS;*

After analyzing STS and actual solutions in tag suggestion systems, we used Information Retrieval (IR) and Machine Learning (ML) methods to create a classifier that has reached these two goals.

In ANTaReS a Web page is given as input and the system recommends relevant keywords to it, regardless tags previously assigned to pages or STS-specific features, thus fulfilling the goals.

3. *Be evaluated using a real tagging dataset;*

To be assured that ANTaReS will behave well in real environments, we tested tag suggestion using a real tagging dataset as described in Section 4.3, where we compared tags assigned to Web pages by users in Delicious and tags suggested by our system. This way, we could evaluate it using standard methods of ML classification systems such as precision and recall.

The results were very satisfying as we achieved 83% of precision, meaning that from 10 tag suggestions at least 8 were good, and 95% of recall, therefore,

from 100 tags we should suggest only 5 were not recommended by ANTaReS. This experiment proved that ANTaReS is able to be plugged in a STS and produce relevant results.

4. *Be compared with related work;*

To simulate a real environment is an important evaluation, but this is a research work and therefore it was also important to compare with related work to validate our approach for tag suggestion.

We compared ANTaReS with Song *et al.* (2008a) and Song *et al.* (2008b), coming to the conclusion that our work is in between this two in terms of F-1 measure with a small difference to the best one, i.e. our system has a suggestion quality comparable to these state of the art works.

5. *Be evaluated by users.*

Although we have experimented ANTaReS using a real tagging dataset, it was also important to simulate user-experience by performing an experiment with them evaluating the suggestions.

In this experiment we also had a confirmation that ANTaReS tag suggestion is relevant given that users considered that half of our suggestions were good and that 20% of our suggestion were good and new to them (i.e. they would not assign these tags to the page without the suggestion mechanism).

In short, we have fulfilled the goals planned for ANTaReS, that is a fully functional tag suggestion system. The results of the several experiments designed are very encouraging and we believe that the system is ready to be plugged in a Social Tagging System and put into use.

We also believe that the methods used in this work are a useful contribution to research in tag suggestion, therefore, researchers are able to reuse our approaches or develop new ideas by reading this work or other ANTaReS-related publications.

5.1 Future Work

Naturally, there is work to be done in ANTaReS. We propose a lot of extensions and improvements to the system, such as:

1. We have not fine tuned parameters of the ML classifiers we used in the experiments. This tuning could provide us better results than we have already;

2. Besides, a strategy for the classifier to improve its performance during use can be created, i.e. the classifier “learns” from experience in classifications, either by self-improving methods or user-feedback;
3. We could implement a classifier that instead of giving a binary “answer” that a term should or should not be used as tag, could recommend only a rank of the best terms to the users;
4. ANTaReS could be enhanced by using hybrid approaches, i.e. including in the classifier methods such as collaborative filtering and graph-like models of folksonomies. Although these methods require previous information of assigned tags, which is not possible in all scenarios, it is a valuable information for tag suggestion.
5. Our system is only able to suggest tags to Web pages with a good amount of text, as it is the most important source of information and it is required to create the second most important source (the related terms). We could suggest tags to videos and images as they have a URL and possible inbound links, but using surrounding text and other contextual information is probably the best approach, as seen in ([Song et al., 2008a](#)).

This is an important improvement, as assigning tags to images and videos is usually more difficult, and suggestion can facilitate the task to users.

6. Finally, a larger user-based evaluation could be performed, to simulate a real environment in a more realistic way. In this experiment, we could analyze how the presentation of the suggested tags influence their feedback and how the different information sources affect the results of precision, coverage and novelty.

5.2 Final Words

Tag suggestion is an important task that improves folksonomies, lightweight knowledge structures that are popular on the Web and used to organize content, among other functions. In this work we developed ANTaReS, a novel tag recommendation system, that uses multiple sources of information to suggest keywords to pages.

After performing several experiments, we believe that our system is ready for massive use and can be very relevant in present and future Web applications.

Bibliography

- Aha, D. and Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, **6**, 37–66.
- Angeletou, S., Sabou, M., and Motta, E. (2008). Semantically enriching folksonomies with flor. In *Collective Intelligence & the Semantic Web Workshop (CISWeb)*.
- Apache Software Foundation (2008a). Apache lucene - features. <http://lucene.apache.org/java/docs/features.html>.
- Apache Software Foundation (2008b). Apache lucene - index file formats. http://lucene.apache.org/java/2_4_0/fileformats.html.
- Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley.
- Bouckaert, R. R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., and Scuse, D. (2010). Weka manual for version 3.6.0. <http://prdownloads.sourceforge.net/weka/WekaManual-3.6.0.pdf?download>.
- Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *International World Wide Web Conference (WWW)*, pages 107–117. Elsevier.
- Brooks, C. H. and Montanez, N. (2005). An analysis of the effectiveness of tagging in blogs. *AAAI Spring Symposium on Computational Approaches to Analyzing Weblogs*, pages 9–15.
- Brooks, C. H. and Montanez, N. (2006). Improved annotation of the blogosphere via autotagging and hierarchical clustering. In *International World Wide Web Conference (WWW)*, pages 625–632. ACM.
- Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, **2**(2), 121–167.
- Cattuto, C., Loreto, V., and Pietronero, L. (2007). Semiotic dynamics and collaborative tagging. *National Academy of Sciences*, **104**(5), 1461–1464.
- Cattuto, C., Benz, D., Hotho, A., and Stumme, G. (2008). Semantic grounding of tag relatedness in social bookmarking systems. In *International Conference on Semantic Web (ISWC)*, pages 615–631, Berlin, Heidelberg. Springer-Verlag.

- Chirita, P. A., Costache, S., Nejdl, W., and Handschuh, S. (2007). P-tag: large scale automatic generation of personalized annotation tags for the web. In *International World Wide Web Conference (WWW)*, pages 845–854. ACM.
- Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. In *Meeting of the Association for Computational Linguistics*.
- Dellschaft, K. and Staab, S. (2008). An epistemic dynamic model for tagging systems. In *ACM conference on Hypertext and hypermedia (HYPERTEXT)*, pages 71–80. ACM.
- Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley, New York, 2. edition.
- Echarte, F., Astrain, J. J., Córdoba, A., and Villadangos, J. E. (2007). Ontology of folksonomy: A new modelling method. In S. Handschuh, N. Collier, T. Groza, R. Dieng, M. Sintek, and A. de Waard, editors, *Semantic Authoring, Annotation and Knowledge Markup Workshop (SAAKM)*, volume 289 of *Central Europe Workshop (CEUR-WS)*. CEUR-WS.org.
- Fellbaum, C., editor (1998). *WordNet: an electronic lexical database*. MIT Press.
- Giarlo, M. J. (2005). A comparative analysis of keyword extraction techniques.
- Golder, S. and Huberman, B. A. (2006). The structure of collaborative tagging systems. *Journal of Information Science*, **32**(2), 198–208.
- Gruber, T. (2007). Ontology of folksonomy: A mash-up of apples and oranges. *International Journal on Semantic Web & Information Systems*, **3**(2), 1–11.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explorations*, **11**.
- Hatcher, E. and Gospodnetic, O. (2004). *Lucene in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA.
- Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, **22**(1), 5–53.

- Heymann, P., Koutrika, G., and Garcia-Molina, H. (2008a). Can social bookmarking improve web search? In *International Conference on Web Search and Web Data Mining (WSDM)*, pages 195–206. ACM.
- Heymann, P., Ramage, D., and Garcia-Molina, H. (2008b). Social tag prediction. In *International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 531–538. ACM.
- Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. (2006a). Emergent semantics in bibsonomy. In C. Hochberger and R. Liskowsky, editors, *Informatik 2006 - Informatik für Menschen. Band 2*, volume P-94 of *Lecture Notes in Informatics*.
- Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. (2006b). FolkRank: A ranking algorithm for folksonomies. In *Fachgruppe Information Retrieval (FGIR)*.
- Hulth, A. (2003). Improved automatic keyword extraction given more linguistic knowledge. In *Conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics.
- Hulth, A. (2004). Enhancing linguistically oriented automatic keyword extraction. In *HLT-NAACL*, pages 17–20. ACL.
- Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., and Stumme, G. (2007). Tag recommendations in folksonomies. In *Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 506–514. Springer-Verlag.
- Jäschke, R., Hotho, A., Schmitz, C., Ganter, B., and Stumme, G. (2008). Discovering shared conceptualizations in folksonomies. *Journal of Web Semantics*, 6(1), 38–53.
- Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML)*, pages 137–142. Springer-Verlag.
- John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo. Morgan Kaufmann.
- Junichiro, M., Yutaka, M., Mitsuru, I., and Boi, F. (2004). Keyword extraction from the web for personal metadata annotation. *International Workshop on Knowledge Markup and Semantic Annotation (KCAP)*, pages 51–60.
-

BIBLIOGRAPHY

- Kim, H. L., Scerri, S., Breslin, J. G., Decker, S., and Kim, H. G. (2008). The state of the art in tag ontologies: a semantic model for tagging and folksonomies. In *International Conference on Dublin Core and Metadata Applications (DCMI)*, pages 128–137. Dublin Core Metadata Initiative.
- Kinsella, S., Budura, A., Skobeltsyn, G., Michel, S., Breslin, J. G., and Aberer, K. (2008). From web 1.0 to web 2.0 and back -: how did your grandma use to tag? In *WIDM '08: Proceeding of the 10th ACM workshop on Web information and data management*, pages 79–86, New York, NY, USA. ACM.
- Kittler, J., Hatef, M., Duin, R. P., and Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(3), 226–239.
- Lacerda, A., Cristo, M., Gonçalves, M. A., Fan, W., Ziviani, N., and Neto, B. R. (2006). Learning to advertise. In *International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 549–556. ACM.
- Leacock, C. and Ravin, Y., editors (2000). *Polysemy: Theoretical and Computational Approaches*. Oxford University Press.
- Lipczak, M., Hu, Y., Kollet, Y., and Milios, E. (2009). Tag sources for recommendation in collaborative tagging systems. In F. Eisterlehner, A. Hotho, and R. Jäschke, editors, *ECML PKDD Discovery Challenge*, volume 497 of *Central Europe Workshop (CEUR-WS)*, pages 157–172.
- Marlow, C., Naaman, M., Boyd, D., and Davis, M. (2006). Ht06, tagging paper, taxonomy, flickr, academic article, to read. In *Conference on Hypertext and Hypermedia (HYPERTEXT)*, pages 31–40. ACM.
- Mathes, A. (2004). Folksonomies cooperative classification and communication through shared metadata. <http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.pdf>.
- Matsuo, Y. and Ishizuka, M. (2003). Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, **13**, 2004.
- Mika, P. (2007). Ontologies are us: A unified model of social networks and semantics. *Journal of Web Semantics*, **5**(1), 5–15.

- Mishne, G. (2006). Autotag: a collaborative approach to automated tag assignment for weblog posts. In *International World Wide Web Conference (WWW)*, pages 953–954. ACM.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Education.
- Newman, R., Ayers, D., and Russell, S. (2005). Tag ontology. <http://www.holygoat.co.uk/owl/redwood/0.1/tags/>.
- Oliveira, B., Calado, P., and Pinto, H. S. (2008). Automatic tag suggestion based on resource contents. In *International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 255–264.
- Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press.
- Qu, L., Müller, C., and Gurevych, I. (2008). Using tag semantic network for keyphrase extraction in blogs. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1381–1382, New York, NY, USA. ACM.
- Rader, E. and Wash, R. (2006). Tagging with del.icio.us: Social or selfish? In *Conference on Computer Supported Cooperative Work (CSCW)*, pages 211–212. ACM.
- Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. In *Information Processing and Management*, pages 513–523.
- Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**(3), 210–229.
- Schmitz, C., Hotho, A., Jäschke, R., and Stumme, G. (2006). Mining association rules in folksonomies. In V. Batagelj, H.-H. Bock, A. Ferligoj, and A. Žiberna, editors, *Data Science and Classification. Proceedings of the 10th IFCS Conf.*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 261–270, Heidelberg. Springer.
- Sebastiani, F. (2005). Text categorization. In A. Zanasi, editor, *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, pages 109–129. WIT Press.
-

BIBLIOGRAPHY

- Song, Y., Zhuang, Z., Li, H., Zhao, Q., Li, J., Lee, W.-C., and Giles, C. L. (2008a). Real-time automatic tag recommendation. In *International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 515–522. ACM.
- Song, Y., Zhang, L., and Giles, C. L. (2008b). A sparse gaussian processes classification framework for fast tag suggestions. In *Conference on Information and Knowledge Management (CIKM)*, pages 93–102. ACM.
- Sood, S., Owsley, S., Hammond, K., and Birnbaum, L. (2007). Tagassist: Automatic tag suggestion for blog posts. In *International Conference on Weblogs and Social Media (ICWSM)*.
- Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *International World Wide Web Conference (WWW)*, pages 697–706, New York, NY, USA. ACM.
- Suchanek, F. M., Vojnovic, M., and Gunawardena, D. (2008). Social tags: meaning and suggestions. In *Conference on Information and Knowledge Management (CIKM)*, pages 223–232. ACM.
- Symeonidis, P., Nanopoulos, A., and Manolopoulos, Y. (2008). Tag recommendations based on tensor dimensionality reduction. In *Conference on Recommender Systems (RecSys)*, pages 43–50. ACM.
- Tanasescu, V. and Streibel, O. (2007). Extreme tagging: Emergent semantics through the tagging of tags. In P. Haase, A. Hotho, L. Chen, E. Ong, and P. C. Mauroux, editors, *International Workshop on Emergent Semantics and Ontology Evolution (ESOE)*, volume 292 of *Central Europe Workshop (CEUR-WS)*. CEUR-WS.org.
- Tonella, P., Ricca, F., Pianta, E., and Girardi, C. (2003). Using keyword extraction for web site clustering. In *International Workshop on Web Site Evolution (WSE)*, pages 41–48. IEEE.
- Waikato University (2010). Weka - programmatic use. <http://weka.wikispaces.com/Programmatic+Use>.
- Wang, J. and Davison, B. D. (2008). Explorations in tag suggestion and query expansion. In *Workshop on Search in Social Media (SSM)*, pages 43–50. ACM.
- Wang, X., Wang, Y., and Wang, L. (2004). Improving fuzzy c-means clustering based on feature-weight learning. *Pattern Recognition Letters*, **25**(10), 1123–1132.

- Wikipedia (2009). Confusion matrix — wikipedia, the free encyclopedia. [Online; accessed 19-January-2010].
- Wikipedia (2010). Support vector machine — wikipedia, the free encyclopedia. [Online; accessed 19-January-2010].
- Wu, X. and Bolivar, A. (2008). Keyword extraction for contextual advertisement. In *International World Wide Web Conference (WWW)*, pages 1195–1196. ACM.
- Xu, Z., Fu, Y., Mao, J., and Su, D. (2006). Towards the semantic web: Collaborative tag suggestions. In *Collaborative Web Tagging Workshop at International World Wide Web Conference (WWW)*.
- Yih, W.-t., Goodman, J., and Carvalho, V. R. (2006). Finding advertising keywords on web pages. In *International World Wide Web Conference (WWW)*, pages 213–222, New York, NY, USA. ACM.

A Apache Lucene

Apache Lucene is a text search engine written in Java that helps building search engines and other IR-based systems¹. Some interesting features of the library include (Apache Software Foundation, 2008a):

- Access to functions through a powerful API;
- Fast and scalable text indexing, with few memory and disk space requirements;
- Use of fast, precise, and reliable search algorithms such as ranked search, phrase queries, wild-card queries, proximity queries, among others;
- It is a multi-platform and open source solution, written in pure Java, and ported to other programming languages such as Perl, Python, C++, C#/.NET, Ruby, and PHP, where the indexes are fully-compatible regardless the language used.

One main factor for Lucene's success is its easiness of use, given by a powerful API for indexing and search. It is not necessary to know how the IR techniques work to use the library.

Lucene does not care about the source of the data, its format, or language, as long as you can convert it to text. Therefore, you can use the library to index and search data stored in Web pages, documents in local file systems, simple text files, Microsoft Word documents, PDF files, or any other format from which you can extract textual information (Hatcher and Gospodnetic, 2004).

A.1 Lucene Index

The Lucene index is an inverted index, as described in Section 2.3, with information about documents, positions, and term occurrences in the collection. Therefore, accessing this index can give us the values necessary to calculate IR measures such as TF and IDF.

Specifically, the term n_{ij} in the TF formula in Equation (2.1), that represents the number of occurrences of term i in document j , is given by the “Term Frequency Data” field of the Lucene index. The “Term Dictionary” field in the index provides the number of documents in which a term appears, needed for measuring IDF in Equation (2.2) (Apache Software Foundation, 2008b).

¹Lucene is available in <http://lucene.apache.org/>. We used the 2.4.0 version of the system.

A.2 Important Classes for Index Creation and Access

To the process of indexing and to access documents in the index, it is necessary to use some classes of the Lucene library². We give a brief summary of the most important classes used, following definitions in [Hatcher and Gospodnetic \(2004\)](#).

Document

A Lucene `Document` represents a collection of fields, objects of the `Field` class, described in next section. It can be seen as a virtual document representing a Web page, an email message, or a text file that should be retrievable in future. In ANTaReS every Web page is mapped in a `Document` object.

Field

Every document — object of the `Document` class — in index contains one or more labeled fields, where each field represents a part of the document data, representing metadata such as title, author, URL, or the full text content. In ANTaReS we use fields to store the entire content of the page, and specific fields regarding HTML tags, used in the HTML features presented in Section 3.4.2.

Term

The `Term` is the unit of search and stores a string representing the word from text and the name of the field the term is stored in.

Analyzer

Before effectively indexing the text, the content is filtered with a concrete implementation of this class, which is responsible for extracting the terms that will be indexed and filtering the rest of them. The analyzers are responsible for stop words removal, conversion to lower case, stemming, among other functions.

The concrete analyzers used in ANTaReS include:

- The `StopAnalyzer` removes stop words;
- The `SnowballAnalyzer` is responsible for the stemming terms;

²For a full description of Lucene classes and methods, users can consult the API Javadocs of Lucene 2.4.0 in http://lucene.apache.org/java/2_4_0/api/index.html

- The `PerFieldAnalyzerWrapper` creates different analyzers for groups of fields — every field is in lower case, except the “uppercase” field. This field does not include a lower case filter and it is used to extract words in capital letters, for the feature described in Section 3.4.5.

Directory

This class represents the location of the index and its concrete subclasses effectively store the index either using the volatile RAM memory with `RAMDirectory` or persistent files in the file system with `FSDirectory`. The latter one is used in ANTaReS.

IndexWriter

It is the main component in the process of indexing. This class can create a new index or add documents to a stored index. It can be seen as an object that gives write permissions to the index but that does not give permission for reading or querying it. `Analyzer` and `Directory` objects must be passed as arguments to this class in order to build the index.

IndexReader

It is the class responsible for reading the index and accessing information about term frequencies, term positions, retrieving `Document` objects from the index, among other functions. The most important methods used in ANTaReS are:

- `docFreq(Term t)` : returns the number of documents containing the term `t`, used in [IDF](#) formula of Equation (2.2);
- `document(int n)` : returns a `Document` object for the *n*th document in the index. This *n* is unique and it represents the insertion order extraction of the page in the index. But we can retrieve this number using the page URL, as every `Document` has a URL field;
- `getTermFreqVector(int n, String field)` : returns the term frequency vector — of class `TermFreqVector`, described in next section — for a specific document and a specific field.
- `numDocs()` : returns the number of documents in this index for measuring [IDF](#);

- `open(Directory directory)` : a static method that returns an `IndexReader` object by reading the index in the given `Directory`;
- `termDocs()` : returns an enumeration of $\langle document, frequency \rangle$ pairs of the type `TermDocs` for a specific term in the collection;
- `termPositions()` : returns an enumeration — `TermPositions` — responsible for retrieving term positions in documents, used for the feature in Section 3.4.5;

TermFreqVector

This class contains the methods `getTerms()` and `getTermFrequencies()`, which return arrays of the same size mapping terms and their respective number of occurrences in a document, used in TF calculation, Equation (2.1).

B WEKA

WEKA — Waikato Environment for Knowledge Analysis — is an open source library comprising ML and data mining algorithms³. For example, WEKA provides tools for data classification, clustering, and visualization. It can also be used as a framework for building new learning methods or adapting existing solutions (Hall *et al.*, 2009).

The system can be used to apply ML algorithms in existing data, through a user-friendly interface or to include these algorithms in the application code of a larger scope system, e.g. by embedding a classifier in it, as in ANTaReS. We cover some WEKA features used in ANTaReS by explaining how to build feature vectors, create training and test sets, and train, test, and use a classifier⁴.

B.1 Building Feature Vectors

A feature in WEKA is represented through the `Attribute` class. Every attribute must have a name and a type, which can be numeric, nominal, string, date, or relational. For building a feature vector for an example — e.g. a term in an inbound page and its inbound

³WEKA is available in <http://www.cs.waikato.ac.nz/~ml/weka/>. We used the 3.6.2 version of the library.

⁴For a detailed explanation of classes and methods for the WEKA current stable version, the API documentation can be found in <http://weka.sourceforge.net/doc.stable/>.

features, Section 3.4.6 — users have to create `Instance` objects and set feature values through its `setValue(Attribute attribute, double value)` method.

B.2 ARFF File Format

Before talking about creating training sets and a classifier in WEKA, we should talk about the ARFF — Attribute-Relation File Format. This type of files can be used to describe a list of instances sharing a set of attributes (Bouckaert *et al.*, 2010), which can then be used to create training or test sets. In ANTaReS, we use an ARFF file for every term in a document of every source. After this, we merge the data in ARFF files to create the training and test sets.

An example of ARFF file for the term “singl” — a stemmed version of “single” — in a document is shown in Table B.1, containing only the TF and IDF features. The nominal label attribute represents the class of this example, i.e. this example is not a tag.

Table B.1 An example of an ARFF file.

```
@relation singl

@attribute tf numeric
@attribute idf numeric
@attribute label {false,true}

@data
0.038462,0.222278,false
```

To save an ARFF file, the class `ARFFSaver` must be used. A collection of `Instance` objects — of the type `Instances` — must be set using the `setInstances(Instances data)` method of `ARFFSaver`. Also, the file that will be created must be specified with the `setFile(File arffFile)` method. Finally, the `writeBatch()` method creates the file.

To load an ARFF file, we simply pass a `Reader` object — such as `FileReader` — to the constructor of the `Instances` class and get the list of `Instance` objects stored in the file.

B.3 Training a Classifier

A classifier in WEKA is an object of the `Classifier` class. Representing learning methods, WEKA classifiers — subclasses of `Classifier` — include `NaiveBayes`, `MultilayerPerceptron`, and `LibSVM` (Bouckaert *et al.*, 2010). This latter, used in ANTaReS, is a wrapper made for WEKA of the LibSVM library, a popular library for SVM algorithms written in C⁵.

The WEKA classifiers, used in experiments of Section 4.3.3, are:

- The `weka.classifiers.functions.LibSVM` classifier, used in almost all experiments. It is a SVM implementation. Options used: `-B` to enable posterior probability output and `-K` number for choosing different kernel types. 0, 1, 2, and 3 represent the choice of linear, polynomial, RBF and sigmoid kernels, respectively.
- The `weka.classifiers.bayes.NaiveBayes` classifier, with default options, a simple probabilistic classifier based on applying Bayes theorem (Duda *et al.*, 2001);
- The `weka.classifiers.functions.MultilayerPerceptron` classifier, with default parameters. It creates a neural network with multiple layers (Mitchell, 1997);
- The `weka.classifiers.lazy.IBk` classifier, an instance-based classifier, also known as K Nearest Neighbor. We used the `-X` and `-K 50` options, that discovers the best K using hold-out evaluation in training data, where $1 \leq K \leq 50$;
- The `weka.classifiers.functions.SMO` classifier, a type of SVM, trained using Sequential Minimal Optimization. We used the `-M` and `-R` options, enabling posterior probabilities estimation and to use a radial basis function kernel. We have also tested the default polynomial kernel.

To train a classifier we must call the `buildClassifier(Instances data)` method in `Classifier`, using an `Instances` object (Waikato University, 2010). In ANTaReS, this object is created by merging the ARFF files for every term in every document of the training set.

⁵The wrapper is available at <http://www.cs.iastate.edu/~yasser/wlsvm>.

B.4 Saving and Loading Classifiers

We can save a `Classifier` object that is in memory to a file and we can load a file that contains a representation of a `Classifier`. This was helpful in ANTaReS in the training phase — Section 3.3, when we create three classifiers, one from each source, store these classifiers as files. When testing — Section 4.3, we convert the files back to `Classifier` objects to evaluate our methods.

The file that represents the `Classifier` object is a serialized version of the object, i.e. the file is in a format that can be transformed into the original object in the process of deserialization. Serialization is a method largely used to transmit objects in a network in distributed systems.

Serialization and deserialization can be done through two methods of the class `SerializationHelper`: `write(String filename, Object o)` and `read(String filename)`, respectively. The first one writes a file in the path defined in `fileName` with a serialized version of `Object o`, which can be a classifier. The method for reading uses the file path in `fileName` to deserialize the object.

B.5 Testing a Classifier

To test a classifier, WEKA provides the `Evaluation` class ([Waikato University, 2010](#)), that, given training and test sets — `Instances` objects — and a `Classifier` provides measures like precision and recall — Section 2.4.2.

Although WEKA has this facility, this evaluation could not be done directly in ANTaReS with `Evaluation` because the combination of classifiers is not supported. Therefore, we had to implement classes for calculating precision, recall, and to divide the training and test sets using the k-fold cross-validation — Section 2.4.2.

B.6 Using a Classifier

Suppose we have a `Classifier` object that is used for suggestion in a Social Tagging System (STS) and we must suggest tags for a Web page. After the page terms are preprocessed, the features of each term are extracted, and `Instance` objects are created for each term. Then, we can classify this term as suitable or not to be a tag using the method `classifyInstance(Instance instance)` of `Classifier` ([Waikato University, 2010](#)).

C GATE

GATE - General Architecture for Text Engineering - is an infrastructure, a framework, and a development environment for language processing software⁶, very popular in systems that use Natural Language Processing (NLP) techniques.

As an architecture, it defines the organization of such systems. As a framework, it provides reusable design and software components for NLP systems where engineers can use, extend and customize the solutions. As a development environment, it helps its users by aiding overall development and providing a debugging mechanism for new modules (Cunningham *et al.*, 2002).

C.1 Main Classes Used in ANTaReS

In this section we overview the most important classes⁷ used in ANTaReS to extract the linguistic features in Section 3.4.3.

GATE

This class is responsible for initializing the GATE components, and providing access to singleton utility objects, such as the GATE class loader. A singleton class is a class that can only have one instance per execution.

Document

This class represents an abstraction of a document, such as a text file or a Web page. One important method used is `getAnnotations()`, which returns a `AnnotationSet`.

AnnotationSet

It contains a collection of annotations — of the `Annotation` class, regarding features such as the Part-of-Speech Tagging, which determines the grammatical class of a term in text (Cunningham *et al.*, 2002), used in a feature of Section 3.4.3.

⁶GATE is available in <http://gate.ac.uk/>. We used the 5.0 version of the framework.

⁷The entire collection of GATE classes with their descriptions can be found in <http://hudson.gate.ac.uk/job/GATE-Nightly/javadoc/index.html>.

Annotation

It represents annotations about a term in text. For example, the method `getFeatures()` returns a `FeatureMap`, a collection of $\langle attribute, value \rangle$ pairs. If the *attribute* is the string “category” this map returns the value of the grammatical class of the term, i.e. the Part-of-Speech tag.

AnnotationUtils

For the features described in Section 3.4.3 regarding sentences, the `AnnotationUtils` class is used, through the method `annotationValues(Document doc, FeatureMap constraints, String... types)` to call the sentence splitter (Cunningham *et al.*, 2002). Passing a document as argument, a null `FeatureMap` as constraint, and “sentence” in types, we get a list of strings representing the sentences of a document.

D Detailed Experiment Results

This appendix describe in detail some of the results for experiments used to evaluate ANTaReS, presenting the full information about them, which appear summarized in sections 4.3.1 and 4.3.2.

D.1 One Web Page Feature Used in Isolation

Table D.2 shows the experiment where only one Web page feature was used to build a classifier and shows the values for precision, recall, and F-1. In this experiment we tested all the Web page features, described in sections 3.4.1 to 3.4.5.

We see more examples of the behavior mentioned in Section 4.3.1, where features with highest precision tend to have the lowest recall values and that features with high recall and reasonable precision values have the best F-1 values.

D.2 Adding Web Page Features to the Set of Features

Table D.3 shows the experiments in which we add a feature in turn to the set of features. We see that with few features the precision is low and when almost all of them are used the F-1 values stabilize at about 75%. The “last” feature that provides a significant increase in F-1 value is *Part-of-Speech Tag*.

Table D.2 Web pages features used in isolation (Complete).

Feature	P	R	F-1
Title	94.05	33.82	49.75
Keywords	87.90	24.21	37.96
Description	85.07	21.69	34.57
Emphasizing Tags	76.00	30.29	43.32
Anchor	62.70	58.86	60.72
Term Frequency (TF)	86.90	17.20	28.72
Inverse Document Frequency (IDF)	51.31	99.78	67.77
Term Frequency-Inverse Document Frequency (TF-IDF)	75.95	11.18	19.49
First Occurrence	74.07	63.61	68.44
Capital Letter	51.31	59.97	55.30
Part of the URL	91.41	17.72	29.69
Part-of-Speech Tag	70.22	41.53	52.19
Average Sentence Length	52.56	71.05	60.42
Sentence Co-Occurrence	59.00	40.32	47.90

Table D.3 Adding each single Web page feature to the set of features.

Added features	P	R	F-1
Title	94.05	33.82	49.75
+Keywords	90.18	46.41	61.28
+Description	86.50	50.72	63.95
+Emphasizing Tags	85.35	52.31	64.87
+Anchor	87.45	53.55	66.72
+TF	87.12	53.85	66.56
+IDF	83.84	61.03	70.64
+TF-IDF	83.84	60.98	70.61
+First Occurrence	82.82	66.44	73.73
+Capital Letters	85.31	64.23	73.28
+Part of the URL	85.52	64.66	73.64
+Part-of-Speech Tag	85.29	67.34	75.26
+Average Sentence Length	83.42	69.35	75.74
+Sentence Co-Occurrence (all)	83.25	69.06	75.49

D.3 Removing a Single Feature from the Full Set of Web Page Features

Table D.4 shows in detail the results of the experiment in which a single feature is removed from the full set of Web page features. We see that removing a single feature has little impact on F-1 values, that range from 73.08% to 75.74%.

Table D.4 Removing a single Web page feature from the full set of features.

Excluded feature	P	R	F-1
NONE (i.e. all features)	83.25	69.06	75.49
-Title	82.24	68.64	74.83
-Keywords	83.15	66.74	74.05
-Description	83.54	68.67	75.38
-Emphasizing Tags	83.35	68.86	75.42
-Anchor	82.65	68.19	74.73
-TF	83.20	69.35	75.65
-IDF	83.25	65.12	73.08
-TF-IDF	83.21	69.37	75.66
-First Occurrence	82.01	67.37	73.97
-Capital Letters	82.55	69.97	75.74
-Part of the URL	83.27	68.88	75.39
-Part-of-Speech Tag	82.51	66.48	73.63
-Average Sentence Length	85.29	67.34	75.26
-Sentence Co-Occurrence	83.42	69.35	75.74

D.4 Orders of Precedence in Majority Voting

This experiment consisted in testing different orders of precedence when combining classifiers with majority voting, i.e. in case of classifiers diverging about a classification output, the final result will be given by the most “important” classifiers. In Table D.5, the orders of precedence in the first column are represented using the letters *T*, *I*, and *W*, meaning the Web page **t**ext, **I**nbound, and **W**ordnet classifiers, respectively.

The best overall result — which achieves the best F-1 measure for all combinations — follows the order $\{Wordnet, Text, Inbound\}$ and is used in Table 4.9. This is justified by the results in the single classifier tests shown in Table 4.8, i.e. the best classifiers in that experiment must have more “power” in the voting.

Table D.5 Results for different orders of precedence for majority vote.

	Classifiers			
	Web page + Inbound	Web page + Wordnet	Inbound + Wordnet	ALL
TWI	90.13 75.12 81.93	82.50 75.12 78.63	86.08 54.85 66.97	82.20 76.15 79.05
TIW	90.13 75.12 81.93	82.50 75.12 78.63	64.16 15.85 25.40	82.18 76.15 79.04
IWT	85.19 75.50 80.03	83.91 95.41 89.28	64.16 15.85 25.40	79.82 85.17 82.40
ITW	85.19 75.50 80.03	82.50 75.12 78.63	64.16 15.85 25.40	79.82 85.17 82.40
WTI	90.13 75.12 81.93	83.91 95.41 89.28	86.08 54.85 66.97	79.82 85.17 82.40
WIT	85.19 75.50 80.03	83.91 95.41 89.28	86.08 54.85 66.97	79.82 85.17 82.40

Table E.6 Results for the experiment comparing ANTaReS with Heymann et. al (2008b).

Classifier	Precision	Recall	F-1
Page text	89.88	27.69	42.34
Inbound Link Context	—	—	—

E Comparing ANTaReS with Heymann et. al (2008b)

Besides comparison with Song *et al.* (2008b) and Song *et al.* (2008a), presented in Section 4.3.4, in which we were able to compare our system using the dataset used in their work, but this was not possible for comparing with the work of Heymann *et al.* (2008b). Therefore, we decided to implement a similar version of a related work in order to compare it to our system.

Although there is some obvious difference in implementation characteristics such as tag preprocessing, text preprocessing, proportion of negative/positive examples, method to extract inbound links, among others, we designed an experiment to compare ANTaReS with the system presented in Heymann *et al.* (2008b).

Heymann *et al.* (2008b) uses three SVMLight classifiers, each one based on a different source: the page text, inbound links context and surrounding pages — i.e. pages that have been linked in the target page. All classifiers use only a TF-IDF feature for each source. We tested the first two sources.

We resume the results in Table E.6. As seen, the classifier using Inbound Link Context did not converged, as in our experiments in Section 4.3.1. The result for the SVMLight classifier using TF-IDF of page text terms was very bad in comparison to ours. In fact, we reached the conclusion in Section 4.3.1 that classifiers with only one feature have worse results then when more features are added.

Therefore, using a similar classifier and feature set as the work in Heymann *et al.* (2008b), the results are worse than the ones we obtained with our methods.