FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# Ensemble Learning for Keyword Extraction

Masters' Degree in Informatics Engineering

Dissertation

Final Report

Pedro Geadas
pmrg@student.dei.uc.pt

Advisors:
Ana Alves
Bernardete Ribeiro
Francisco Câmara Pereira

Date: 4 de Setembro de 2013

*The personal selfishness, self-indulgence,*
*lack of generosity, small everyday cowardice,*
*all contribute to this pernicious form of mental blindness,*
*that is being in the world and not seeing the world,*
*or just seeing what, in each time, is likely to serve our interests.*

*O egoísmo pessoal, o comodismo,*
*a falta de generosidade, as pequenas cobardias do quotidiano,*
*tudo isto contribui para essa perniciosa forma de cegueira mental,*
*que consiste em estar no mundo e não ver o mundo,*
*ou só ver dele o que, em cada momento, for susceptível de servir os nossos interesses.*

JOSÉ SARAMAGO

# Acknowledgements

I would like to thank those that have been important during the process of completing this report. Without their support, I am sure that it would have not been possible to accomplish.

I start to thank to my coordinator Bernardete Ribeiro and to my co-coordinator Ana Alves for useful discussions and valuable support. I would also like to thank to my (omnipresent) coordinator Francisco Câmara Pereira for being one of the greatest teachers I had during my academic course.

In second place to my beloved parents, António José and Leopoldina, for giving me the opportunity of being where I am now and for allowing me to continue my studies, even if that has meant a lot of sacrifice from them, and also to my family, especially to my aunt Bia and my uncle Isidro who were literally like second parents to me, to my godmother São for being always present during a good part of my growth process and for teaching me how to read and do arithmetic when I was just a five years old man (ت). Also, I would like to thank, not that much, to my older and only brother, for being such an annoying creature during all my entire life.

In last but not least, to all my dear friends and colleagues, including all the new ones from the AmILab, from which I would like to emphasize Filipe Rodrigues for providing me his own implementation of one of the tools used in this work and for giving me additional support when it was needed, and Marisa Figueiredo for the "holy water"and LaTeX tips (ت).

I would like to give also a special thanks to Anette Hulth as well for pointing me out to valuable datasets, since some were eventually used throughout the experimentation phase of this work and to Su Nam Kim for maintaining the on-line repository which contains these.

Finally, to the Portuguese Science and Technology Foundation (FCT) for the financial support during this work.

Peace and ♡,
Pedro Geadas

# Abstract

Nowadays, the most relevant *events* occurring in the city are advertised on-line, generally through small textual descriptions. The exponential growth of the *Web* often hampers the task of finding relevant information, turning the existence of good *information extraction and summarization* methods in a necessity.

As such, the main goal of this dissertation is to develop an *ensemble learning* application for automatically extracting keywords from those *event textual descriptions*, since using human indexers is slow and expensive. Through rich *information on events*, one should be able to understand its *mobility implications* and possibly correlate both, allowing to foreseeing eventual repercussions that a specific event may cause in the city's normal behavior.

The proposed application intends to apply *Supervised Machine Learning* approaches, namely from known automatic keyword extraction systems, retrieving a set of keywords as output from the event descriptions usually found in the Web.

**Keywords**: Keyword, Keyphrase, Automatic Keyword Extraction, Ensemble learning, Artificial Intelligence, Machine Learning, Supervised Machine Learning, Information Extraction, Information Retrieval, Natural Language Processing, Events, Event textual descriptions

# Resumo

Hoje em dia, os *eventos* mais relevantes que ocorrem na cidade são anunciados on-line, geralmente através de pequenas descrições textuais. O crescimento exponencial da *Web* dificulta muitas vezes a tarefa de encontrar informações relevantes acerca dos mesmos, tornando a existência de boas técnicas de *extração de informação e sumarização* numa necessidade.

Como tal, o objetivo principal desta dissertação é desenvolver uma *aplicação de aprendizagem conjunta*, i.e, uma aplicação que utiliza um grupo de aplicações, e que desempenhe a tarefa de extração de palavras-chave a partir dessas *descrições textuais* automaticamente, pois a utilização de indexadores humanos é uma tarefa tanto lenta como cara. Para além do mais, através de um bom conjunto de informações sobre esses eventos, devemos ser capazes de compreender melhor as suas *implicações* a nível de *mobilidade* e possivelmente correlacionar ambos, permitindo assim prever eventuais repercussões que um determinado evento possa ter no comportamento normal da cidade.

A aplicação proposta pretende aplicar técnicas de *Aprendizagem de Máquina Supervisionada*, nomeadamente de sistemas de extração de palavras-chave automáticos já conhecidos, e deste modo obter um conjunto de palavras-chave a partir das descrições textuais desses eventos, encontradas usualmente na Web.

**Palavras-Chave**: Palavra-Chave, Extracção Automática de Palavras-Chave, Aplicação de Aprendizagem Conjunta, Inteligência Artificial, Aprendizagem de Máquina, Aprendizagem de Máquina Supervisionada, Extracção de Informação, Processamento de Linguagem Natural, Eventos, Descrições textuais de Eventos

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This chapter comprises six sections and presents the main reasons which led to the development of this work and its objectives. Section 1.1 presents its motivations while Section 1.2 provides most of the terminology and concepts approached in the remaining chapters. Section 1.3 summarizes the goals of this dissertation and Section 1.4 contextualizes it. In Section 1.5 the contributions of this research are depicted and finally Section 1.6 lays out the organization of the current report.

## 1.1 Motivation

Nowadays the most relevant *events* in the city are advertised on-line. However, it often becomes difficult to know exactly what is happening in a place: *information* is spread across too many websites, many times not easily understandable. The result of the World Wide Web (WWW) exponential growth is an huge amount of data chaotically organized, which turns out tasks like accessing, searching and keeping information a lot harder. With so many data drifting in the Web, most of the times not labelled nor categorized, searching for desired information is generally a *time-wasting* task.

Consequently, the existence of automatic *information extraction and summarisation* methods has become a growing necessity. Because such knowledge is mostly in Natural Language text, many different Information Extraction techniques emerged over the last few years. Such techniques provide insight about the content of a given document or article in a fast way, allowing one to quickly understand what the subject is, while minimizing the effort and time that one normally had to spend to perform the same task.

Contrary to what one might think, referring to *keyword extraction* is not exactly

the same as referring to *automatic keyword extraction*. While the former term refers to the task of choosing keywords from a document by hand, the latter refers to a way of doing that same task with minimal or no human intervention at all [Hul04b]. Therefore, the goal of *automatic extraction* is to apply the power and speed of computation to the problems of access and discoverability, adding value to information organization and retrieval without significant costs and drawbacks associated with human indexers, which are slow and expensive [Gia05].

In the context of this dissertation, the information consists on *textual descriptions* (namely about events occurring in the city) that reside in *websites* rather than in a document. Although one might say that textual descriptions are nothing less than text documents, which is perfectly acceptable, it is worth noting that the textual descriptions found in websites are usually a lot *smaller* than regular *documents or articles*. That fact may be significant when one pretends to automatically extract meaning from them, like it is the case on this work.

Regardless of the name by which they are called, *keywords* provide a simple and fast way to describe the main points present in textual sources in general, easily giving the reader some clues about its content. Moreover, *extraction techniques* are in fact very useful in a wide range of applications, such as retrieval engines, clustering, question-answering, named entity recognition, thesaurus construction and text mining applications [OPTJS10, SNG10, Gup10, Zha09], which portrays its importance.

By extracting the right information from event descriptions (like, for example, the type of the event or the specific place where it is going to happen), one should be able to understand, among other aspects, its *mobility* implications and possibly correlate both. This may be important for predicting which will be the most bustling streets within the city or even in order to perceive which transport means will most likely being used by that specific event audience, just to give a couple of examples now more contextualized in the ambit of the work presented here (see Section 1.4), which demonstrate the wide applicability arising for such techniques and proving themselves to be a crescent value these days.

## 1.2 Background

In order to understand the work being presented throughout this dissertation, one needs to get familiar with a set of background concepts first, such as *keyword* and *keyphrase*, *information extraction*, or even the concept of *ensemble learning*. Terms

defined in this section represent the basis of all the work presented ahead in this dissertation, being then a prerequisite for further understanding the work here depicted.

The *retrieval of information* from the Web (IR) consists in finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) [Alv11]. IR is often confused with the term of *Information Extraction* (IE). Despite related and many times simultaneously used, IE, contrary to IR, refers to the process of automatically extracting structured information such as entities, the relationships between them and even their attributes from the unstructured sources (if applied in a semantic context) or simply in extracting the most relevant set of terms of a given document or textual source, that can be useful to describe it and for many other different tasks, like summarisation.

Although, according to common sense, the terms *keyword* and *keyphrase* may refer to an *unique word* or *two or more words* respectively, the majority of the authors in the field consider them to be the same.

Ortiz et al [OPTJS10] says that a *keyphrase* may be considered as a sequence of *one or more words* that capture the main topic of the document, representing one of the key ideas expressed by the document author.

Similarly, Hulth [Hul04a] defines a *keyword* as being *one* or even a *small set of words* or *key segments* from a document, aiming to describe the meaning of that document, without the need of one to read it completely.

From now on, the preferred term will be *keyword* rather than *keyphrase*. However, because authors do not always use the same, both terms *will be referring* to the same subject and are going to be used interchangeably, depending on the author.

In the specific domain of *keywords/keyphrases*, there are two fundamentally different approaches: *keyword extraction* and *keyword assignment* [WPF+99].

That a keyword is *extracted* means that it is present verbatim in the document to which it is assigned, i.e., extracted keywords must *exist* in the document from which they were extracted [Hul04a].

On the other hand, keyword *assignment* seeks to select the phrases from a controlled vocabulary (in short, a controlled vocabulary represents a way to organize knowledge for subsequent retrieval) that best describe a document [WPF+99].

This work aims using the former approach, that of *extraction*, further detailed in Chapter 2.

The *training data* is a well known term in *Machine Learning* and depends of the specific problem in hands. In the context of this work, it consists in a group of documents with a set of human-chosen keywords previously assigned that need to be presented to supervised systems *a priori*, so they can apply the gained knowledge to find keywords from new unlabelled documents.

*Machine Learning* approaches can be roughly separated into two different categories: *supervised* (the one to be used throughout this work) and *unsupervised* [WX08]. The main difference between them resides on the fact that *supervised* learning methods *require the use* of training data, samples from the data source with the correct classification already assigned, while *unsupervised* ones *do not require* so [Oel09, Gup10]. The unsupervised approaches usually consists in ranking each of the candidate keywords using multiple features and heuristics and selecting the top rated ones [WX08]. One of the main contributions to the (supervised) field was KEA (*Keyphrase Extraction Algorithm*, described in Section 2.4), presented by Frank et al [FPW+99].

Apart from the kind of approach used and even before any classification can take place, a *preprocessing* phase is usually required. The objective of this phase is cleaning the training/testing data as much as possible by eliminating unnecessary words and characters, thus facilitating the classification phase itself.

For traditional text documents (no HTML tags), the tasks may include *stop-word removal*, *stemming* and other operations like handling of digits, hyphens, phrase punctuations and phrase case. For web-pages, additional tasks such as HTML tag removal and identification of the main content blocks also require careful attentions [Liu09].

*Stop-words* are frequent and meaningless words in a language. They belong to closed morphological classes, such as articles, prepositions and conjunctions. Stop-word removal is applied both in documents before indexing and storing, and in the query as well.

*Stemming* refers to the process of reducing inflected words to their stem, i.e, their base or root form. A stem is the portion of a word that is left after removing its prefixes and suffixes (e.g. ”Study”, ”Students”, and ”Studying” are reduced to ”Stud”).

Most of the times, what distinguishes a system from another is the *classifier* in which it relies on. The *classifier* is built during the training phase, i.e., when the training data is presented to the learner algorithm. In order to surpass the state-of-the-art methods, one basically intends to build a more accurate and precise classifier, a classifier that can label correctly more potential terms as keywords.

A different approach, is to use *multiple* already existing classifiers and *combine* their predictions, rather than developing a new one. That way, one expects to infer better keywords than a single classifier would do alone. This approach is addressed as *ensemble learning* and is depicted ahead in Chapter 3.

In the current paradigm, where the Internet growth leads to an increasing number of existing documents and web-pages on-line, from where we want to extract information, using human indexers becomes obviously impracticable. Thereby, enhancing such automatic methods should be seen as a crucial task not only presently but futurely as well, which is one of the motivations for the development of this work.

## 1.3 Objectives

The aim of this work is to create an *ensemble learning* application for automatically extracting keywords from textual sources in general and more specifically from *event textual descriptions*. The main idea here is *combining* some of the already *existing solutions*, aiming to refine *system specific extracted keywords*, instead of fully developing a new extraction algorithm.

In short, after receiving as input a set of small textual descriptions (like those usually found in web-pages), the application intends to apply *Supervised Machine Learning* approaches to the results from known automatic keyword extraction systems retrieving a set of keywords as output.

For the experimentation and validation of the intermediate and final applications, some widely known datasets[1] were used, which had been previously cited in many research work (e.g. [Hul03, MT04, SRC10]). These datasets are described in detail in Section 5.1.

Below, the main objectives for this dissertation are then summarized:

[1] https://github.com/snkim/AutomaticKeyphraseExtraction#readme

- develop an ensemble learning application that combines some of the already existing solutions, in order to obtain better keywords (from event textual descriptions) than those obtained by such methods when performing individually;
- test the chosen algorithms/systems separately with some widely known datasets, previously used in the field of keyword extraction or related;
- test the ensemble learning application with the same datasets, comparing the results obtained;

## 1.4 Project Contextualization

The work here presented is a part of a larger project, designated as *CROWDS*, which consists on understanding urban land use from digital footprints of crowds. One of the main goals of the CROWDS project is to understand how the *events* occurring within or close to the city correlate with *crowds' mobility patterns* around that place. Through semantics information found on the *event textual descriptions* discussed before (more specifically through *keywords* extracted from those descriptions), one pretends to analyse the implications arising from those occurrences.

An application like the one described here can be of great utility, since it allows foreseeing eventual repercussions that a specific event may cause in the city's normal behaviour, such as for predicting which would be the most bustling streets in the city, or even in order to perceive which transport means would most likely to be used, and when, by that specific event audience. In fact, it can be the case that an event occurring in a certain part of the city affects more than the surrounding areas and, in the extreme case, it can have repercussions even to neighbour cities.

Accordingly to what has been said, one can easily realize that knowing this kind of information in advance can certainly help the respective responsible entities taking any compensatory actions in accordance, if needed. Yet using the example of the transport means given above, a compensatory action could pass by increasing the frequency of buses near the event location or even in forbidding the use of private transport means in that same zone, thus allowing for a smoother and quicker dispersion of the crowds and consequent faster normalization of daily life in the city.

## 1.5 Contributions

Below, the main contributions of the research presented in this dissertation are summarized:

- A preprocessing tool for automatic keyword extraction from documents using Conditional Random Fields (CRF) approach;
- An empirical investigation showing that document structure have major impact in CRF learning;
- A base architecture for building an Ensemble Learning application;
- A methodology for combining Classifier Models of different already existing applications;
- An empirical investigation showing that an application like the one proposed can be successfully applied to the task of automatic keyword extraction both from scientific and non-scientific documents, the latter consisting off Event textual descriptions, in this work.
- An empirical investigation showing that the proposed ensemble application achieve better performance than the individual applications that compose it, up to a certain limit concerning the differences between the individual classifiers' reliability, and showing also better results than those reported in recent Keyphrase Extraction Contests (SemEval 2010[2]).

## 1.6 Organization

This report comprises six further chapters. Chapters 2 and 3 introduce two of the main concepts of the dissertation, namely those of Keyword Extraction and Ensemble Learning and respective related work. Additionally, several analysed tools are described and those used in the final application are emphasised. In Chapter 4 a detailed description of the project is given. The architecture of the application is presented as well as a description of it's most important components and operation modules. In Chapter 5, the experimental setup is depicted, which include a description about the datasets, the evaluation metrics and the methodology followed. Chapter 6 portrays the preliminary and final results obtained during the respective experimentation phases. The former concerning the results

[2]http://semeval2.fbk.eu/semeval2.php?location=Rankings/ranking5.html

of one of the systems while used in separate, while the latter those obtained by the final ensemble application itself. Finally, Chapter 7 concludes the work portrayed through this dissertation and addresses additional work to be done in the future.

# Chapter 2

# Automatic Keyword Extraction

This chapter is divided in seven sections. Section 2.1 gives an insight about Automatic Keyword Extraction (*AKE*) and its objectives while in Section 2.2 related work is presented. In Section 2.3 a description of some tested Information Extraction/Retrieval tools is given while Sections 2.4, 2.5 and 2.6 depict the algorithms/applications to be used in the ensemble (KEA, KUSCO and CRF respectively). Section 2.7 contains the conclusion of the chapter.

## 2.1 What is *AKE*?

AKE is the problem of automatically identifying the relevant words lying within a document and has been researched for more than half a century [Luh58, LD59]. While some of such methods rely on *statistical methods* and *linguistic knowledge* about the words in a document, recent works are focusing on *Machine Learning* techniques that may use that knowledge, among other *features*, achieving more interesting results than those methods individually.

*AKE* takes place mainly in *two steps* [Hul04b]:

- (a) - selecting candidate phrases in the document;
- (b) - filtering out the most significant ones to serve as keywords.

Nevertheless, a *third step* is typically performed as well (even before from those enumerated above) and it constitutes a pre-requisite for many existing extraction algorithms:

- (c) - the preprocessing phase.

The objective of this phase is then cleaning the input text as much as possible by eliminating unnecessary words and characters or, as it happens in some cases, just structuring the text, thus facilitating phases (a) and (b) listed above.

Despite being very useful for a wide range of tasks, extraction techniques may not perform as expected in every single situation. One disadvantage of some of these techniques concerns *ill-formed* and/or *inappropriate* phrases. Take as an example the phrase *"presented research"*. Although it could be extracted from a document (because it is a commonly used phrase in scientific research papers, like this one), this term is not meaningful to describe it [MW08], which underlines the identified problem.

## 2.2 Related Work

**Statistical**

Pure statistics based methods are simple and there is no need of previous training, once they rely only in the statistical information about the words, identifying keywords present in a document using merely that information. The benefits of purely statistical methods are their ease of use and relatively low computational power needed [Gia05].

Examples of statistical methods include word frequency [Luh58], word co-occurrence [MI04] and the *TF*IDF (Term Frequency - Inverse Document Frequency)* term weighting model ([SY73]) [Rob04].

Much work has shown that TF*IDF is very effective in extracting keywords for scientific journals (e.g., [FPW+99, Hul03, HKGM05]), [LPLL09]. TF*IDF-style measures emerged from extensive empirical studies of combinations of weighting factors, particularly by the Cornell group ([SY73]) [Rob04].

The *Term-Frequency* **(*TF*)** factor was originally presented by Luhn in 1958 [Luh58]. The author used the statistical information derived from *word frequency* and distribution to compute a relative *measure of significance*, first for individual words and then for sentences.

Later, in 1972, Sparck Jones [Jon72] presented a measure of *term specificity*, now known as *Inverse Document-Frequency* **(*IDF*)**, that is basically based on counting the number of documents in the collection being searched, which contain (or are

indexed by) the term in question. The intuition was that a query term which occurs in many documents is not a good discriminator, and should be given less weight than one which occurs in few documents [Rob04].

In short, one can refer to $\boldsymbol{TF}$ as simply being the number of times $\boldsymbol{n}$ a term $\boldsymbol{W_k}$ occurs in a document $\boldsymbol{d}$, and can be denoted by $\boldsymbol{n_{W_k,d}}$. This value can be normalized if the total number of candidate terms $\boldsymbol{|T|}$ is considered [Alv11, FPW+99], and can be denoted by the equation:

$$\boldsymbol{TF(W_k) = \frac{n_{W_k,d}}{|T|}} \tag{2.1}$$

However, since the $\boldsymbol{TF}$ representation can bias the document vector towards words with more occurrences, and coming in accordance with what was said in the preceding paragraph, it can be modified by the significance or rarity with which one occurs in all documents of the corpus [SR10]. That way, the *Document-Frequency* **(DF)** of a word corresponds to the number of documents in the collection in which the word $\boldsymbol{W_k}$ occurs and the *Inverse-Document-Frequency* **(IDF)** is then given by equation 2.2,

$$\boldsymbol{IDF(W_k) = \frac{|D|}{DF(W_k)}} \tag{2.2}$$

where $\boldsymbol{|D|}$ is to be the total number of documents in the collection [SR10].

Usually, instead of the $\boldsymbol{IDF(W_k)}$ itself, to avoid amplifying the importance of multiple occurrence of terms, some monotonous function such as the logarithm or the square root is used. Thereby, we can define TF*IDF, for example, as follows in equation 2.3, [SR10].

$$\boldsymbol{TFIDF(W_k) = TF(W_k) \times log(IDF(W_k))} \tag{2.3}$$

Another example of a statistical method was presented by Cohen [Coh95]. In is work he shows a fast statistical approach to highlight relevant terms from documents. Using almost no linguistic information, once it is a language and domain independent approach, the idea is to extract the resulting index terms representing the text by its *N-Gram* counts, using that statistical information to automatically index the terms in a document.

### Linguistics

Another slope of automatic extraction methods focus on the linguistic features of words present in the sources such as *part-of-speech* (PoS), *syntactic structure* and *semantic qualities*. In fact, many of these methods combine linguistic features with common statistical measures such as term-frequency (TF) and inverse document-frequency (IDF) [Oel09], earlier discussed.

Plas et al [vdPPRG04] uses for evaluation two lexical resources: the EDR electronic dictionary and Princeton University's freely available WordNet. The author showed that using lexical resources in such a task results in slightly higher performances than using a purely statistically based method, while Hulth [Hul03] examines a few different methods of incorporating linguistics into automatic keyword extraction (such as *syntactic* features) to improve the quality of the results. In particular she considers the *part-of-speech tags* as features to the classifier and looks only to*noun phrases* to be candidate phrases. Similar process is actually done by KUSCO as well, which operation is detailed ahead in Section 2.5.

### Supervised Machine Learning

As referenced before, supervised machine learning (that being used in this work) consists of training the system previously with a set of documents having a range of human-chosen keywords previously assigned, for applying the gained knowledge to label keywords from new documents.

A wide range of methods have been proposed for tasks of *automatic keyword extraction*, most of them being based on machine learning techniques [JHL09, WPHZ06]. These methods tend to give better results than those that only use statistical or linguistics information about the words in the documents [vdPPRG04], despite the fact that, in some cases, statistical or linguistic knowledge is used to enhance such algorithms.

In 1999, Turney [Tur99b] approached the problem as a supervised learning task, classifying words as positive or negative examples of keyphrases. In a first set of experiments, he applied the *C4.5* decision tree induction algorithm to that task. Then, he used a custom-developed algorithm, *GenEx*, concluding that incorporating specialized procedural domain knowledge could generate better keyphrases than a general-purpose algorithm (C4.5). The GenEx algorithm has two components: the Genitor genetic algorithm (Whitley, 1989) and the Extractor

keyphrase extraction algorithm (Turney, 1997, 1999). The Extractor takes a document as input and produces a list of keyphrases as output. The parameters of Extractor are tuned by the Genitor genetic algorithm, to maximize performance (fitness) on training data. Genitor is used to tune Extractor, but it is no longer needed once the training process is complete. Thus, when the best parameter values are known, Genitor can be discarded [Tur99b].

Perhaps one of the main contributions to the field, KEA (*Keyphrase Extraction Algorithm*) was presented by Frank et al [FPW+99]. The author showed that it generalizes as well as Turney's GenEx across collections with no need to use a special-purpose genetic algorithm (Genitor) for training and extracting like GenEx does. Therefore, training is a much quicker task in KEA than it is in GenEx. Once it will be used in the ensemble application proposed in this dissertation, a more detailed description is given in Section 2.4.

An improvement of KEA [FPW+99], called KEA++, has been proposed by Medelyan & Witten [MW06]. The main improvement over the first system is the use of semantic information on terms and phrases gleaned from a domain-specific thesaurus.

Other approach based on KEA [FPW+99], but relying on *bagged decision trees* instead of Naive Bayes for classification was proposed in 2009 by Medelyan et al [MFW09] and was called *Maui*. In addition to the new classification model, Maui enhances KEA's successful machine learning framework with semantic knowledge retrieved from Wikipedia. The new features tested focus on exploring the Wikipedia capabilities to further compute the semantic relatedness between terms or the likelihood of a phrase being a link in the Wikipedia corpus.

Sarkar et al [SNG10] presented in 2010 an algorithm that outperformed KEA in the experiences they conducted. This method, like the one proposed by Wang et al [WPHZ06], make use of Neural Networks. Candidate phrases are classified, either in keyphrases or not, based on features like *phrase frequency*, *phrase links to other phrases* and *inverse document frequency (IDF)*. This algorithm overcomes the problem in [WPHZ06], once that the authors consider keyphrase extraction to be a ranking problem rather than a classification problem, thus associating probabilities to the classes and sorting them in increasing order of their class probabilities. That way, if we want K+1 keyphrases, instead of K, we just have to pick the one that has the next higher associated probability. It is however obvious that, if we increase the number of desired keyphrases, the probabilities associated with the last picked ones can be smaller than what might be desirable.

Later in 2011, H. Kian and M. Zahedi [KZ11] introduced a new method for Per-

sian language, that uses a *genetic algorithm* based on results of popular *web search engines* to optimize the keyword extractor function. That way, they pretended to choose the keywords that were more important to search engines ranking algorithms, while achieving acceptable recall/precision scores at the same time. Extracted terms are evaluated and sorted by score function which uses statistical features of terms, and then top terms of the extracted keyword list are used to construct a search query. The search query is then submitted to popular search engines and the results obtained are analysed to rank query terms by a fitness function. As the authors refer on their work, to the best of their knowledge, no previous study has investigated similar method, so it can be the case that it works well also when applied to other languages.

Conditional Random Fields (CRF), today considered the state-of-the-art sequence labelling method, was first proposed by Lafferty et al [LMP01] back in 2001 and since then many published works explored this technique. Peng and McCallum [PM06] showed that CRFs outperform other methods at the task of extracting structured information, like the authors and citations from research papers. Zhang et al [Zha08] proposed the use of CRFs for the task of keyword extraction from Chinese scientific papers, presenting very promising results. Last year Feng et al [FYZ12] extracted keywords based on a combination of CRFs and specific document structure. The results presented improved dramatically the best published so far and were in fact the best results found meanwhile.

Similarly to KEA, CRF will also be used in the ensemble application proposed here, hence further details about the algorithm are given in Section 2.6.

#### Unsupervised Machine Learning

Despite the fact that it is not the focus of this dissertation, it is worth referring some of the most recently developed works under *unsupervised learning*. In a recent past, those methods tended to focus on graph-mining techniques, showing promising results.

In [MT04], Mihalcea and Tarau presented TextRank, a graph-based ranking model based on the co-occurrence relation between words, demonstrating that the approach outperformed the best published results so far.

Another example is the work presented by Grineva et al [GGL09]. The author showed a novel method for key term extraction from text documents, which uses a graph of semantic relationships to model the document, extracted from Wikipedia, and an algorithm for detecting community structures of a network. Evaluations done by the authors show that it outperforms existing methods, producing key

terms with higher precision and recall.

Ortiz et al [OPTJS10] uses the combination of two techniques: maximal frequent sequences and PageRank [PBMW99] (also a graph-based approach) for selecting the most prominent terms of a given text.

By its turn, Rose et al [SRC10] presented RAKE, acronym for Rapid Automatic Keyword Extraction, showing that RAKE is more computationally efficient than TextRank while achieving higher precision and comparable recall scores. Also, RAKE is domain and language-independent extracting keywords from individual documents.

In addition to the graph-mining techniques depicted, there are another unsupervised approaches that are worth referring.

Yutaka Matsuo and Mitsuru Ishizuka [MI04] presented a new keyword extraction algorithm that applies to a single document without using a corpus, identifying the importance of a term in a document by the co-occurrence between each term and the frequent terms. The two main advantages of this method are its simplicity, once using a co-occurrence matrix instead of a corpus enables keyword extraction using only the document itself, and its high performance comparable to TF*IDF.

Liu et al [LCZS11] realizing that appropriate keyphrases are not always statistically significant or even maybe do not appear in the given document, propose to use word alignment models **(*WMA*)** in statistical machine translation to learn translation probabilities between the words in documents and the words in keyphrases. The results presented outperform existing unsupervised methods on precision, recall and F-measure and the suggested keyphrases are not necessarily statistically frequent in the document, which indicates that this method is more flexible and reliable than previous ones.

## 2.3 Analysed tools

As described previously in this chapter, for creating an ensemble one needs to combine several classifiers. Thereby, and having that in mind, several known IE tools were analysed in order to exploring its capabilities and finding the tool best suited for the job in question. Despite revealing also some interesting features, the tools here addressed ended up not being used, either because they are proprietary and one cannot freely access the source code of the same, or just because they did not function as desired. However, this section presents a brief description on

several other tools that were also investigated.

### OpenCalais

OpenCalais is a web service available through an API that accepts unstructured text (like news articles, blog postings, etc.), processes it using natural language processing (NLP) and machine learning (ML) algorithms, returning RDF-formatted entities, facts and events extracted from that text. OpenCalais labels as *entities* things like people, places or companies. *Facts* can be relationships, like *Pedro Geadas is a student of Informatics Engineering*. Likewise, *events* represent things that happened, like *there was a natural disaster of landslide* in place *Portugal*. Beyond this superficial explanation about how OpenCalais processes and returns the processed information, there is not much more information available.

OpenCalais is an proprietary software, turning out impossible to find both mechanisms and the respective algorithms in which it relies. Another disadvantage, concerns its daily limited rate of transactions allowed. In fact, one can send as much as four transactions per second and fifty thousand transactions per day (for the free version), depending on how big are the documents sent.

### ANNIE - A Nearly New IE system

ANNIE is an open-source tool designed for Information Extraction (IE), supporting many natural language processing tasks, like *tokenising*, *sentence splitting*, *POS tagging* or *named entity recognition* (for more NLP tasks available, see [1]). A live demo showing ANNIE capabilities is available in [2].

The *tokeniser* simply splits text into simple tokens, such as numbers, punctuation, symbols, and words of different types (e.g. with an initial capital, all upper case, etc.). This enables greater flexibility by placing the burden of analysis on subsequent tools, and it means that the tokeniser does not need to be modified for different applications or text types. The *sentence splitter* segments the text into sentences. This module is required for the *tagger*. Both the splitter and tagger are domain and application-independent. The *tagger* is a modified version of the Brill tagger, which produces a *part-of-speech* tag as an annotation on each word or symbol. Neither the splitter nor the tagger are a mandatory part of the IE system,

[1] http://gate.ac.uk/sale/tao/splitch6.html#chap:annie
[2] http://services.gate.ac.uk/annie/

but the extra linguistic information they produce increases the power and accuracy of Annie's IE tools. The *named entity recogniser* (NER) recognizes entities like person names, organizations, locations, money amounts, dates, percentages, and some types of addresses as well.

Annie further supports multiple languages through Unicode and has been adapted to do IE tasks in Bulgarian, Romanian, Bengali, Greek, Spanish, Swedish, German, Italian, and French (support for other languages, like Chinese and Russian, are under current development as well) and can be used and customised in GATE's graphical development environment and integrated in other applications through its API.

ANNIE function as follows: after choosing the type of term one pretends to find within the textual source, it just underlines all the terms found, telling not which ones are the most relevant. Thus, despite being able to recognize things like persons, organizations and dates, this system is not suitable for the work in question because it does not retrieve a set of ***N*** *ranked* terms, instead just highlighting the ones it finds from a list of preselected types.

#### TextWise

TextWise has recently developed *Semantic Gist* to provide intuitive *semantic modelling* on a large number of samples, particularly vertical text documents that often do not have classification schemes associated with them. These semantic models will automatically adapt to rapidly changing content, ensuring a high level of accuracy over time.

Semantic Gist represents a significant advance in the use of machine learning, image and speech characterization, relying in *neural networks* to attack *unsupervised* semantic modelling. Their patent-pending approach generates a compact representation of any text by using advanced statistical language models to identify the significant features of a document.

About the algorithms used, the only information found was in their official website [3], shortly describing how they process the information. In short, an autoencoder neural network encodes the features into a low-dimensionality semantic representation, and then reconstructs an approximation of the original feature vector from the semantic representation. The software highlights keywords that

[3] www.textwise.com

may be under-represented by the semantic representation and encodes these separately as a complementary feature vector. Finally, the complementary feature vector is combined with the semantic representation to produce a Semantic Gist that can be used for document indexing, matching and other applications.

Additionally, their general web API services come free of charge, but are usage limited daily, like OpenCalais (discussed above).

**Maui**

Maui was another tested tool that allows performing keyword extraction. As mentioned earlier in Section 2.2, this tool is based on KEA [FPW+99], enhancing KEA's successful machine learning framework with semantic knowledge retrieved from Wikipedia.

Maui explores Wikipedia capabilities to further compute the semantic relatedness between terms or the likelihood of a phrase being a link in the Wikipedia corpus. Unfortunately, Wikipedia indexing features could not be tested due to some problems related to the import of Wikipedia dump files into MySQL database. Somehow, the 9GB SQL file could not be imported into the database (more than 24 hours later the import was not finished yet and the problem still persisted after increasing memory limits and page table sizes), thus only the simpler version could be tested.

Maui works in two stages: candidate selection and machine learning based filtering. In the original paper, Maui was applied to automatic tagging, but keyword extraction is also possible to be performed. In the candidate selection stage, Maui first determines textual sequences defined by orthographic boundaries and splits these sequences into tokens. Then all n-grams up to a maximum length of 3 words that do not begin or end with a stopword are extracted as candidate tags. In the filtering stage several features are computed for each candidate, which are then input to a machine learning model to obtain the probability that the candidate is indeed a tag.

Tests performed to Maui (without the Wikipedia indexing features), showed results virtually equal to those obtained by KEA and that is why this application was not used in the end.

## 2.4 KEA

As mentioned in Section 2.2, KEA [FPW+99] was one of the main contributions to the field of keyword extraction.

KEA operation can be basically separated into *three phases*:

- First, because not all phrases in a document are equally likely to be keyphrases *a priori*, there is a need to *clean* the input text according to phrase boundaries (punctuation marks, dashes, brackets, and numbers). All the unneeded characters and numbers are deleted. This is how *candidate phrases* are generated;

- Second, and because phrases by themselves are useless (it is their properties or *attributes* that matters), two specific *attributes* are used to discriminate between keyphrases and not-keyphrases: the *TF*IDF* score of a phrase (earlier explained), and the *distance* into the document of the phrase's first appearance (the number of words that precede the first occurrence of the term, divided by the number of words in the document). These were the only two attributes that turned out to be useful in discriminating between keyphrases and non-keyphrases in the author first set of experiments. A *Naive Bayes* model [DP97] is then built from a set of *training documents* for which keyphrases are known (typically because the author provided them) and the model can then be applied to a new document from which keyphrases are to be extracted;

- Finally, KEA computes the TF*IDF scores and distance values for all phrases in the *new document*, using the procedure described above, taking the discretization obtained from the training documents. The naive Bayes model is then applied to each phrase, computing the estimated probability of it being a keyphrase. The result is a list of phrases ranked according to their associated probabilities. Assuming that the user wants to extract *r* keyphrases, KEA then outputs the *r* highest ranked phrases [FPW+99].

Another interesting fact about KEA, is how its performance scales with the amount of training data available. The experiments conducted by the author regarding to this matter showed that no further performance improvement was gained by increasing the number of documents used beyond fifty [FPW+99]. The author also shows that performance can be significantly boosted if KEA is trained on documents that are from the same domain. Such enhancement was

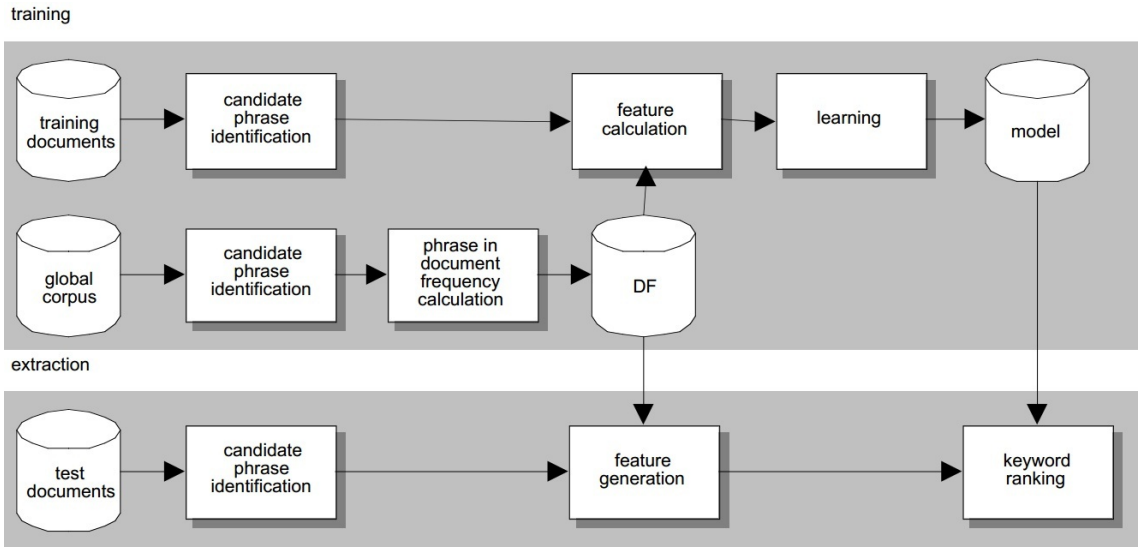


**Figure 2.1:** KEA training and extraction processes ([WPF+99]).

achieved extending the original KEA model, by adding another feature called *keyphrase-frequency*, which is the frequency of a phrase's being keyphrase in all the documents in the corpus [FPW+99, JHL09].

### The Naive Bayes classifier

As previously referred, in supervised classification learning problems, the learner is given a set of *training examples* and the corresponding class labels. From the training examples it outputs a classifier, which then takes an unlabelled example and assigns it to a class. Many classifiers can be viewed as computing a set of discriminant functions of the example, one for each class, and assigning the example to the class whose function is maximum [DH73]. If $E$ is the example, and $f_i(E)$ is the discriminant function corresponding to the $i^{th}$ class, the chosen class $C_k$ is the one for which

$$f_k(E) > f_i(E) \quad \forall\, i \neq k. \tag{2.4}$$

Supposing that an example is a vector of $a$ attributes, as is typically the case in classification applications. Let $v_{jk}$ be the value of attribute $A_j$ in the example, $P(X)$ denote the probability of $X$, and $P(Y_j \mid X)$ denote the conditional probability of $Y$ given $X$. Then one possible set of discriminant functions is

$$f_i(E) = P(C_i) \prod_{j=1}^{a} P(A_j = v_{jk} \mid C_i). \tag{2.5}$$

The classifier obtained by using this set of discriminant functions, and estimating the relevant probabilities from the training set, is then called the *Naive*

*Bayes classifier* (or Naive Bayesian classifier) [DP97]. If the *naive* assumption is made that the attributes are independent given the class, this classifier can easily be shown to be optimal, in the sense of minimizing the misclassification rate, or *zero-one loss*, by a direct application of Bayes' theorem, as follows: if $P(C_i \mid E)$ is the probability that example $E$ is of class $C_i$, zero-one loss is minimized if, and only if, $E$ is assigned to the class $C_k$ for which $P(C_k \mid E)$ is maximum [DH73]. In other words, using $P(C_i \mid E)$ as the discriminant functions $f_i(E)$ is the optimal classification procedure. Accordingly to Bayes' theorem,

$$P(C_i \mid E) = \frac{P(C_i)\, P(E \mid C_i)}{P(E)}. \tag{2.6}$$

Nevertheless, $P(E)$ can be ignored since it is the same for all classes and does not affect the relative values of their probabilities. If the attributes are independent given the class, $P(E \mid C_i)$ can then be decomposed into the product $P(A_1 = v_{1k} \mid C_i) \ldots P(A_a = v_{ak} \mid C_i)$, leading to $P(C_i \mid E) = f_i(E)$ as defined in equation 2.5, Q.E.D. In practice attributes are seldom independent given the class, which is why this assumption is *naive* and which explains the given name [DP97].

## 2.5 KUSCO

KUSCO *(Knowledge Unsupervised Search for instantiating Concepts on lightweight Ontologies, [Alv11])* is a tool that indexes a set of *concepts* with given Points of Interest (POIs), *semantically enriching* them. It was developed by Ana Alves in the ambit of her PhD thesis, at Coimbra's University Informatics Department AmILab[4].

Figure 2.2 depicts KUSCOS' generic *Semantic Enrichment* model. The most important module for the work presented in this dissertation, however, is the Meaning Extraction module, where *term extraction* from event descriptions is performed. All the information presented in this section can be found in [Alv11].

KUSCO's processing flow can then be synthesized as follows:

- A POI's Source is specified as input for the system. In the present architecture, the structure of this source (e.g. a collection of documents, a directory Web site) is mapped to the POI entity on the conceptual data model in order to automatically populate a database of POIs. This intensive extraction, or

[4] Ambient Intelligence Laboratory

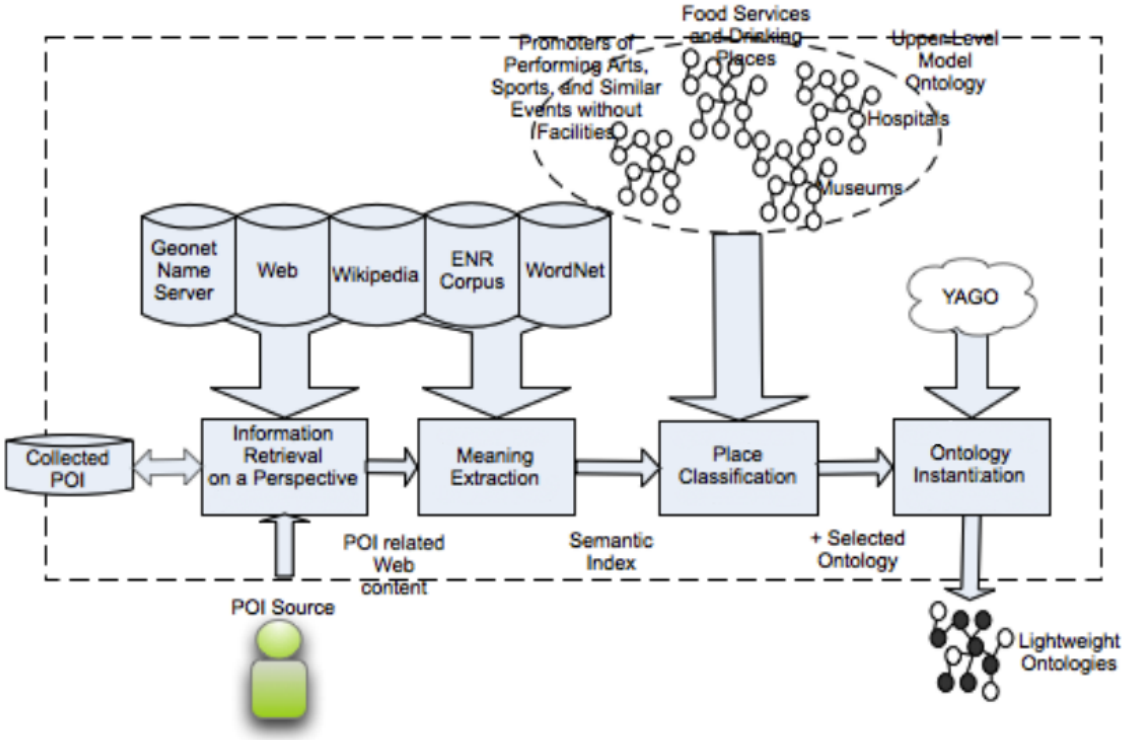


**Figure 2.2:** KUSCO's generic model of Semantic Enrichment ([Alv11]).

POI Mining, may be accomplished by simply invoking API (when available) or by Web scraping.

- Information Retrieval on a Perspective module consists in finding documents about each POI from a given background collection. To achieve that, the Wikipedia and the World Wide Web are explored, applying two different approaches to each of the collections, called perspectives: using the Web through the use of a search API, where one can find a perspective focused specifically in the POI website, while there is an broader perspective that covers documents of several websites; using Wikipedia and its API, one can also find a more specific perspective, Yellow Wiki, and a broader one, Red Wiki. While the former searches for the specific page of a POI, the latter searches for pages related to POIs categories. These categories are normally given from where the POI is extracted.

- *Meaning Extraction* module finds the bag of relevant concepts in a document retrieved from a given source (Web or Wikipedia). In other words, having a set of pages as input, it extracts a ranked list of terms. Instead of a common bag of words, this set contains terms with meaning, since the disambiguation of each term is also performed in this module. Therefore, the intermediate output of the Meaning Extraction module is called a Seman-

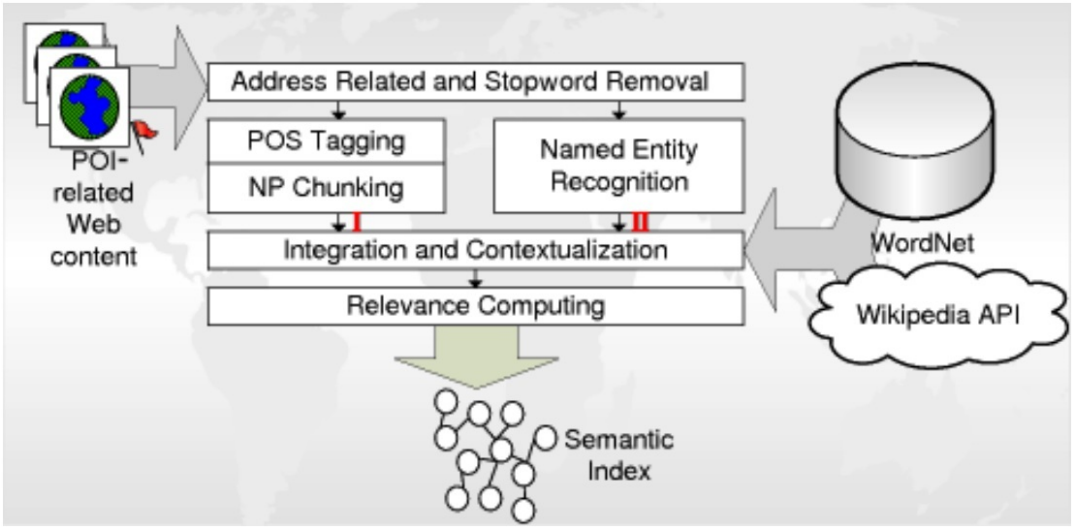


**Figure 2.3:** KUSCO's Meaning Extraction module in detail ([Alv11]).

tic Index, where each concept is weighted using statistical relevance metrics.

For each POI, KUSCO extracts related information from the Web and executes a sequence of Information Extraction and Natural Language Processing steps to automatically extract the relevant related terms. Each term is contextualized in lexical resources (WordNet and Wikipedia) which guide the extraction process by validating common-sense terms and which are also used to infer the meaning of each term. These terms are called concepts only after they are contextualized, and their relevance is computed through an extended version of TF*IDF that considers the semantics of each term.

In Figure 2.3, one can see a more detailed visualization of KUSCO's Meaning Extraction module.

## 2.6 CRF - Conditional Random Fields

Conditional Random Fields (CRF) [LMP01] is a state-of-the-art sequence labeling method and has been successfully applied to many different problems, including natural language processing, named-entity recognition (NER), feature induction for NER, identifying protein names in biology abstracts, segmenting addresses in Web pages, information integration word alignment in machine translation, citation extraction from research papers, word segmentation and many others [SM10]. As mentioned in Section 2.2, and just like KEA, CRF belongs to the supervised field of machine learning.

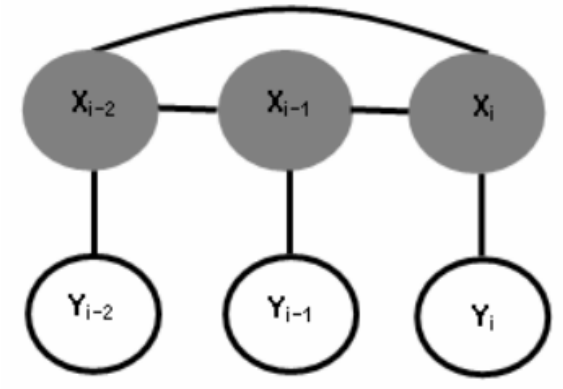


**Figure 2.4:** Graphical Structure of a Chain-structured CRF for Sequences ([Zha08])

One of the first large-scale applications of CRFs was done by Sha and Pereira [SP03], who matched state-of-the-art performance on segmenting noun phrases in text. Since then, linear-chain CRFs have been applied also in natural language processing, including named-entity recognition (NER), feature induction for NER, identifying protein names in biology abstracts, segmenting addresses in Web pages, information integration word alignment in machine translation, citation extraction from research papers, word segmentation and many others [SM10].

Existing methods on keyword extraction can not use most of the features existing in a document. However, CRF can utilize most of those features for efficient keyword extraction more sufficiently and effectively. Experimental results indicate that the CRF model can enhance keyword extraction and it outperforms the other machine learning methods [Zha08].

In short, CRF is an *undirected graphical model* that encodes a *conditional probability distribution* with a given set of features (Table 2.1 portrays the list of features used to train CRF model used in this work).

For the given observation sequential data $X(X_1X_2, \ldots, X_n)$, and their corresponding status labels $Y(Y_1Y_2, \ldots, Y_n)$, a linear chain structure CRF defines the conditional probability as follows:

$$P(Y|X) = \frac{1}{Z_x} \exp \sum_i \sum_j \lambda_j f_j(y_{i-1}, y_i, X, i) \tag{2.7}$$

where $Z_x$ is a normalization and it makes the probability of all state sequences sum equal to 1, $f_j(y_{i-1}, y_i, X, i)$ is a feature function and $\lambda_j$ is a learned weight associated with feature $f_j$.

The main advantage of CRF comes from that it can relax the assumption of con-

| *Nº* | **Features** | **Explanation** | **Normalization method** |
|---|---|---|---|
| 1 | Word | current token | - |
| 2 | PoS | Part-of-Speech of a token in the sentence | {DT, VB, NN, (. . . )} |
| 3 | First Position | whether a token is the first token in each sentence | {0, 1} |
| 4 | CAPITALIZED | whether a token is capitalized | {0, 1} |
| 5 | Initial CAP | whether a token begins with a capital | {0, 1} |
| 6 | Mixed CAPS | whether a token contains both lower and upper cases | {0, 1} |
| 7 | Contains Digits | whether a token contains digits | {0, 1} |
| 8 | All Digits | whether a token is a number | {0, 1} |
| 9 | Hyphenated | whehter a token contains hyphens | {0, 1} |
| 10 | Dollar Sign | whether a token contains the $ sign | {0, 1} |
| 11 | Ends In Dot | whether a token ends with a dot | {0, 1} |
| 12 | Lonely Initial | whether the token is an initial (e.g.: P.) | {0, 1} |
| 13 | Single Char | whether the token is a single char (letter, number, symbol) | {0, 1} |
| 14 | End Punctuation | whether the token is sentence end punctuation | {0, 1} |
| 15 | Apostrophe | whether the token contains an apostrophe ( ' ) | {0, 1} |
| 16 | E-mail | whether the token matches a valid email format | {0, 1} |
| 17 | Line Number | the line number of the current sentence in the document | {1, 2, (. . . ), N} |
| 18 | TF | Term Frequency of a word in the document | See equation 2.1 |
| 19 | IDF | Inverse Document Frequency of a word in the document | See equation 2.2 |
| 20 | TF*IDF | the Term-Frequency * Inverse Document Frequency of a term in the document | See equation 2.3 |
| 21 | Windowed Features | the PoS and Word features of the *first*, *first and second* and *first, second and third* tokens before and after the current token | - |

**Table 2.1:** List of Features used to train CRF model in this work.

ditional independence of the observed data often used in generative approaches, an assumption that might be too restrictive for a considerable number of object classes and it also avoids the label bias problem. The interested reader can find more information about this in respective literature ([LMP01]) once it is not objective for this dissertation exhaustively exploring the algorithm. Additionally, Figure 2.4 shows the graphical structure of a chain-structured CRF.

The original CRF implementation used in this work was developed under the MALLET[5] framework, acronym for MAchine Learning for LanguagE Toolkit, and it was kindly supplied by Filipe Rodrigues, a PhD student currently researching at the AmILab as well.

## 2.7 Conclusion

Various works in the field of *keyword/keyphrase* extraction have been presented. Those rely basically in *three* main types, namely *statistics*, *linguistics* and *machine learning*, however most of them are a mixture of the former, which turns them into *hybrid approaches* after all.

The most prominent methods so far rely on *machine learning* techniques, which can be further divided into *supervised* and *unsupervised*, being the supervised ones the focus for this work.

Semantic analysis tools, like the ones investigated in this chapter, allow for powerful semantic modeling like text categorization and segmentation. These tools are based on different approaches and generally do not merely rely in a single algorithm, combining instead a set of different ones, which is in essence also the paradigm that serve as base for the ensemble learning method.

*KEA system* represents one of the most valuable contributions to the supervised field and it is one of the tools used in the proposed application. From the remaining analyzed/tested systems, KUSCO and CRF were the ones that best fit into the objectives proposed for this dissertation (as discussed earlier) and are used along with KEA in the final ensemble system.

[5]MALLET (http://mallet.cs.umass.edu/) is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text.

# Chapter 3

# Ensemble Learning Methodologies

This chapter is comprised of five sections. Section 3.1 gives some insight about the concept of *Ensemble* applied to the field of Machine Learning. Section 3.2 presents related ensemble works applied to keyword extraction, while Section 3.3 introduces some widely used ensemble learning schemes. Section 3.4 concerns two well known principles of combining classifier's label outputs and finally Section 3.5 summarizes and concludes the information introduced in this chapter.

## 3.1 What is an *Ensemble* ?

When wise people need to make critical decisions, they usually prefer to rely in the opinions of *several* experts rather than on their own opinion or that of a single adviser. Similarly in a boxing fight, it often happens that the combat ends up tied, because none of the fighters knocks out the opponent in due time. In that specific case, the decision of selecting the winner of the combat is then decided through voting, by *multiple* referees. In either case, in order to find a final (and very likely the right) choice, opinions of different experts are being *combined*.

In Machine Learning, like in real life, an obvious approach for making decisions more reliable is to *combine* the output of different classifying models to form the final prediction. Several techniques in the field achieve this by learning an *ensemble of models* and using them in combination [WF05].

*Ensembles*, also known as *committees*, are justly receiving increasing attention and accolade, generating a wealth of research [KR13]. Theoretical and empirical studies have demonstrated that ensembles frequently *increase predictive performance* over a single base classifier and can be applied to different kinds of problems, as numeric prediction (the output is a number between 0 and 1) problems and classification (like binary classification: *keyword* and *not-keyword*) tasks as well

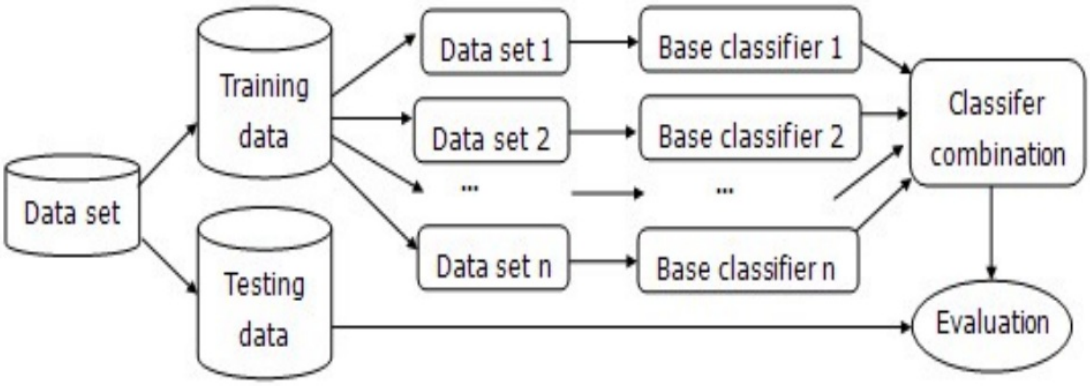


**Figure 3.1:** Logical view of ensemble learning method ([Zha09]).

[Mel05, KR13].

Different methods for combining models exist, like *bagging*([Bre96]), *boosting*([FS96]) and *stacking*([Wol92]), being the first two the most widely used[WF05, Mel05, Sew11]. These combined models share sometimes however the disadvantage of being difficult to analyse, once they can comprise dozens or even hundreds of individual classifiers. Figure 3.1 illustrates the ensemble learning method.

Although ensembles perform well, it is not trivial to understand in an intuitive way which factors are contributing to the improved decisions of such approaches. That fact has been motivating researchers struggling in order to understand why [WF05].

Additionally, another typical concern around ensembles relates to the output combination rule to use, i.e., how should the results coming from the individual applications be combined. *Majority voting* and it's *weighted* version are the most widespread choices when the individual classifiers give label outputs [Kun04], and are detailed ahead in this chapter.

## 3.2 Related Work

In 2004, Hulth [Hul04a] presented an algorithm for automatic keyword extraction combining statistical and linguistic methods, showing that the number of incorrect assigned keywords could be highly reduced. That reduction was achieved by combining then the predictions of several classifiers. Moreover, the classifiers are trained on different representations of the data, where the difference lies in the definition on what constitutes a candidate term in a written document.

Using an ensemble of Neural Networks (formerly Bagging and Boosting en-

semble schemes, described in Section 3.3), Wang et.al [WPHZ06] show an approach where keyphrase extraction is viewed as a crisp binary classification task, training the neural network ensemble to classify whether a phrase is keyphrase or not. To determine whether a phrase is a keyphrase, the following features (or attributes) of a phrase in a given document are adopted: its *term frequency*, whether they *appear in the title, abstract or headings* (subheadings), and its *frequency* appearing in the paragraphs of the given document, i.e., the distribution of a phrase in a given document. However, due to the binary classification model adopted, this model is not suitable when the number of phrases classified by the classifier as positive is less than the desired number of keyphrases.

Later in 2009, Zhang [Zha09] combined several statistical machine learning models to extract keywords from Chinese documents. His method selects keywords through voting, from the multiples models created, and the experimental results show that the proposed ensemble learning method outperforms other methods, according to $F_1$ measurement. Moreover, the extraction model using weighted votes outperforms the model that does not use weighted voting.

## 3.3 Approaches

### 3.3.1 Bagging

*Bagging*, acronym formed from *Bootstrap Aggregation*, was first introduced by Breiman [Bre96]. The idea behind this approach is building multiple classifiers from the training set by using the bootstrap sampling, i.e., sampling with replacement: instead of using a fresh and independent training dataset each time (which in practice is obviously not feasible), the algorithm just alters the original training data, $Z = \{z_1, \ldots, z_N\}$, deleting some instances and replicating others, creating a pseudo-new training set of the same size, $L$, each time. The hope is that the base classifiers, generated from the different bootstrapped training sets, disagree often enough that the ensemble performs better than the base classifier. The outputs of these classifiers are then combined by *averaging or voting* to create the final prediction.

To make use of the variations in the training set, bagging must be used with *unstable* (i.e. a small change in the training set can cause a significant change in the classifier output) non-linear classifiers [Oza00, WF05, Sew11, Kun04]. Otherwise, the resultant ensemble will be a collection of almost identical classifiers, therefore unlikely to improve on a single classifier's performance, once they will

BAGGING

**Training phase**

1. Initialize the parameters
   - $\mathcal{D} = \emptyset$, the ensemble.
   - $L$, the number of classifiers to train.
2. For $k = 1, \ldots, L$
   - Take a bootstrap sample $S_k$ from $\mathbf{Z}$.
   - Build a classifier $D_k$ using $S_k$ as the training set.
   - Add the classifier to the current ensemble, $\mathcal{D} = \mathcal{D} \cup D_k$.
3. Return $\mathcal{D}$.

**Classification phase**

4. Run $D_1, \ldots, D_L$ on the input $\mathbf{x}$.
5. The class with the maximum number of votes is chosen as the label for $\mathbf{x}$.

**Figure 3.2:** The bagging algorithm ([Kun04]).

be specialized in the same part of $\boldsymbol{Z}$. Examples of unstable classifiers are neural networks and decision trees, while naive Bayes stands as an example of a stable classifier. Figure 3.2 illustrates the training and operation of bagging.

### 3.3.2 Boosting

*Boosting* was originally proposed by Schapire and Freund [FS96]. Like bagging, it is used to combine unstable models of the same type, however boosting does that iteratively: whereas in bagging individual models are built in separate (i.e., equal weight is given to each one), in boosting each new model is influenced by the performance of those built previously (i.e., models that label correctly more examples are given more weight, being this weights directly proportional to each model's specific reliability). The hope is that subsequent base models correct the mistakes of the previous, encouraging new models to become experts for instances handled incorrectly by earlier ones [Oza00, WF05, Sew11]. Figure 3.3 shows the flowchart of the boosting algorithm.

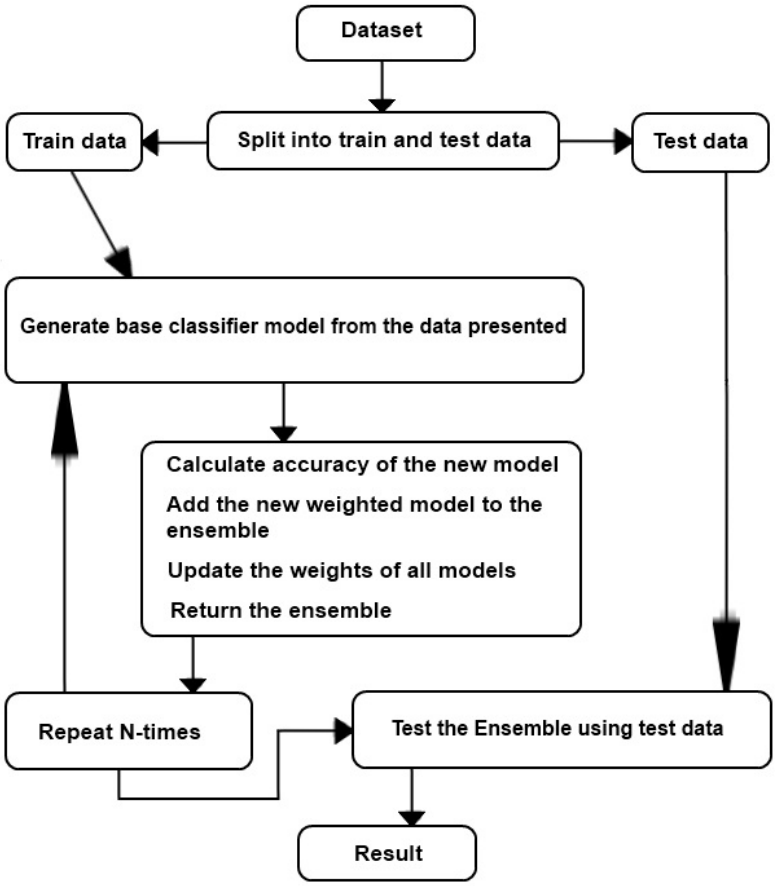


**Figure 3.3:** Flowchart of the boosting algorithm (adapted from [GW07]).

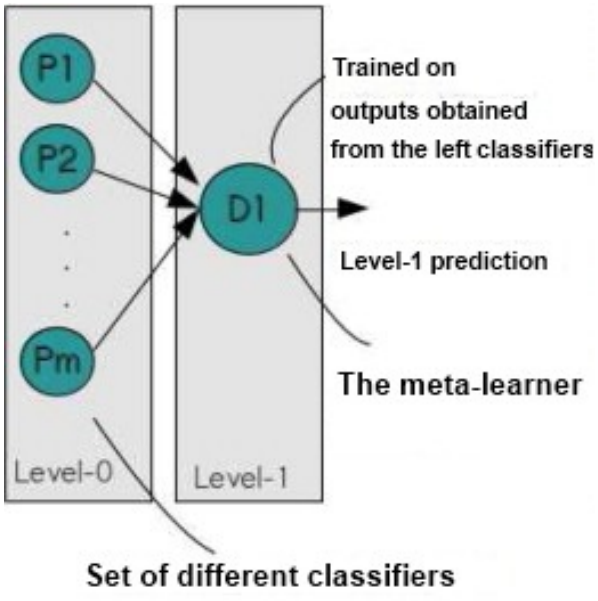


**Figure 3.4:** Stacking algorithm.

### 3.3.3 Stacking

*Stacked generalization*, or *stacking*, was first presented by Wolpert [Wol92] and is a different way of combining multiple models. Unlike bagging and boosting, stacking is not normally used to combine models of the same type. Instead, it is applied to models built by different learner algorithms.

Stacking introduces the concept of *meta-learner*, which replaces the voting procedure of the former algorithms. By using the meta-learner (that is basically another learning algorithm), stacking tries to learn which classifiers are reliable and to discover how best to combine the output of the base learners. Figure 3.4 tries to illustrate the mechanism described ahead.

In short, stacking basically splits the training set into *two* disjoint sets, training several base learners, or *level-0 models*, on the *first part*. After that it tests the base learners on the *second part* and trains a higher level learner (using the predictions from the testing phase as the inputs and the correct responses as the outputs), the *level-1 model* [Sew11]. In short, an instance is first fed into the level-0 models and each one guesses a class value. These guesses are fed into the level-1 model, which combines them into the final prediction [WF05].

Although developed some years ago, it is less used than bagging and boosting, partly because it is difficult to analyze theoretically and to implement [WF05].

### 3.3.4 Bayesian Model Averaging

Ensemble learning can be further seen as a tractable approximation to full Bayesian learning. In *full Bayesian learning*, the final learned model is a mixture of a very large set of models (i.e., combinations of predictors), normally all possible models in a given family, when making inferences about quantities of interest. If one wants to predict a certain quantity $\delta$, and has a set of models $m_k$ and a training set $T$, then the final learned model is:

$$P(\delta \mid T) = \sum_i P(\delta \mid m_k)\, P(m_k \mid T) = \sum_i P(\delta \mid m_k)\, \frac{P(T \mid m_k)\, P(m_k)}{P(T)} \qquad (3.1)$$

Full Bayesian learning combines the explanatory power of all the models $(P(\delta \mid m_k))$ weighted by the posterior probability of the models given the training set $(P(m_k \mid T))$. Ensembles can then be seen as an *approximation* to full Bayesian learning by using a mixture of a small set of the models having the highest posterior probabilities $(P(m_k \mid T))$ or highest likelihoods $(P(T \mid m_k))$ [Oza00].

In [RMV94], the authors show that full Bayesian averaging provides optimal predictive ability. However, full Bayesian learning is intractable because it uses a very large (possibly infinite) set of models.

The tractable approximation to full Bayesian Model is called *Bayesian Model Averaging* (BMA) [RHM97]. This solution leads with the problem of full Bayesian Model, earlier discussed, and it was first introduced by Leamer [Lea78]. However, because the implementation of the method proposed was difficult and contained some flaws, Raftery et al [RHM97] proposed two approaches to solve the addressed problems in Leamer's work:

- *Occam's window* method;
- *Markov Chain Monte Carlo model Composition* ($\boldsymbol{MC^3}$) method.

The former involves averaging over a reduced set of models only. In the latter, the idea is to directly approximate the complete solution by applying the $\boldsymbol{MC^3}$ approach of Madigan and York ([YL95]).

Additionally in [RHM97], it is also shown that both of these *model averaging* approaches provide better predictive ability than any *single model* approach, which is a pretty good premise for developing committee applications over single model ones.

BMA is the generally accepted method for applying Bayesian learning theory to the task of model combination. Although the result of BMA is a combination of models, this combination is actually just integrating out system's uncertainty as to *which* model is correct, assuming that *one and only one* of the models is indeed the right one in each specific case. Thus, BMA is actually a model selection procedure that deals with uncertainty about its selection using a combination [MCSM11].

Despite that BMA is theoretically the optimal method for combining learned models, it has seen very little use in machine learning [Dom00]. Additionally, the effectiveness of BMA seems to depend from the models used [Ali95]. In [Ali95], Bayesian model averaging did poorly in domains in which the posterior probability of one model dominated those of others. This is the same as saying that BMA is not useful if one model significantly outperforms the other.

Domingos [Dom00] studied its application to combining rule sets, comparing it with methods as bagging and partitioning and, surprisingly or not, BMA's

error rates were consistently higher than the other methods'. In the same work the author concluded that BMA does not obviate the over-fitting problem in classification, contradicting previous beliefs that it solves (or avoids) it, and may in fact aggravate the problem.

Recently, Monteith et al [MCSM11] proposed that, in order to more effectively access the benefits inherent to ensembles, Bayesian strategies should therefore be directed more towards *model combination* rather than the *model selection* implicit in BMA. Their work provides empirical verification for that hypothesis using several different Bayesian model combination (BMC) approaches, tested on a wide variety of classification problems. Empirical results shown on that paper reveal that even the most simplistic BMC strategy outperforms the traditional *ad hoc* techniques of bagging and boosting, as well as outperforming BMA over a wide variety of cases, suggesting that the power of ensembles does not come from their ability to account for model uncertainty, but instead comes from the changes in representational and preferential bias inherent in the process of combining several different models.

As cited above BMA is not quite understood nowadays and, even currently, researchers struggle to understand the concepts behind Bayesian theories. Despite being not the focus of this dissertation, the interested reader can further explore the concepts in dedicated literature ([Lea78], [RMV94], [YL95], [RHM97], [Dom00], [MCSM11]).

## 3.4 Combining label outputs

### 3.4.1 Majority Voting

Figure 3.5 portrays different ways of reaching a consensus about which results should be considered being the relevant ones. If we assume that black, gray, and white correspond to class labels, and the decision makers are the individual classifiers in the ensemble, the final label will be *black*, for all three patterns.

To better understand how the voting procedure takes place, assume that the label outputs of the classifiers are given as $c$-dimensional binary vectors, $[d_{i,1}, \ldots, d_{i,c}]^T \in \{0,1\}^c, i = 1, \ldots, L$, where $d_{i,1} = 1$ if $D_i$ labels $x$ in $\omega_j$, and $0$


**Figure 3.5:** Consensus patterns in a group of 10 decision makers: unanimity, simple majority, and plurality. In all three cases the final decision of the group is *black*. ([Kun04]).

otherwise. The *plurality vote* will result in an ensemble decision for class $\omega_k$ if

$$\sum_{i=1}^{L} d_{i,k} = \max_{j=1}^{c} \sum_{i=1}^{L} d_{i,j} \tag{3.2}$$

The plurality vote of Equation 3.2, is called in a wide sense *the majority vote*, and is the most often used rule from the majority vote group. Various studies are devoted to the majority vote for classifier combination [Kun04].

Ties are resolved arbitrarily, or following some pre-set rule. In fact, to solve this question Xu et al [XKS92] suggest a *thresholded* plurality vote. They augment the set of class labels $\Omega$ with one more class, $\omega_{c+1}$, for all objects for which the ensemble either fails to determine a class label with a sufficient confidence or produces a tie. Thus the decision is

$$\begin{cases} \omega_k, & \text{if } \sum_{i=1}^{L} d_{i,k} \geq \alpha.L, \\ \omega_{c+1}, & \text{otherwise,} \end{cases} \tag{3.3}$$

where $0 < \alpha < 1$. For the simple majority, we can pick $\alpha$ to be $\frac{1}{2} + \epsilon$, where $0 < \epsilon < 1/L$. When $\alpha = 1$, Equation 3.3 becomes the unanimity vote rule: a decision is made for some class label if all decision makers agree on that label; otherwise the ensemble refuses to decide and assigns label $\omega_{c+1}$ to $x$.

### 3.4.2 Weighted Majority Voting

If the classifiers in the ensemble are not of identical accuracy, then it is reasonable to attempt to give the more competent classifiers more power in making the final

decision.

The label outputs can be represented as degrees of support for the classes in the following way:

$$d_{i,j} = \begin{cases} \mathbf{1}, & \text{if } D_i \text{ labels } x \text{ in } \omega_j, \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{3.4}$$

The discriminant function for class $\omega_j$ obtained through weighted voting is

$$g_j(x) = \sum_{i=1}^{L} b_i d_{i,j}, \tag{3.5}$$

where $b_i$ is a coefficient for classifier $D_i$, which corresponds to the weight of that classifier. Thus the value of the discriminant Function 3.5 will be the sum of the coefficients for these members of the ensemble whose output for $x$ is $\omega_j$ [Kun04].

## 3.5 Conclusion

Ensemble learning lies between *traditional learning* with single models and *full Bayesian learning* since it uses an intermediate number of models.

Bagging and boosting are the most widely ensemble schemes used but they derive the individual models differently. The former gives equal weight to each model, whereas in the latter weighting is used to give more influence to the most successful ones. This actually portrays what happens in the real world, just as an executive might place different values on the advice of different experts, depending on how experienced they are. Nevertheless, these are not suitable when using different or stable classifiers, as it happens in this work.

Ensembles normally achieve better results than individual classifiers. One way of explaining its superior performance, is that the former have greater expressive power than the latter.

Two of the simplest and most widely used techniques applied to combine the results obtained by the classifiers that compose the ensemble are those of *simple majority* and *weighted majority voting*, which normally use a plurality consensus rather than a simple majority one.

# Chapter 4

# Ensemble Learning for AKE from Event Descriptions

This chapter is composed by four sections. In Section 4.1 the architecture of the proposed system is depicted and a description of its main modules is given. Section 4.2 and Section 4.3 describe and detail the main stages identified in the previous section: the preprocessing phase (stage one) and how the classifying process and label output combination are done after candidate words being selected (stage two and three). Relevant aspects and decisions made during the assemblage of the ensemble application are addressed in Section 4.4.

## 4.1 Application's Architecture

Following what has been introduced over the last few chapters, it soon became evident that the spectrum of existing keyword extraction algorithms is already somehow substantial. The raising interest in the field in question, conduced to the development of different solutions, each one pretending to be more effective than the former. Usually, each of these applications rely on a classifier and most of the improvement process passes by enhancing the success rate of that classifier, however, the main idea for this dissertation is slightly different, once it concerns *ensembling* a set of already existing classifiers to solve the problem, instead of adding some new features to an existing one, which is what is normally done. The premise in doing so, is that a set of applications/algorithms working together is better than all when working alone, as once someone pointed out.

The architecture of the proposed system is represented in Figure 4.1. It de-

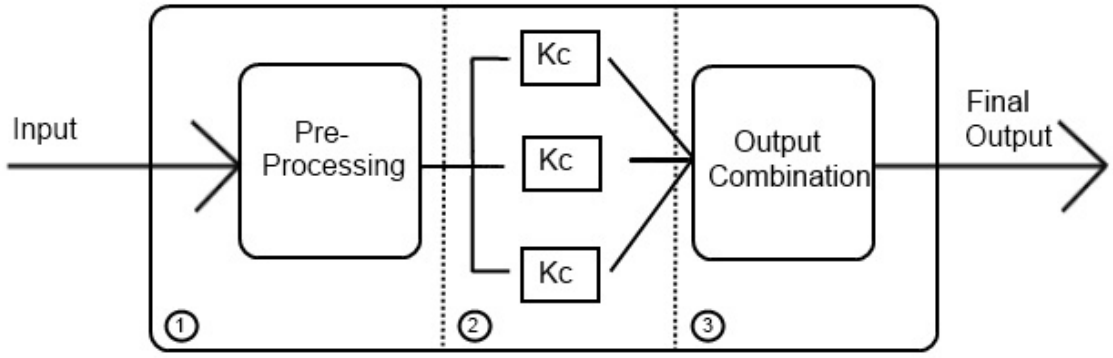


**Figure 4.1:** System's architecture. (*Input*): Any suitable textual source (e.g.: textual descriptions from events); (*1*): In this stage, the input is preprocessed in order to be ready for classification; (*2*): The *Keyword classifiers (Kc)* from the currently chosen existing systems, that will classify the preprocessed input; (*3*): The last stage, *combining the results* from the individual classifiers; (*Final Output*): a set of keywords extracted from the input.

picts the different components of the application and it also reveals the system's processing flow, since the moment an input text is passed to the application (i.e., after being retrieved from the Web and stored as a text file), until the moment that it produces the desired output. The main stages identified there are then listed above:

- The first stage consists in *preprocessing* the input text (see Section 1.2). KEA and KUSCO preprocessing is already implemented within the applications, however, for the CRF system, further preprocessing was needed, as depicted in the next section;

- The second stage is composed by a set of trained classifiers or models generated by the individual applications listed before, that are now able to classify new unseen instances of documents and find the keywords among them. These individual predictions are then passed to the third and last stage;

- In the third stage, the individual predictions of the classifiers are interpreted and combined in order to make a final decision. This decision will tell which candidate terms are in fact keywords and make part of the final set of keywords, that probably best describe that document.

## 4.2 Preprocessing

As already referred before, preprocessing tasks are usually a prerequisite to text classification. The objective of the preprocessing stage is cleaning the input text as much as possible by eliminating unnecessary words and characters (e.g. KUSCO, KEA) or, as it happens with other methods (e.g. CRF), just structuring the text in some manner for further classification. Thus, this section details the preprocessing phase of each application, which corresponds to the *stage 1* identified in Figure 4.1.

### 4.2.1 KEA

KEA preprocessing works as follows:

- The input text is split up according to phrase boundaries (punctuation marks, dashes, brackets, and numbers) and non-alphanumeric characters (apart from internal periods) and numbers are deleted. KEA takes then all the sub-sequences of these initial phrases (up to length three by default, but it can be changed), as *candidate phrases*;
- In the next step, it eliminates the phrases that begin or end with a *stop-word* or phrases that are a *proper noun*. The stop-word list used by KEA contains 425 words in nine syntactic classes (conjunctions, articles, particles, prepositions, pro-nouns, anomalous verbs, adjectives, and adverbs);
- Finally, all words are case-folded and stemmed using a stemmer (originally the iterated Lovins' Stemmer was used [Lov68], but any suitable stemming algorithm can be used). Stemmed phrases that occur only once in the document are also removed (by default, also configurable) for large texts, while for small documents it is not recommended to do so [WPF+99].

### 4.2.2 KUSCO

Like KEA, KUSCO needs to separate candidate terms from irrelevant ones a priori, so it can extract keywords more efficiently. KUSCO's preprocessing phase is depicted below [Alv11]:

- Starting at the Meaning Extraction module (described in Section 2.5 and portrayed in Figure 2.3) with Noun Phrase (NP) chunking and Named Entity Recognition (NER), using available Natural Language Processing (NLP)

tools, KUSCO brakes up the received texts into paragraphs, paragraphs into sentences, and sentences into words;

- After that, a sentence analyser identifies the boundaries of sentences in a document and a tokeniser decomposes each sentence into tokens. Tokens are then obtained by splitting a sentence also according to predefined phrase boundaries. Words in a sentence are then labelled as a noun, verb, adjective, etc., according to its PoS;

- Furthermore, a Noun Phrase chunker is then applied in order to identify every group of words with a head noun which functions together just as a single term and, at the same time, the original text is also processed by a Named Entity recognizer to identify proper names in the text, labelling them accordingly (person, organization, location);

- Finally, and similarly to what happens with KEA, a widely used stop-word list is consulted to filter future concept candidates. Besides this, some heuristics are used to determine the validity of each term before the next phase, that of information integration, begins.

### 4.2.3 CRF

CRF required extra/different preprocessing operations than those needed by the other two applications and it revealed to be a very important step, once a slight modification in the text being passed to the system resulted in major differences in the results obtained.

In order to effectively extract keywords using CRF method, the text needed to be well structured and a set of features correctly tagged, enumerated below, contrary to what was needed by the other two applications.

Two major aspects were then considered, during this phase:

***Structural issues:***

- separating each phrase of the text, one per line;

- keeping the punctuations present in each phrase (except quote marks, which actually produced better results when removed);

- keeping the stop-words present in each phrase;

- each token of each phrase separated by a space.

***Feature and keyword's automatic tagging:***

- tagging the true keywords of each document within brackets[1] (E.g.: [ correctly tagged keyword ] );
- identifying a set of features present in the text, namely those described in Table 2.1.

## 4.3 Keyword Classifiers and Output Combination

When a new document is to be classified arriving from the preprocessing phase, it is presented to the respective application classifier ($\boldsymbol{K_c}$). Based on previous training, that ended up generating a model for each application, the individual systems are now able to make predictions for the new examples given. Ideally these models should complement one another, each being specialized in a part of the domain where the others do not perform so well (just as human executives seek advisers whose skills and experience complement, rather than those whose skills are based in the same domain). Thus, models that are capable of doing predictions as uncorrelated as possible (i.e. they differ much on the examples they can label as positive or negative) are preferable, once they will be further restricting the result space and the ensemble will then probably perform better than the base models individually. However, this optimal scenario cannot be entirely controlled by hand or otherwise we would have the problem here approached solved already.

Figure 4.2 illustrates what have been said and portrays the expected behaviour of the several classifiers working together. $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ represent a set of three classifiers, which are identified by $\boldsymbol{K_c}$ in Figure 4.1. Note yet that this example depicts a classification task (examples are classified as positive (+) or negative (-) only) rather than a regression task (in which a real value is to be predicted, normally representing the likelihood of a given word being a keyword).

The models represented by $\boldsymbol{K_c}$ could be any suitable machine learning classifier, but in this specific case one corresponds to the Naive Bayes classifier used by KEA (see Chapter 2.4), other to KUSCO's Meaning Extractor module (see Chapter

[1]For the other applications, the keywords were just identified in separated files (*.key*), as mentioned previously.

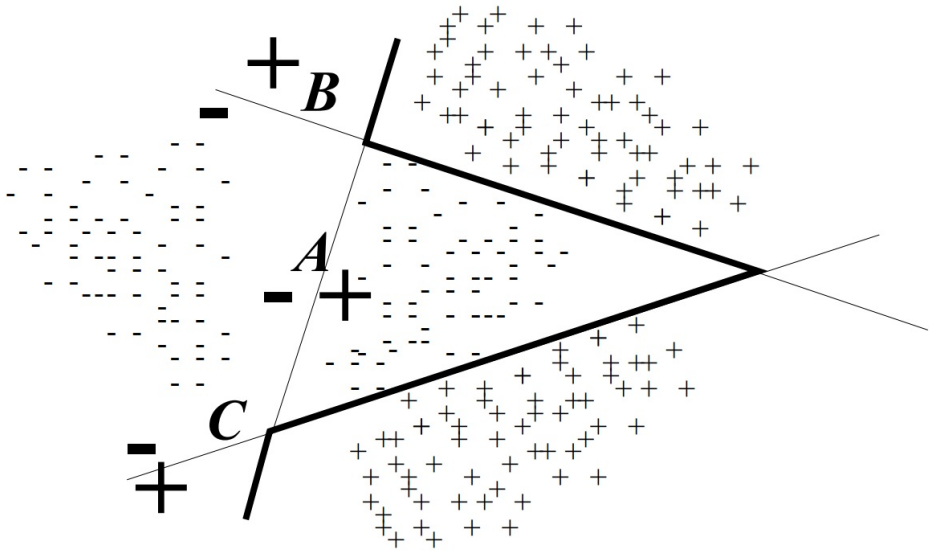


**Figure 4.2:** An ensemble of linear classifiers. Each line ***A***, ***B***, and ***C*** is a linear classifier. The boldface line is the ensemble that classifies new examples by returning the majority vote of the classifiers ([Oza00]).

2.5), which consists on pure statistical and linguistics methods, and the last one concerning a CRF model (see Chapter 2.6).

After the document has been labelled by all the classifiers, the results must be gathered and combined, which is then the final stage of the ensemble application here presented. For that, both voting scheme procedures depicted in Section 3.4, namely those of simple and weighted majority voting, were used.

## 4.4 Ensemble Learning Assemblage

The first step forward creating the application here proposed was selecting the set of different tools/algorithms combine thereafter. KEA because it is one of the most widely known keyword extraction systems and because it is an open-source tool available on-line for download. Additionally, it is considered one of the main contributions to the area and many subsequent works used it as a baseline for comparisons. KUSCO, because it was developed in the AmILab (the lab where this work is being developed as well) and in addition of showing decent results, all the documentation, code and support are easy accessible. The last one, CRF, because it is the state-of-the-art sequence labelling method and is showing the best current results in the field.

Second, finding a way for combining the models produced by the applications chosen before. As mentioned in Section 3.2, the most widely known (and used) ensemble schemes are those of bagging and boosting, that apply on models of the same type, but that is not the case in here. Once we are already using different models for predicting, using this techniques was not quite necessary. In fact, once Naive Bayes is a stable algorithm and KUSCO does not even needs training for extracting the important terms from a document, the models produced by applying those techniques would not differ much from each other from training set to another, as both of those methods exploit the *instability* (slight changes to the training data result on quite different models being produced) inherent of learning algorithms to produce different models. However, if we were using an ensemble purely composed by CRF models, it would be worth considering using such techniques once the empirical tests carried out indicated that CRF model is dependent of the given training data.

Stacking and Bayesian model averaging (also introduced in Section 3.2) were the other ensemble schemes investigated. While Bayesian Model Averaging seemed to be the most promising approach to be used, which is an approximation to the optimal result, this technique only works well theoretically and empirical studies performed by other authors showed that it tends to produce worse results than bagging and boosting, as described in that same chapter. Stacking, in the other hand, showed to be a technique that is hard to implement. In addition to the fact that very little work in the field use this technique, it also needs to account for two crucial issues: the type of generalizer that is suitable to derive the higher-level model, and the kind of attributes that should be used as its input [TW99], so that is why it was not tested and used in this work.

Third, as also referred in the previous section, the last step was choosing some technique to combine the output of the individual classifiers. The ones presented here are the most widely used ones, namely those of simple and weighted majority voting (see Section 3.4), and that using weights achieved better results than that not using, once the degree of confidence inherent to each model was quite different.

## 4.5 Conclusion

This chapter presented the architecture of the application, as well as a description of its main modules, and resumed and justified why some options were taken, in

detriment of others.

Summarizing what have been said, because the classifier models used here are already different from each other, the approach taken is simply using a single model (per application) generated through training (once they use supervised machine learning) and combining their outputs using a majority voting scheme (simple and weighted versions), filtering the keywords in which those models agreed the most and thus amalgamating their individual outputs into a single final prediction.

# Chapter 5

# Experimental Setup

This chapter introduces the reader to the setup employed in the experiments and it is composed by three sections. Section 5.1 reveals some recursive difficulties about finding suitable datasets and describes those actually used in this project. Section 5.2 gives the metrics utilized to evaluate the applications' performance and finally Section 5.3 depicts the adopted methodology.

## 5.1 Datasets

### 5.1.1 Finding suitable datasets

Reproducibility is and has always been the main problem of most of the datasets considered in the state of the art. It is really hard to compare new algorithms and methodologies with previous works in the field mainly because results may vary from dataset to dataset drastically, even when papers from the same storage and nearly the same domain are used. A simple change in the dataset used can result on an huge difference in the obtained results, depending on the machine learning method [KAM08].

Peter Tourney [Tur99a, Tur99b] was one of the pioneers in the field of automatic keyword extraction. Additionally to proposing a detailed investigation on decision trees based algorithms, he also published several links to freely available datasets [Tur99a], which are no longer available on-line (his work was more than one decade ago and those links do not work any more).

Although KEA is freely available for download through KEA website [1], it does not bring any standard datasets along with the download package and the

[1] http://www.nzdl.org/Kea/

datasets reported in [WPF+99] were unfindable as well. KEA authors mention that they obtained Tourney dataset directly from Tourney himself, however Nguen et al [NyK07] pointed out to the impossibility to find that or any proper datasets and used its own dataset [KAM08].

Unable to find the original dataset used to test KEA, which one intended to use during the experimentation phase, the solution passed by using other two datasets, each one composed by documents varying in length (the size in characters or words), once it was intended to test how well the algorithms performs given not only document's source but also its size. As the final application is to be used on textual descriptions (as mentioned before), that are commonly much shorter than average documents, this test needed to be done mandatorily.

### 5.1.2 Dataset descriptions

A description of the datasets used to evaluate the applications is given above. Notwithstanding, the general configurations of the tests made like the number of documents used for training and testing, will only be provided in Section 6, since they will vary according to the experiment in question.

#### Krapivin's Dataset

The first dataset used, from now on designated by *Krap's dataset*, consists of 2304 *full papers* from *Computer Science* domain, which were published by ACM in the period from 2003 to 2005. All these papers are written in English and stored in UTF-8 text encoding. Each document has clearly indicated its title, abstract, body and references.

This dataset is composed of two types of files: *[id].txt*, which refers to the text itself and *[id].key*, which refers to the keyword's list of the document identified by *[id]*, one per line [KAM08].

#### Hulth's Dataset

The second dataset, from now on addressed as *Hulth's dataset*, consists of 2000 scientific journal *paper abstracts* with their corresponding title, from the Inspec[2] database, and are also written in English. Hulth's documents were obtained from

[2]http://www.iee.org/publish/inspec/

*Computers and Control* and *Information Technology* and have been widely used in previous researches (e.g. [MT04, Hul03]), have being published in the period from 1998 to 2002 [Hul04a].

There are three types of files on this dataset: *[id].abstr*, containing the title and the abstract; *[id].contr*, containing the controlled manually assigned keywords (keywords restricted to the Inspec thesaurus), separated with semicolon; *[id].uncontr*, containing the uncontrolled manually assigned keywords (keywords that can actually be any suitable term), also separated with semicolon.

Before being possible testing the applications using this dataset, two slight modifications needed to be performed. In first place, file extensions were modified: *[id].abstr* files were converted to *[id].txt* files and the *[id].abstr* files were also modified and converted to *[id].key* files. In second place, because keywords were separated with a *semicolon* instead of being *one per line* as expected, it was necessary to change that as well.

Yet about this dataset, an inherent problem taken in account was the percentage of true keywords (i.e, the author preassigned keywords) that were in fact present in the abstracts. Since the proposed task for this work is *extraction*, i.e, terms must exist in the text source (as mentioned in Chapter 1), some refinements were preformed during the preprocessing phase to guarantee that 100% of the terms could be found in the text after all. The problem had already been pointed out previously by Hulth in [Hul04a], as she stated that only 76.2% of the terms for the uncontrolled and 18.1% for the controlled terms were really present in the abstracts (note that this happened because the assigners had access to the full articles and some of the keywords only occurred there).

**Events' Dataset**

To validate the results obtained from the previous scientific datasets, another two groups of documents were used: one composed by 420 descriptions about events in general, like theatre plays and music concerts, retrieved from webpages, while the other comprising 112 personalities' descriptions extracted from Wikipedia. Independently of the datasets used, *[id]* identifies and distinguishes one document from another.

| | **Relevant** | **Non-relevant** |
|---|---|---|
| **Retrieved** | *true positives (tp)* (*items correctly extracted*) | *false positives (fp)* (*items extracted that are not correct*) |
| **Not retrieved** | *false negatives (fn)* (*items not extracted that should have been*) | *true negatives (tn)* (*items correctly not extracted*) |

**Table 5.1:** Classification of each type of result from IR systems.

## 5.2 Evaluation Metrics

In the evaluations performed during this work, *three* already traditional metrics were used to evaluate applications' performance: Precision, Recall and F-measure, the latter calculated by combining the former. Additionally, in Table 5.2 a couple of examples illustrate the theoretical definitions given below.

With reference to the presented contingency table (Table 5.1) that classifies each type of result, *P* and *R* are formally denoted by equations 5.1 and 5.2 respectively.

*Precision* is denoted as the number of true positives (i.e. the number of items correctly labelled as belonging to the positive class) divided by the total number of elements extracted by the application, while *Recall* is denoted as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were *not labelled* as belonging to the positive class but should have been) [MRS08, Alv11]. *Precision* can thus be seen as a measure of exactness or fidelity, whereas *Recall* is a measure of completeness.

In other words, precision *(P)* represents the proportion of automatic selected keywords that are also manually assigned keywords, while recall *(R)* is the proportion of manually assigned keywords found by the automatic method.

Following what has been told, precision and recall can be defined namely by equations 5.3 and 5.4, which are equivalent to equations 5.1 and 5.2 respectively.

$$\boldsymbol{Precision = \frac{tp}{(tp + fp)}} \tag{5.1}$$

$$\boldsymbol{Recall = \frac{tp}{(tp + fn)}} \tag{5.2}$$

$$\textbf{Precision} = \frac{\#\ of\ manual\ keywords\ automatically\ selected}{Total\ \#\ of\ automatically\ selected\ terms} \tag{5.3}$$

$$\textbf{Recall} = \frac{\#\ of\ manual\ keywords\ automatically\ selected}{\#\ of\ manual\ keywords\ in\ document} \tag{5.4}$$

Precision and recall can yet be combined in the *F-measure*, another standard information retrieval metric. F-measure (also known as $\boldsymbol{F_1}$ score) is the harmonic mean between precision and recall, and is denoted by equation 5.5. In the great majority of the experiments, there is no particular reason to favour precision or recall (despite being one of the arguments given by the authors in [WPF+99] for using not these metrics in their work), so most researchers use equal weight of precision and recall to compute F-measure [Alv11].

$$\boldsymbol{F - measure = 2 \times \frac{Recall \times Precision}{Recall + Precision}} \tag{5.5}$$

An alternative metric used to evaluate this kind of solutions is known as *Accuracy*, which corresponds to the fraction of IR system classifications that are correct. Accuracy can be denoted by equation 5.6, representing then the proportion of true results (both true positives and true negatives) in the population [Alv11].

$$\boldsymbol{Accuracy = \frac{tp + tn}{(tp + fp + tn + fn)}} \tag{5.6}$$

Another important aspect about the evaluation metrics, concerns the calculation of these measures, which can be achieved using two *averaging operations*, called *macro-averaging* and *micro-averaging* [Yan99].

There is an important distinction between macro-averaging and micro-averaging: while *micro-average* performance scores gives *equal weight* to every *document*, and is therefore considered a *per-document* average (more precisely, an average over all the document/category pairs), *macro-average* performance scores give *equal weight* to every *category*, regardless of its frequency, and is therefore a *per-category* average

| Example 1 | | # |
|---|---|---|
| **Document true keywords** | US dollar, exchange rate | 2 |
| **Retrieved** | US dollar, market share, profit | 3 |
| *tp* | US dollar | 1 |
| *fp* | Market share, profit | 2 |
| **Not retrieved** | Exchange rate | 1 |
| *tn* | - | 0 |
| *fn* | Exchange rate | 1 |
| **Precision** | 1 / (1 + 2) = 1 / 3 | - |
| **Recall** | 1 / (1 + 1) = 1 / 2 | - |
| Example 2 | | # |
| **Document true keywords** | US dollar, exchange rate | 2 |
| **Retrieved** | US dollar | 1 |
| *tp* | US dollar | 1 |
| *fp* | - | 0 |
| **Not retrieved** | Exchange rate | 1 |
| *tn* | - | 0 |
| *fn* | Exchange rate | 1 |
| **Precision** | 1 / (1 + 0) = 1 | - |
| **Recall** | 1 / (1 + 1) = 1 / 2 | - |

**Table 5.2:** Performance metrics' calculation example.

[TKV10]. Thus, if we have a collection of files and their respective keywords, like it is the case in this dissertation, and we want to calculate precision and recall for all the documents in that collection, a micro-averaged approach consists in calculating these metrics individually for each document, while in a macro-averaged approach we count all the matches of all documents together, dividing the obtained result by the total number of files in both cases.

Consider a binary evaluation measure $B(tp, tn, fp, fn)$ that is calculated based on the number of true positives $(tp)$, true negatives $(tn)$, false positives $(fp)$ and false negatives $(fn)$ ( see Table 5.1 ). Let $tp_\lambda$, $fp_\lambda$, $tn_\lambda$ and $fn_\lambda$ be the number of true positives, false positives, true negatives and false negatives after binary evaluation for a label $\lambda$ (keyword and not-keyword in this case) [TKV10]. The *macro-averaged* and *micro-averaged* versions of $B$, are then calculated as follows:

$$B_{micro} = B\left(\sum_{\lambda=1}^{q} tp_\lambda, \sum_{\lambda=1}^{q} fp_\lambda, \sum_{\lambda=1}^{q} tn_\lambda, \sum_{\lambda=1}^{q} fn_\lambda\right) \tag{5.7}$$

$$B_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} B\left(tp, tn, fp, fn\right) \tag{5.8}$$

Note that micro-averaging has the *same* result as macro-averaging for some measures, such as *accuracy*, while it differs for other measures, such as *precision* and *recall*.

As referred in [WPF+99], there are also several disadvantages in using author keyphrases as a *gold standard*, primarily because authors do not always choose keyphrases that best describe the content of their paper and also because keyphrases are often chosen hastily, usually just before a document is finalized. The variance in author's selected keyphrases makes it obviously more difficult for an automatic extraction scheme to perform well, once that sometimes words that are not necessarily poor keyphrases are considered incorrect, just because they did not match an author's choice.

In [Hul04a], Hulth refers that the performance of state-of-the-art keyword extraction is much lower than for many other NLP-tasks, such as tagging and pars-

ing, and there is plenty room for improvement. Looking at the results obtained in recent Keyphrase Extraction Contests [Por11], which was devoted specifically to the extraction of keywords from scientific articles, one should note that, although the precision and recall of most current keyphrase extractors is still much lower compared with other NLP tasks (in the range of [0:05; 0:31] ), this does not necessarily indicate poor performance [Alv11, WPF+99]. As described in [WX08], when two human annotators were asked to label keyphrases on 308 documents, the *kappa* value (value that *measures* the agreement between human judges) for inter-agreement among them was only 0.70 [Alv11], which supports what have been said.

## 5.3 Methodology

This section presents and explains all the main steps followed during the practical phase of this work, i.e., during the elaboration and testing of the applications. The steps taken followed an logical order that are also roughly portrayed in Figure 4.1, from left to right, and are itemized above.

- (***T*1**) The first task performed consisted in finding suitable datasets to test the applications with (the input);

- (***T*2**) The second, focused the preprocessing of these datasets so they could in fact be *understood* by the applications and used in the most efficient possible way. While KEA and KUSCO already do their preprocessing internally, CRF tool does not. This task was of great value, once it enhanced the results significantly and it was gradually and systematically improved with new small ideas and corrections almost every day until its final version;

- (***T*3**) The third step was to fully test all the individual applications, using the standard n-fold cross-validation testing method. This was a very important step because, beyond evaluating the performance of the systems, it served to get even more familiar and also for better understanding their operation mode (and how they were coded), so it would become easier to change them as needed.

- (***T*4**) After getting that first version of the evaluation phase done (from now on designated by Evaluator), it was time to check the results obtained and try to improve them even more. This step consisted in the correction of errors in

the Evaluator and further refinements were performed in the preprocessing method (from now on designated as Preprocessor).

- (***T*5**) With a stable version of the Evaluator in hands, that simply accounted for full key matches and performed an macro-averaged evaluation, it was time to extend it. Thus being, the Evaluator was extended in MacroTermEvaluator and MacroTermStemEvaluator. While the first remained the same as the Evaluator itself, the TermStemEvaluator was now accounting key matches based on the stems of the words, rather than the words themselves. As expected, this improved the results. Not satisfied yet, the MicroTermEvaluator and MicroTermStemEvaluator were also added, but performing a micro-averaging evaluation, obtaining better results than the macro-averaged ones.

- (***T*6*a***) This step consisted in putting all the applications working together, i.e., a beta-version of the ensemble. This was achieved using a multi-threaded environment, one thread per application.

- (***T*6*b***) This step is an extension of the previous and comprises several modifications made to the original applications, so they could actually work together efficiently. As an example, to avoid reading the same documents three times (one time per application), a time-consuming task, it soon became evident that this needed to be done only once, before each individual system could start its job. Thus, when the application starts, the documents are immediately read and saved into memory. After that, they are split into N equal sized folds (this number is configurable and the default chosen is ten), and pointed out to each application, so the Cross-Validation test can take place. After all applications finished their training/testing phases, the results are sent to the main thread, which is in charge of combining and printing the results.

  In order to make everything running smoothly, and obtaining the expected behaviour from the final application, several methods were modified and created, which required a great comprehension about the operation of each tool.

- (***T*7**) The final task was then combining the results of each individual application. This was done using two different voting schemes: simple majority and weighted majority, both based on a *plurality* consensus.

# Chapter 6

# Experimental Results

This chapter presents the results and respective discussion of several experiences performed during the course of this dissertation which were obtained through *10-fold-cross-validation* and is divided into two main sections. Namely, Section 6.1 shows the preliminary tests carried out with one of the applications during an early phase of the work, while Section 6.2 depicts the final results of both the individual and ensemble applications.

## 6.1 Preliminary Results

Before the development of the ensemble application itself, this work comprised a preliminary test phase of a well known system in the field, actually considered one of its main contributions: the Keyphrase Extraction Algorithm or KEA.

The results drawn ahead show different analysed traits about KEA (based on the experiments reported in [WPF+99]):

- Its overall efficiency when *extracting* up to twenty keywords;
- The impact of changing the number of training files;
- The effect of varying document's length.

As discussed before (see Chapter 5.1), it was not possible to use the original dataset used then by the author. Notwithstanding, to validate the results reported there, two different datasets were used in the preliminary experiments below: Krap's and Hulth's dataset (also described in Chapter 5.1). The former was intended to be as close as possible to the original dataset (in ways that it is composed by full articles only, like the original dataset was) while the latter

| Training-set size | Average author assigned keywords per article | Average number of words per article |
|---|---|---|
| 50 | 6.28 | 9455 |

**Table 6.1:** Article information for the overall test.

| Keywords extracted | Average matches with author assigned keywords | |
|---|---|---|
| | Author reported results | Obtained results |
| 5 | 0.93 | 0.82 |
| 10 | 1.39 | 1.31 |
| 15 | 1.68 | 1.67 |
| 20 | 1.88 | 1.90 |

**Table 6.2:** KEA's overall effectiveness (average key matches per file) using Krap's dataset.

aimed simulating the usage of small *textual descriptions*, closer to those to which the final application pretends being applied to.

### 6.1.1 Overall effectiveness

The first experiment assessed KEA's overall effectiveness when extracting between 5 and 20 keywords per test document. These tests used 50 documents to train the model and a set of 500 document to test it. Table 6.2 draws the results obtained and compares them with those reported by the author, while in Table 6.3 the same results are now shown in terms of Precision, Recall and F-measure scores. Information about the documents used is displayed in Table 6.1.

### 6.1.2 Effect of training set size

In this experiment, the main interest resided in the practical problem of how many *training documents* are necessary for achieving good results. Thus being, the size of the training set was varied from 1 to 150 documents and KEA's performance

| Keywords extracted | Detailed results | | | |
|---|---|---|---|---|
| | $A$ | $P$ | $R$ | $F_1$ |
| 5 | 0.820 | 0.164 | 0.131 | 0.145 |
| 10 | 1.310 | 0.131 | 0.209 | 0.161 |
| 15 | 1.668 | 0.111 | 0.266 | 0.157 |
| 20 | 1.898 | 0.095 | 0.302 | 0.144 |

**Table 6.3:** KEA's detailed overall effectiveness (Krap's dataset). ($A$): Avg. matches/file; ($P$): Precision ; ($R$): Recall; ($F_1$): F-measure.

| | Keywords extracted | |
|---|---|---|
| | 5 | 15 |
| Full texts | 0.778 | 1.545 |
| Abstracts | 0.525 | 1.435 |

**Table 6.4:** Average keywords extracted per file, for full text and abstracts.

| Test ID | Training-set size | Average author assigned keywords per article | Average number of words per article |
|---|---|---|---|
| 1 | 1 | 6 | 8758 |
| 2 | 5 | 5.4 | 8996 |
| 3 | 10 | 5.7 | 9518 |
| 4 | 20 | 6.6 | 9382 |
| 5 | 30 | 6.5 | 9581 |
| 6 | 50 | 6.0 | 9331 |
| 7 | 100 | 6.2 | 9258 |
| 8 | 130 | 6.3 | 9299 |
| 9 | 150 | 6.2 | 9236 |

**Table 6.5:** Article's information for each test (Krap's dataset).

was tested with each set. The test set was composed by the same standard 500 documents. Table 6.6 shows the number of keywords correctly identified when 5 and 15 phrases are extracted against the number of documents used for training and Table 6.5 displays the information about each of the tests made.

### 6.1.3 Effect of document length

This experiment verified if KEA's performance is affected when applied to documents are differing in length, which is actually an important aspect to be checked, moreover because these documents are much likely to be similar (at least in length) with the textual descriptions that will be used futurely, from where one pretends extracting information.

Thus, the results shown in the next two tables can be compared with those presented above as follows: Tables 6.6 and 6.8, denoting full articles or abstracts, respectively; Tables 6.5 and 6.7, contrasting the information about the content of the documents used in the respective tests.

The training set size was varied like in the previous one and the test set used was composed with the standard 500 test documents for each dataset tested.

| Test ID | Number of keywords extracted | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 5 | | | | 15 | | | |
| | ***A*** | ***P*** | ***R*** | ***F*$_1$** | ***A*** | ***P*** | ***R*** | ***F*$_1$** |
| 1 | 0.412 | 0.082 | 0.069 | 0.075 | 0.942 | 0.063 | 0.157 | 0.090 |
| 2 | 0.624 | 0.125 | 0.116 | 0.120 | 1.468 | 0.098 | 0.272 | 0.144 |
| 3 | 0.784 | 0.157 | 0.138 | 0.147 | 1.502 | 0.100 | 0.264 | 0.145 |
| 4 | 0.812 | 0.162 | 0.123 | 0.140 | 1.614 | 0.108 | 0.245 | 0.149 |
| 5 | 0.805 | 0.156 | 0.120 | 0.136 | 1.602 | 0.107 | 0.246 | 0.149 |
| 6 | 0.838 | 0.168 | 0.139 | 0.152 | 1.674 | 0.112 | 0.277 | 0.159 |
| 7 | 0.894 | 0.179 | 0.144 | 0.160 | 1.684 | 0.112 | 0.271 | 0.159 |
| 8 | 0.912 | 0.182 | 0.146 | 0.162 | 1.716 | 0.114 | 0.274 | 0.161 |
| 9 | 0.922 | 0.184 | 0.148 | 0.164 | 1.706 | 0.114 | 0.274 | 0.161 |

**Table 6.6:** Results obtained with complete documents (Krap's dataset). (***A***): Avg. matches/file; (***P***): Precision ; (***R***): Recall; (***F*$_1$**): F-measure.

| Test ID | Training-set size | Average author assigned keywords per article | Average number of words per article |
|---|---|---|---|
| 1 | 1 | 5.0 | 71 |
| 2 | 5 | 6.6 | 143 |
| 3 | 10 | 7.5 | 124 |
| 4 | 20 | 7.2 | 102 |
| 5 | 30 | 6.9 | 99 |
| 6 | 50 | 7.1 | 107 |
| 7 | 100 | 7.0 | 107 |
| 8 | 130 | 7.0 | 106 |
| 9 | 150 | 7.1 | 106 |

**Table 6.7:** Abstracts' information (Hulth's dataset).

### 6.1.4 Discussion

The overall results of the preliminary experimentations made, portrayed in Table 6.4, are in concordance with those presented by the author. The average number of keywords that were chosen by KEA and also by the document's author, when a fixed number of keywords are extracted, was practically the same as reported in [WPF+99]. However, the values for the other metrics tested are not very high, which does not necessarily indicates bad performance, as explained in Section 5.2.

In second place, results depicted in Tables 6.6 and 6.8 demonstrate that performance improved proportionally to the size of the training set, for both tests made (full text and abstracts). When more than 50 documents are used to train the algorithm, the gains obtained are consecutively smaller. These results are

| Test ID | Number of keywords extracted | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 5 | | | | 15 | | | |
| | $A$ | $P$ | $R$ | $F_1$ | $A$ | $P$ | $R$ | $F_1$ |
| 1 | 0.412 | 0.082 | 0.082 | 0.082 | 1.282 | 0.085 | 0.256 | 0.128 |
| 2 | 0.458 | 0.092 | 0.069 | 0.079 | 1.360 | 0.091 | 0.206 | 0.126 |
| 3 | 0.476 | 0.095 | 0.063 | 0.076 | 1.394 | 0.093 | 0.186 | 0.124 |
| 4 | 0.518 | 0.104 | 0.071 | 0.085 | 1.430 | 0.095 | 0.197 | 0.129 |
| 5 | 0.512 | 0.102 | 0.074 | 0.086 | 1.432 | 0.095 | 0.207 | 0.131 |
| 6 | 0.584 | 0.117 | 0.082 | 0.096 | 1.472 | 0.098 | 0.206 | 0.133 |
| 7 | 0.604 | 0.121 | 0.086 | 0.100 | 1.518 | 0.101 | 0.216 | 0.138 |
| 8 | 0.576 | 0.115 | 0.082 | 0.096 | 1.506 | 0.100 | 0.214 | 0.137 |
| 9 | 0.590 | 0.118 | 0.083 | 0.097 | 1.522 | 0.101 | 0.214 | 0.138 |

**Table 6.8:** Results obtained with abstracts (Hulth's dataset). ($A$): Avg. matches/file; ($P$): Precision ; ($R$): Recall; ($F_1$): F-measure.

an indicator that good extraction performance can be acquired with a relatively small set of training documents, but the gathered data clearly indicates that the more training files, the better the achieved performance. Despite that, if one had a collection without any keywords assigned to be processed, human experts would only need reading and assigning keywords to a minimum of 20 documents, in order to extract reasonably keywords from the rest of the collection. If less than 20 documents are used, however, one can see that KEA does not perform as good as expected.

The last test made showed that better performance is achieved for large documents rather than for small documents (Tables 6.6 and 6.8).

Nevertheless, while results obtained with abstracts (for 5 keywords) are a lot lower than those with large documents, the differences in performance between both cases seems to decrease as more keywords are extracted. The average values shown in Table 6.4 seem to support that.

Another pertinent observation has to do with the standard deviation in the results obtained (whether KEA is applied to full texts or abstracts). It seems that as the size of documents increase, the higher the deviation between the initial and final corresponding tests (test $n^o$**1** and $n^o$**9**).

Having that said, two possible conclusions arise when comparing the results obtained. First, from the *larger documents point of view*, because these documents have an higher word average, one can conclude that KEA struggles in identifying the words that are actually important within the document in question, those representing true keywords, in cases where the training files are scarce (less than

20, as discussed in 6.1.2). Second, from the *smaller documents point of view*, because these documents have such a small number of words compared with full text documents above, consequently giving less useful information for the algorithm to perform well.

Concluding, the preliminary results here shown that the source of dataset does not significantly affects the performance of the tested algorithm. In fact, looking at the overall test, Table 6.2, it is visible that a very similar performance was achieved here and in author's experiments.

It is also notorious that the number of training documents influences considerably the performance of the system and that fact is even more visible when the size of the documents increases. Also, the results shown here clearly indicate that the number of training documents and document length is correlated, and better results were achieved when using full documents rather than abstracts.

Additionally, it is expected that the quality and number of labelled keywords in both datasets is influencing the results obtained (as seen in Section 5.1, only 76.2% of the terms labelled as keywords in Hulth's dataset is in fact present in the abstracts).

## 6.2 Final Results

For the final results presented here it is worth noting that, contrary to what happened in the preliminary experimental phase where both full articles and abstracts were used, it was not possible testing with the former basically due to the time (very long time indeed) that it takes training the CRF classifier model using large documents, which is the main problem of this application. Instead, from the full articles dataset (Krap's dataset), only the title and abstract were considered, turning both dataset's documents very similar in terms of length.

That fact can be justified not only because the number of features being applied to the texts (twenty-one, as depicted in Figure 2.1), but also because of its own nature (imagine calculating TF*IDF scores and tagging the PoS for all the terms in the documents...), which is directly proportional to the time it take to train the model. So, the longer the text, the longer it takes the preprocessing and consequently the training and testing phases. Just to give an overall idea of the time it takes, even for the 2000 small abstracts containing roughly between 5 and 15 lines, the training time of the 10-fold-cross-validation was roughly between 2-3 hours per run. Testing the ensemble with full documents containing roughly

| Weighted version | CRF weight | KUSCO weight | KEA weight |
|---|---|---|---|
| WMV 1 (CRF $F_1 < 0.48$) | 54% | 36% | 10% |
| WMV 2 (CRF $F_1 > 0.48$) | 62% | 30% | 8% |

**Table 6.9:** Model weights used in the final ensemble application.

between 500 and 1000 lines would take in the best of the scenarios at least 100 times more, thus turning out impossible performing all tests that were actually made, in due time.

Beyond what has already been said, two concerns about the tests here presented need yet to be discussed:

- The number of keywords extracted by each application:

  - from the applications used here, only KEA allows to define the concrete number of keywords to be extracted; KUSCO and CRF do not and they only extract as much keywords as they can. So, instead of defining a specific number, the approach taken was to set a limit of keywords that can be extracted. For example, rather than saying *Extract five keywords*, we say *Extract a maximum of five keywords*. Best results were achieved when setting a maximum of thirty keywords per application (although the average being actually lower);

- The weights given to each application classifier, in the weighted version of the majority voting:

  - here two different weight combinations were used: the first one (*WMV 1*) when the reliability of the CRF system (measured as $F_1$ score) is approximately lower than 0.48 and the second one (*WMV 2*) when it is higher, values that came from the empirical results performed, depicted in Table 6.9.

Further details are given ahead in this chapter.

### 6.2.1 Hulth & Krap's Datasets

The idea for the final experimentation phase was testing the application using both abstracts and full articles, before validating the results with event datasets. However, as mentioned earlier, the latter test could not be done due to time constraints, namely of CRF application. Thus, both datasets here tested ended up being very similar in terms of document's length and that is why both results are

shown grouped in the same section.

In Tables 6.10 and 6.11 information about the documents used for testing and about the tests themselves is given. Some relevant concerns taken during the experimentation are itemized and explained below as well:

- The tables showing the results contain four types of *Evaluators*, depending on the method used to compare the matches. The names are quite elucidative themselves, but to avoid any misunderstanding they are detailed here:
  - **MacroTermEvaluator**: evaluation performed based on full matches only, macro-averaged;
  - **MacroTermStemEvaluator**: evaluation performed based on stem matches, macro-averaged;
  - **MicroTermEvaluator**: evaluation performed based on full matches only, micro-averaged;
  - **MicroTermStemEvaluator**: evaluation performed based on stem matches. micro-averaged;
- Tests ranging from 1 to 5 (Hulth's dataset) and Test 5 (Krap's dataset) used all the documents in that collection; For Krap's dataset, tests 1, 2, 3 and 4 were not performed because as we extracted the title and abstract from the full documents, all the possible miss-indentations, miss-structuring and missing keywords were immediately corrected and the labelling method already used merely the stems, rather than the full keywords;
- Test 6 was performed removing all the documents whose keywords contained digits;
- Test 7, additionally to the previous constraint used in Test 6, was performed using only documents that had between 5 and 10 keywords;
- Test 8 exploits another method of accounting for matches, which it is detailed in its respective sub-section, called **SegmentEvaluator**.

**Test 1 - Original Dataset**

This test can be seen as the baseline test. It was performed knowing in advance that only about 76% of the keywords were in fact present in the abstracts (as

| Test Number | Documents used | Average author assigned keywords per document | Average number of words per document |
|---|---|---|---|
| 1 | 1990 | 9.6 | 161.1 |
| 2-5 | 1990 | 7.3 | 161.1 |
| 6 | 1080 | 7.4 | 158.3 |
| 7-8 | 976 | 7.4 | 158.2 |

**Table 6.10:** Abstract's information (Hulth's dataset).

| Test Number | Documents used | Average author assigned keywords per document (abstract and title only) | Average number of words per document (abstract and title only) |
|---|---|---|---|
| 5 | 2304 | 3 | 188.4 |
| 6 | 2270 | 3 | 188.3 |
| 7-8 | 293 | 6.1 | 205.5 |

**Table 6.11:** Abstracts' information (Krap's dataset).

referenced in Section 5.1). This fact will negatively impact the results because no extraction-based tool can extract something that is simply not there. Beyond that, this first test was also conducted without any kind of preprocessing being applied to the text files. Results are shown in Table 6.12.

#### Test 2 - Removing unseen Keywords

This second test is almost identical to the first one, but with one big difference: keywords that did not exist in the abstracts were removed from the respective file's true keywords. At this point, it is guaranteed that 100% of the keywords can be in fact found in the document they pertain which, as one can see in Table 6.13, improved the results, as expected.

#### Test 3 - Document structuring using OpenNLP Sentence Splitter

Another relevant fact noticed, was the way that documents were structured and written, containing multiple spaces, tabs and even line breaks in the middle of sentences. Thus, the third test consisted in separating the lines of the documents correctly, using the *OpenNLP Sentence Splitter* for the task. Despite no further enhancement being expected in KEA and KUSCO, once they do not have document structure in account, having the documents correctly organized improved the learning of the CRF system, generating considerably better results.

| *Application* | *Average keys extracted* | ***A*** | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.682 | 0.124 | 0.383 | 0.182 |
| *MacroTermStemEvaluator* | ⟶ | 3.819 | 0.129 | 0.397 | 0.19 |
| *MicroTermEvaluator* | ⟶ | 3.682 | 0.125 | 0.439 | 0.19 |
| *MicroTermStemEvaluator* | ⟶ | 3.819 | 0.129 | 0.456 | 0.198 |
| ******* KUSCO ******* | 13.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.45 | 0.104 | 0.151 | 0.11 |
| *MacroTermStemEvaluator* | ⟶ | 1.745 | 0.126 | 0.181 | 0.148 |
| *MicroTermEvaluator* | ⟶ | 1.45 | 0.112 | 0.166 | 0.133 |
| *MicroTermStemEvaluator* | ⟶ | 1.745 | 0.134 | 0.198 | 0.16 |
| ******* CRF ******* | 4.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.022 | 0.447 | 0.21 | 0.284 |
| *MacroTermStemEvaluator* | ⟶ | 2.09 | 0.462 | 0.217 | 0.294 |
| *MicroTermEvaluator* | ⟶ | 2.022 | 0.438 | 0.219 | 0.29 |
| *MicroTermStemEvaluator* | ⟶ | 2.09 | 0.452 | 0.227 | 0.301 |
| ******* ENSEMBLE ******* (MV) | 6.3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.276 | 0.362 | 0.236 | 0.286 |
| *MacroTermStemEvaluator* | ⟶ | 2.338 | 0.372 | 0.243 | 0.294 |
| *MicroTermEvaluator* | ⟶ | 2.276 | 0.38 | 0.257 | 0.307 |
| *MicroTermStemEvaluator* | ⟶ | 2.338 | 0.39 | 0.265 | 0.315 |
| ******* ENSEMBLE ******* (WMV 1) | 6.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.586 | 0.381 | 0.269 | 0.315 |
| *MacroTermStemEvaluator* | ⟶ | 2.711 | 0.4 | 0.282 | 0.33 |
| *MicroTermEvaluator* | ⟶ | 2.586 | 0.401 | 0.289 | 0.335 |
| *MicroTermStemEvaluator* | ⟶ | 2.711 | 0.421 | 0.304 | 0.352 |
| ******* ENSEMBLE ******* (WMV 2) | 5.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.349 | 0.417 | 0.244 | 0.307 |
| *MacroTermStemEvaluator* | ⟶ | 2.469 | 0.439 | 0.256 | 0.322 |
| *MicroTermEvaluator* | ⟶ | 2.349 | 0.43 | 0.262 | 0.324 |
| *MicroTermStemEvaluator* | ⟶ | 2.469 | 0.454 | 0.276 | 0.342 |

**Table 6.12:** Test 1 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.382 | 0.114 | 0.519 | 0.187 |
| *MacroTermStemEvaluator* | ⟶ | 3.453 | 0.117 | 0.529 | 0.191 |
| *MicroTermEvaluator* | ⟶ | 3.382 | 0.114 | 0.581 | 0.191 |
| *MicroTermStemEvaluator* | ⟶ | 3.453 | 0.117 | 0.593 | 0.195 |
| ******* KUSCO ******* | 14.1 | - | - | - | - |
| *TermEvaluator* | ⟶ | 1.445 | 0.102 | 0.199 | 0.135 |
| *TermStemEvaluator* | ⟶ | 1.701 | 0.12 | 0.234 | 0.159 |
| *MicroTermEvaluator* | ⟶ | 1.445 | 0.112 | 0.2 | 0.143 |
| *MicroTermStemEvaluator* | ⟶ | 1.701 | 0.133 | 0.237 | 0.17 |
| ******* CRF ******* | 4.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.027 | 0.448 | 0.31 | 0.364 |
| *MacroTermStemEvaluator* | ⟶ | 2.075 | 0.459 | 0.318 | 0.373 |
| *MicroTermEvaluator* | ⟶ | 2.027 | 0.438 | 0.316 | 0.365 |
| *MicroTermStemEvaluator* | ⟶ | 2.075 | 0.449 | 0.324 | 0.374 |
| ******* ENSEMBLE ******* (MV) | 6.3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.171 | 0.345 | 0.333 | 0.338 |
| *MacroTermStemEvaluator* | ⟶ | 2.207 | 0.351 | 0.338 | 0.344 |
| *MicroTermEvaluator* | ⟶ | 2.171 | 0.363 | 0.355 | 0.359 |
| *MicroTermStemEvaluator* | ⟶ | 2.207 | 0.369 | 0.362 | 0.365 |
| ******* ENSEMBLE ******* (WMV 1) | 6.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.484 | 0.366 | 0.38 | 0.372 |
| *MacroTermStemEvaluator* | ⟶ | 2.583 | 0.381 | 0.395 | 0.387 |
| *MicroTermEvaluator* | ⟶ | 2.484 | 0.384 | 0.404 | 0.393 |
| *MicroTermStemEvaluator* | ⟶ | 2.583 | 0.401 | 0.42 | 0.41 |
| ******* ENSEMBLE ******* (WMV 2) | 6.1 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.395 | 0.391 | 0.367 | 0.378 |
| *MacroTermStemEvaluator* | ⟶ | 2.493 | 0.407 | 0.382 | 0.393 |
| *MicroTermEvaluator* | ⟶ | 2.395 | 0.409 | 0.389 | 0.398 |
| *MicroTermStemEvaluator* | ⟶ | 2.493 | 0.428 | 0.406 | 0.416 |

**Table 6.13:** Test 2 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | ***A*** | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 4.088 | 0.137 | 0.562 | 0.22 |
| *MacroTermStemEvaluator* | ⟶ | 4.12 | 0.138 | 0.566 | 0.222 |
| *MicroTermEvaluator* | ⟶ | 4.088 | 0.137 | 0.577 | 0.221 |
| *MicroTermStemEvaluator* | ⟶ | 4.12 | 0.138 | 0.581 | 0.223 |
| ******* KUSCO ******* | 13.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.377 | 0.099 | 0.211 | 0.135 |
| *MacroTermStemEvaluator* | ⟶ | 1.659 | 0.119 | 0.254 | 0.162 |
| *MicroTermEvaluator* | ⟶ | 1.377 | 0.108 | 0.256 | 0.151 |
| *MicroTermStemEvaluator* | ⟶ | 1.659 | 0.129 | 0.302 | 0.18 |
| ******* CRF ******* | 6.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.036 | 0.474 | 0.417 | 0.442 |
| *MacroTermStemEvaluator* | ⟶ | 3.131 | 0.489 | 0.43 | 0.456 |
| *MicroTermEvaluator* | ⟶ | 3.036 | 0.502 | 0.42 | 0.456 |
| *MicroTermStemEvaluator* | ⟶ | 3.131 | 0.517 | 0.434 | 0.471 |
| ******* ENSEMBLE ******* (MV) | 7.7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.086 | 0.402 | 0.423 | 0.412 |
| *MacroTermStemEvaluator* | ⟶ | 3.145 | 0.41 | 0.431 | 0.42 |
| *MicroTermEvaluator* | ⟶ | 3.086 | 0.433 | 0.431 | 0.431 |
| *MicroTermStemEvaluator* | ⟶ | 3.145 | 0.443 | 0.439 | 0.44 |
| ******* ENSEMBLE ******* (WMV 1) | 7.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.377 | 0.43 | 0.463 | 0.445 |
| *MacroTermStemEvaluator* | ⟶ | 3.497 | 0.445 | 0.48 | 0.461 |
| *MicroTermEvaluator* | ⟶ | 3.377 | 0.466 | 0.468 | 0.466 |
| *MicroTermStemEvaluator* | ⟶ | 3.497 | 0.486 | 0.485 | 0.485 |
| ******* ENSEMBLE ******* (WMV 2) | 7.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.289 | 0.444 | 0.451 | 0.446 |
| *MacroTermStemEvaluator* | ⟶ | 3.405 | 0.46 | 0.467 | 0.462 |
| *MicroTermEvaluator* | ⟶ | 3.289 | 0.477 | 0.455 | 0.465 |
| *MicroTermStemEvaluator* | ⟶ | 3.405 | 0.496 | 0.473 | 0.483 |

**Table 6.14:** Test 3 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | ***A*** | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 4.114 | 0.137 | 0.564 | 0.221 |
| *MacroTermStemEvaluator* | ⟶ | 4.148 | 0.139 | 0.569 | 0.223 |
| *MicroTermEvaluator* | ⟶ | 4.114 | 0.138 | 0.58 | 0.222 |
| *MicroTermStemEvaluator* | ⟶ | 4.148 | 0.139 | 0.585 | 0.224 |
| ******* KUSCO ******* | 14.1 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.427 | 0.102 | 0.196 | 0.134 |
| *MacroTermStemEvaluator* | ⟶ | 1.708 | 0.122 | 0.234 | 0.16 |
| *MicroTermEvaluator* | ⟶ | 1.427 | 0.109 | 0.197 | 0.14 |
| *MicroTermStemEvaluator* | ⟶ | 1.708 | 0.131 | 0.237 | 0.169 |
| ******* CRF ******* | 6.2 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.088 | 0.502 | 0.423 | 0.456 |
| *MacroTermStemEvaluator* | ⟶ | 3.176 | 0.516 | 0.435 | 0.469 |
| *MicroTermEvaluator* | ⟶ | 3.088 | 0.528 | 0.426 | 0.469 |
| *MicroTermStemEvaluator* | ⟶ | 3.176 | 0.542 | 0.439 | 0.483 |
| ******* ENSEMBLE ******* (MV) | 7.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.13 | 0.414 | 0.429 | 0.42 |
| *MacroTermStemEvaluator* | ⟶ | 3.175 | 0.42 | 0.435 | 0.426 |
| *MicroTermEvaluator* | ⟶ | 3.13 | 0.446 | 0.436 | 0.44 |
| *MicroTermStemEvaluator* | ⟶ | 3.175 | 0.453 | 0.443 | 0.447 |
| ******* ENSEMBLE ******* (WMV 1) | 7.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.402 | 0.449 | 0.466 | 0.455 |
| *MacroTermStemEvaluator* | ⟶ | 3.515 | 0.464 | 0.482 | 0.47 |
| *MicroTermEvaluator* | ⟶ | 3.402 | 0.482 | 0.471 | 0.474 |
| *MicroTermStemEvaluator* | ⟶ | 3.515 | 0.501 | 0.487 | 0.492 |
| ******* ENSEMBLE ******* (WMV 2) | 7.2 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.335 | 0.466 | 0.457 | 0.459 |
| *MacroTermStemEvaluator* | ⟶ | 3.447 | 0.482 | 0.472 | 0.475 |
| *MicroTermEvaluator* | ⟶ | 3.335 | 0.496 | 0.461 | 0.476 |
| *MicroTermStemEvaluator* | ⟶ | 3.447 | 0.516 | 0.478 | 0.494 |

**Table 6.15:** Test 4 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

This concern had already been mentioned in Section 4.2, but here tests' results are depicted in Table 6.14.

#### Test 4 - Stem-based keyword labelling method

As already referenced before, CRF application required true keywords being labelled directly in text files, rather than having them in a separate file as it happens with KEA. To achieve that, the first approach taken was tagging them in

documents exactly as they appeared in their respective .key files. That process, however, led to unwanted miss-labellings because some keywords were written in plural and only its singular form appeared in text and vice-versa. Instead of searching for full matches only, the solution passed by doing the labelling based on keyword's stems. This solved the problem of plural/singulars and, of course, also identified words in text that had the same base stem then the given keyword and were passing without being labelled. Results obtained after this change are presented in Table 6.15.

**Test 5 - The new Porter Stemmer**

Although the previous cited method for labelling has improved the applications' performances, a closer look at the keywords being compared in the evaluation phase showed that the stemmer used (the English Porter Stemmer) was not performing as expected for some cases. It is known that stemmers are not 100% accurate but, after a quick search on the Internet, a version of the same stemmer with several bugs corrected, was in fact available for download. After changing the stemmer a new test was performed and results are portrayed in Table 6.16 (Hulth's dataset) and in Table 6.17.

**Test 6 - Document filtering: digits in true keywords**

During the development of this work, soon became evident that the structure of the documents was influencing a lot the results obtained and that became even more visible with CRF, once this system can utilize most of the characteristics present in the text.

As referenced before, the preprocessing phase was constantly improved while errors were being found or the results of the experiments were not as expected. One example of this is depicted below and it actually led to a future work idea (namely that of applying clustering to the datasets before the training of the models): After finding that decimal numbers were being split, when they should not be (E.g: 1.5 MB was split in 1 . 5 MB), the error was quickly corrected but, contrary to what was initially expected, the results obtained did not only not improve but they were in fact a little worse. The fact that numbers being not split (i.e., correctly preprocessed) were causing worse results was a little strange, but after a couple of additional tests the source of the problem was found. Because there were documents whose true keywords contained digits, when the numbers in the text were actually split caused those keywords to be obviously not found

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 4.146 | 0.139 | 0.573 | 0.223 |
| *MacroTermStemEvaluator* | ⟶ | 4.179 | 0.14 | 0.578 | 0.225 |
| *MicroTermEvaluator* | ⟶ | 4.146 | 0.139 | 0.59 | 0.225 |
| *MicroTermStemEvaluator* | ⟶ | 4.179 | 0.14 | 0.595 | 0.226 |
| ******* KUSCO ******* | 14.1 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.451 | 0.103 | 0.201 | 0.136 |
| *MacroTermStemEvaluator* | ⟶ | 1.745 | 0.124 | 0.241 | 0.163 |
| *MicroTermEvaluator* | ⟶ | 1.451 | 0.11 | 0.201 | 0.142 |
| *MicroTermStemEvaluator* | ⟶ | 1.745 | 0.134 | 0.243 | 0.172 |
| ******* CRF ******* | 6.4 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.139 | 0.494 | 0.434 | 0.462 |
| *MacroTermStemEvaluator* | ⟶ | 3.236 | 0.509 | 0.447 | 0.476 |
| *MicroTermEvaluator* | ⟶ | 3.139 | 0.524 | 0.436 | 0.476 |
| *MicroTermStemEvaluator* | ⟶ | 3.236 | 0.54 | 0.451 | 0.491 |
| ******* ENSEMBLE ******* (MV) | 7.7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.173 | 0.415 | 0.439 | 0.426 |
| *MacroTermStemEvaluator* | ⟶ | 3.223 | 0.421 | 0.446 | 0.433 |
| *MicroTermEvaluator* | ⟶ | 3.173 | 0.447 | 0.446 | 0.446 |
| *MicroTermStemEvaluator* | ⟶ | 3.223 | 0.454 | 0.453 | 0.453 |
| ******* ENSEMBLE ******* (WMV 1) | 7.7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.441 | 0.446 | 0.476 | 0.46 |
| *MacroTermStemEvaluator* | ⟶ | 3.555 | 0.46 | 0.491 | 0.475 |
| *MicroTermEvaluator* | ⟶ | 3.441 | 0.479 | 0.48 | 0.479 |
| *MicroTermStemEvaluator* | ⟶ | 3.555 | 0.497 | 0.497 | 0.497 |
| ******* ENSEMBLE ******* (WMV 2) | 7.3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.373 | 0.461 | 0.466 | 0.463 |
| *MacroTermStemEvaluator* | ⟶ | 3.487 | 0.476 | 0.482 | 0.479 |
| *MicroTermEvaluator* | ⟶ | 3.373 | 0.492 | 0.47 | 0.48 |
| *MicroTermStemEvaluator* | ⟶ | 3.487 | 0.511 | 0.487 | 0.498 |

**Table 6.16:** Test 5 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 28.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.751 | 0.096 | 0.746 | 0.171 |
| *MacroTermStemEvaluator* | ⟶ | 2.772 | 0.097 | 0.752 | 0.172 |
| *MicroTermEvaluator* | ⟶ | 2.751 | 0.099 | 0.76 | 0.175 |
| *MicroTermStemEvaluator* | ⟶ | 2.772 | 0.099 | 0.765 | 0.176 |
| ******* KUSCO ******* | 9.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 0.964 | 0.098 | 0.262 | 0.143 |
| *MacroTermStemEvaluator* | ⟶ | 1.174 | 0.119 | 0.319 | 0.174 |
| *MicroTermEvaluator* | ⟶ | 0.964 | 0.119 | 0.268 | 0.165 |
| *MicroTermStemEvaluator* | ⟶ | 1.174 | 0.144 | 0.325 | 0.199 |
| ******* CRF ******* | 2.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.242 | 0.449 | 0.337 | 0.384 |
| *MacroTermStemEvaluator* | ⟶ | 1.289 | 0.466 | 0.349 | 0.399 |
| *MicroTermEvaluator* | ⟶ | 1.242 | 0.448 | 0.341 | 0.387 |
| *MicroTermStemEvaluator* | ⟶ | 1.289 | 0.462 | 0.356 | 0.401 |
| ******* ENSEMBLE ******* (MV) | 4.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.546 | 0.324 | 0.42 | 0.366 |
| *MacroTermStemEvaluator* | ⟶ | 1.575 | 0.33 | 0.427 | 0.372 |
| *MicroTermEvaluator* | ⟶ | 1.546 | 0.361 | 0.429 | 0.392 |
| *MicroTermStemEvaluator* | ⟶ | 1.575 | 0.367 | 0.438 | 0.399 |
| ******* ENSEMBLE ******* (WMV 1) | 4.3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.601 | 0.371 | 0.434 | 0.4 |
| *MacroTermStemEvaluator* | ⟶ | 1.696 | 0.393 | 0.46 | 0.424 |
| *MicroTermEvaluator* | ⟶ | 1.601 | 0.403 | 0.443 | 0.422 |
| *MicroTermStemEvaluator* | ⟶ | 1.696 | 0.427 | 0.469 | 0.447 |
| ******* ENSEMBLE ******* (WMV 2) | 3.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.531 | 0.392 | 0.415 | 0.403 |
| *MacroTermStemEvaluator* | ⟶ | 1.626 | 0.417 | 0.441 | 0.428 |
| *MicroTermEvaluator* | ⟶ | 1.531 | 0.415 | 0.423 | 0.419 |
| *MicroTermStemEvaluator* | ⟶ | 1.626 | 0.441 | 0.449 | 0.445 |

**Table 6.17:** Test 5 results - Kraps's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.949 | 0.134 | 0.536 | 0.214 |
| *MacroTermStemEvaluator* | ⟶ | 3.98 | 0.135 | 0.54 | 0.216 |
| *MicroTermEvaluator* | ⟶ | 3.949 | 0.134 | 0.606 | 0.219 |
| *MicroTermStemEvaluator* | ⟶ | 3.98 | 0.135 | 0.61 | 0.221 |
| ******* KUSCO ******* | 13.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.473 | 0.107 | 0.2 | 0.139 |
| *MacroTermStemEvaluator* | ⟶ | 1.784 | 0.129 | 0.242 | 0.169 |
| *MicroTermEvaluator* | ⟶ | 1.473 | 0.116 | 0.245 | 0.157 |
| *MicroTermStemEvaluator* | ⟶ | 1.784 | 0.14 | 0.289 | 0.188 |
| ******* CRF ******* | 6.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.268 | 0.505 | 0.444 | 0.472 |
| *MacroTermStemEvaluator* | ⟶ | 3.36 | 0.519 | 0.456 | 0.485 |
| *MicroTermEvaluator* | ⟶ | 3.268 | 0.514 | 0.454 | 0.481 |
| *MicroTermStemEvaluator* | ⟶ | 3.36 | 0.527 | 0.469 | 0.496 |
| ******* ENSEMBLE ******* (MV) | 7.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.165 | 0.417 | 0.43 | 0.423 |
| *MacroTermStemEvaluator* | ⟶ | 3.211 | 0.423 | 0.436 | 0.429 |
| *MicroTermEvaluator* | ⟶ | 3.165 | 0.435 | 0.458 | 0.446 |
| *MicroTermStemEvaluator* | ⟶ | 3.211 | 0.441 | 0.465 | 0.453 |
| ******* ENSEMBLE ******* (WMV 1) | 7.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.558 | 0.454 | 0.483 | 0.468 |
| *MacroTermStemEvaluator* | ⟶ | 3.669 | 0.468 | 0.498 | 0.482 |
| *MicroTermEvaluator* | ⟶ | 3.558 | 0.47 | 0.504 | 0.486 |
| *MicroTermStemEvaluator* | ⟶ | 3.669 | 0.486 | 0.522 | 0.503 |
| ******* ENSEMBLE ******* (WMV 2) | 7.4 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.496 | 0.471 | 0.475 | 0.472 |
| *MacroTermStemEvaluator* | ⟶ | 3.605 | 0.486 | 0.49 | 0.487 |
| *MicroTermEvaluator* | ⟶ | 3.496 | 0.483 | 0.493 | 0.488 |
| *MicroTermStemEvaluator* | ⟶ | 3.605 | 0.5 | 0.511 | 0.505 |

**Table 6.18:** Test 6 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 28.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.786 | 0.098 | 0.754 | 0.173 |
| *MacroTermStemEvaluator* | ⟶ | 2.808 | 0.098 | 0.76 | 0.174 |
| *MicroTermEvaluator* | ⟶ | 2.786 | 0.1 | 0.767 | 0.177 |
| *MicroTermStemEvaluator* | ⟶ | 2.808 | 0.101 | 0.772 | 0.178 |
| ******* KUSCO ******* | 9.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 0.98 | 0.098 | 0.265 | 0.143 |
| *MacroTermStemEvaluator* | ⟶ | 1.185 | 0.119 | 0.321 | 0.174 |
| *MicroTermEvaluator* | ⟶ | 0.98 | 0.121 | 0.27 | 0.167 |
| *MicroTermStemEvaluator* | ⟶ | 1.185 | 0.146 | 0.326 | 0.201 |
| ******* CRF ******* | 2.7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.23 | 0.452 | 0.333 | 0.383 |
| *MacroTermStemEvaluator* | ⟶ | 1.282 | 0.471 | 0.347 | 0.399 |
| *MicroTermEvaluator* | ⟶ | 1.23 | 0.452 | 0.341 | 0.388 |
| *MicroTermStemEvaluator* | ⟶ | 1.282 | 0.467 | 0.357 | 0.404 |
| ******* ENSEMBLE ******* (MV) | 4.7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.549 | 0.33 | 0.419 | 0.369 |
| *MacroTermStemEvaluator* | ⟶ | 1.577 | 0.335 | 0.426 | 0.375 |
| *MicroTermEvaluator* | ⟶ | 1.549 | 0.365 | 0.43 | 0.394 |
| *MicroTermStemEvaluator* | ⟶ | 1.577 | 0.371 | 0.438 | 0.401 |
| ******* ENSEMBLE ******* (WMV 1) | 4.3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.606 | 0.374 | 0.434 | 0.402 |
| *MacroTermStemEvaluator* | ⟶ | 1.701 | 0.397 | 0.46 | 0.426 |
| *MicroTermEvaluator* | ⟶ | 1.606 | 0.409 | 0.442 | 0.424 |
| *MicroTermStemEvaluator* | ⟶ | 1.701 | 0.431 | 0.469 | 0.449 |
| ******* ENSEMBLE ******* (WMV 2) | 3.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.537 | 0.396 | 0.416 | 0.405 |
| *MacroTermStemEvaluator* | ⟶ | 1.632 | 0.421 | 0.442 | 0.431 |
| *MicroTermEvaluator* | ⟶ | 1.537 | 0.423 | 0.424 | 0.424 |
| *MicroTermStemEvaluator* | ⟶ | 1.632 | 0.449 | 0.451 | 0.45 |

**Table 6.19:** Test 6 results - Kraps's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

and consequently being not labelled as well. The results were suggesting that these documents were then the cause of the decrease in the systems' performance.

As so, this test was performed after removing all the documents having keywords containing numbers, which represented about 10% of the total for Hulth's dataset and even less than that for Krap's dataset. Although this document removal may seem a bit like *"cheating"*, after observing the removed documents' keywords one could see that those were in fact bad ones, which could be causing the algorithm to struggle when faced with that kind of files. Keywords like *30 MB*, *1.5 MB* (are we talking about the speed of an Internet connection, about the capacity of an hard disk?) or *5 HZ* (we can also be talking about the frequency of many different things...) are not meaningful to describe any document once they do not actually give any valuable information about the topics discussed there. The results obtained after this filtering are shown in Table 6.18 (Hulth's dataset) and in Table 6.19 (Krap's dataset).

**Test 7 - Document filtering: number of true keywords**

As mentioned earlier, this test used only documents having between *five* and *ten* keywords and whose keywords did not contain digits. The objective of this test was verifying if document type (like the number and type of keywords, etc.) was in fact influencing the results.

The idea to applying this filter came because while some documents only had one assigned keyword, others had more than fifteen, i.e., the difference between minimum and maximum number of keywords was to big, causing some training groups to have considerably different average of keywords per file, making it difficult to understand and compare the differences caused by changing the number of documents used in the cross-validation and to interpret the respective results. This was in fact the last of the standard tests that were performed and results obtained are shown in Table 6.20 (Hulth's dataset) and in Table 6.21 (Krap's dataset).

**Test 8 - The MicroSegmentEvaluator**

During the experimentation phase, a problem that can actually be considered serious (because it largely affected the results obtained) was detected and is related with traditional ways that predicted keywords are compared to their gold standard ones: the *full matching* and *stem matching* method. Despite the latter improved the results over the former, it only mitigated part of the problem. An

| *Application* | *Average keys extracted* | ***A*** | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 4.399 | 0.147 | 0.602 | 0.236 |
| *MacroTermStemEvaluator* | ⟶ | 4.469 | 0.149 | 0.612 | 0.24 |
| *MicroTermEvaluator* | ⟶ | 4.399 | 0.147 | 0.618 | 0.238 |
| *MicroTermStemEvaluator* | ⟶ | 4.469 | 0.15 | 0.628 | 0.242 |
| ******* KUSCO ******* | 13.7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.614 | 0.118 | 0.221 | 0.154 |
| *MacroTermStemEvaluator* | ⟶ | 1.957 | 0.143 | 0.268 | 0.186 |
| *MicroTermEvaluator* | ⟶ | 1.614 | 0.126 | 0.221 | 0.161 |
| *MicroTermStemEvaluator* | ⟶ | 1.957 | 0.155 | 0.27 | 0.196 |
| ******* CRF ******* | 6.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.304 | 0.506 | 0.452 | 0.477 |
| *MacroTermStemEvaluator* | ⟶ | 3.417 | 0.523 | 0.468 | 0.493 |
| *MicroTermEvaluator* | ⟶ | 3.304 | 0.536 | 0.458 | 0.493 |
| *MicroTermStemEvaluator* | ⟶ | 3.417 | 0.552 | 0.474 | 0.509 |
| ******* ENSEMBLE ******* (MV) | 7.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.37 | 0.432 | 0.461 | 0.446 |
| *MacroTermStemEvaluator* | ⟶ | 3.419 | 0.439 | 0.468 | 0.452 |
| *MicroTermEvaluator* | ⟶ | 3.37 | 0.463 | 0.469 | 0.465 |
| *MicroTermStemEvaluator* | ⟶ | 3.419 | 0.47 | 0.476 | 0.472 |
| ******* ENSEMBLE ******* (WMV 1) | 7.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.627 | 0.463 | 0.497 | 0.479 |
| *MacroTermStemEvaluator* | ⟶ | 3.76 | 0.48 | 0.515 | 0.496 |
| *MicroTermEvaluator* | ⟶ | 3.627 | 0.495 | 0.503 | 0.498 |
| *MicroTermStemEvaluator* | ⟶ | 3.76 | 0.515 | 0.522 | 0.518 |
| ******* ENSEMBLE ******* (WMV 2) | 7.5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.559 | 0.478 | 0.487 | 0.482 |
| *MacroTermStemEvaluator* | ⟶ | 3.69 | 0.496 | 0.505 | 0.5 |
| *MicroTermEvaluator* | ⟶ | 3.559 | 0.507 | 0.493 | 0.5 |
| *MicroTermStemEvaluator* | ⟶ | 3.69 | 0.528 | 0.512 | 0.519 |

**Table 6.20:** Test 7 results - Hulth's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29,6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3,909 | 0,132 | 0,656 | 0,22 |
| *MacroTermStemEvaluator* | ⟶ | 3,975 | 0,134 | 0,667 | 0,224 |
| *MicroTermEvaluator* | ⟶ | 3,909 | 0,133 | 0,672 | 0,223 |
| *MicroTermStemEvaluator* | ⟶ | 3,975 | 0,136 | 0,683 | 0,226 |
| ******* KUSCO ******* | 12,1 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1,231 | 0,102 | 0,206 | 0,136 |
| *MacroTermStemEvaluator* | ⟶ | 1,467 | 0,122 | 0,246 | 0,163 |
| *MicroTermEvaluator* | ⟶ | 1,231 | 0,118 | 0,205 | 0,149 |
| *MicroTermStemEvaluator* | ⟶ | 1,467 | 0,14 | 0,246 | 0,178 |
| ******* CRF ******* | 4,8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2,332 | 0,483 | 0,392 | 0,433 |
| *MacroTermStemEvaluator* | ⟶ | 2,397 | 0,497 | 0,403 | 0,445 |
| *MicroTermEvaluator* | ⟶ | 2,332 | 0,491 | 0,401 | 0,441 |
| *MicroTermStemEvaluator* | ⟶ | 2,397 | 0,505 | 0,412 | 0,453 |
| ******* ENSEMBLE ******* (MV) | 6,5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2,659 | 0,413 | 0,447 | 0,429 |
| *MacroTermStemEvaluator* | ⟶ | 2,686 | 0,417 | 0,451 | 0,433 |
| *MicroTermEvaluator* | ⟶ | 2,659 | 0,428 | 0,455 | 0,44 |
| *MicroTermStemEvaluator* | ⟶ | 2,686 | 0,431 | 0,46 | 0,444 |
| ******* ENSEMBLE ******* (WMV 1) | 6,3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2,755 | 0,438 | 0,463 | 0,45 |
| *MacroTermStemEvaluator* | ⟶ | 2,851 | 0,453 | 0,479 | 0,466 |
| *MicroTermEvaluator* | ⟶ | 2,755 | 0,453 | 0,472 | 0,462 |
| *MicroTermStemEvaluator* | ⟶ | 2,851 | 0,471 | 0,489 | 0,479 |
| ******* ENSEMBLE ******* (WMV 2) | 5,9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2,663 | 0,451 | 0,447 | 0,449 |
| *MacroTermStemEvaluator* | ⟶ | 2,759 | 0,467 | 0,464 | 0,465 |
| *MicroTermEvaluator* | ⟶ | 2,663 | 0,462 | 0,456 | 0,458 |
| *MicroTermStemEvaluator* | ⟶ | 2,759 | 0,48 | 0,473 | 0,476 |

**Table 6.21:** Test 7 results - Kraps's dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average number of keywords extracted* | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|
| ******* KEA ******* | 29.9 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.169 | 0.4475 | 0.2453 |
| ******* KUSCO ******* | 13.7 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.155 | 0.27 | 0.196 |
| ******* CRF ******* | 6.5 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.7361 | 0.5743 | 0.6444 |
| ******* ENSEMBLE ******* (WMV 2) | 7.5 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.6943 | 0.6081 | 0.6478 |

**Table 6.22:** Test 8 results (Hulth's dataset). (***WMV 2***): Weighted Majority Vote version ; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

example is given in the next paragraph, demonstrating what have been said and showing that these two methods are not the most indicated to evaluate CRF's truly power.

During the experimentation phase, it was noticed that *Document* **X**, one of the documents in this dataset, had the following ***true keywords***:

1. US dollar ;
2. Exchange rate;

   Nevertheless, because both words appear following each other in the text and due to CRF inner characteristics that can use most of the features present in the documents for extraction, it happened that the ***predicted keyword*** for that document actually *contained both* of the gold standard ones, i.e.:

3. US dollar exchange rate ;

   thus, when the evaluation phase starts and the matches are accounted, it resulted that true keywords 1 and 2 were compared directly and integrally with predicted keyword 3, finding 0 matches (only full matches or full stem matches are accounted) when both keywords were in fact found, but extracted as a single one.

   In short, what we are saying is that the system found 0 keywords, when a closer look shows that both were extracted, i.e., 2 matches! This reveals a limitation inherent in the current adopted evaluation method by the community and it is severely decreasing Precision, Recall and $F_1$ scores of the

CRF system and consequently of the ensemble as well (for the other systems the problem is not observed, due to the nature of the keywords extracted by these systems and the results obtained remained practically unchanged, which seems to support the validity of the proposed evaluation method).

To mitigate this problem and to realize how it was affecting the applications, another way of comparing the results was exploited which, instead of comparing if both keywords are exactly equal, checks if the predicted keyword *contains* the true keyword (i.e., predictedKey.contains(trueKey) and not the inverse!). This way, because both true keywords (1 and 2) are contained within the predicted one (3), we will count 2 matches instead of 0, which represents a more accurate result.

Despite the problem seemed solved, another situation had been taken into account: the opposite case, when more than one predicted keyword contains a true keyword. In this case, rather than degrading the performance, it will over-inflating the results (as a matter of fact, in the example given ahead it would result in a Precision greater than 1, which is obviously impossible, as 1 is the maximum possible). To avoid that, only 1 match is accounted in these situations, i.e, only 1 match per true keyword is considered, although both of the predicted keywords could be considered correct (because they are just more specific than that given by the author).

**True keyword**:

4. boolean ;

**Predicted keywords**:

5. boolean function ;

6. boolean method;

To test this method for accounting matches between true and predicted keywords, the test (from the previous) that achieved best results was used (Test 7, Hulth's dataset) and the results are portrayed in Table 6.22. As one can see, this method only influenced significantly CRF system and consequently the ensemble, which suggests that this system is extracting keywords that are even more specific than those given as gold standard. Its worth noting that only the best results for each application and the best weighted setup used are illustrated.

| Test Number | Number of documents | Average author assigned keywords per document | Average number of words per document |
|---|---|---|---|
| 9-10 | 419 | 3.1 | 118 |

**Table 6.23:** Document's info (Events' descriptions dataset).

| Test Number | Number of documents | Average author assigned keywords per document | Average number of words per document |
|---|---|---|---|
| 9-10 | 112 | 6.8 | 85.1 |

**Table 6.24:** Document's info (Personalities' descriptions dataset).

### 6.2.2 Events' Dataset

In this section the objective was validating the results obtained with the scientific datasets previously tested, using for that both Events and Personalities datasets, where the application is being used in the future. The results obtained are very promising and, for one of these datasets (namely that of Personalities' descriptions), surpassed those achieved with the previous ones. For the other dataset (Events' descriptions) the results were not bad at all, but it was labelled by a single (not professional) assigner and the keywords lacked quality, being one of the reasons for a poorer performance when compared with the other datasets. Another reason is the number of keywords labelled, considerably less than the setups that actually produced the best results. Tables 6.23 and 6.24 portray the information about the Events' and Personalities' descriptions datasets.

It is also worth referring that the results presented in this section are divided in two:

- Test 9, using the version of the ensemble that achieved the best performance from the previous set of tests (that of Test 7) ;
- Test 10, using also the new Evaluator presented in a previous test (Test 8).

#### Test 9 - Validating the results using events and personalities' descriptions

Results displayed in Table 6.25 concerns the usage of Events' descriptions, as mentioned earlier. Those that used Personalities' descriptions are shown in Table 6.26.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 29.3 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.607 | 0.123 | 0.595 | 0.204 |
| *MacroTermStemEvaluator* | ⟶ | 3.677 | 0.125 | 0.606 | 0.208 |
| *MicroTermEvaluator* | ⟶ | 3.607 | 0.124 | 0.614 | 0.206 |
| *MicroTermStemEvaluator* | ⟶ | 3.677 | 0.126 | 0.626 | 0.21 |
| ******* KUSCO ******* | 13.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 1.566 | 0.113 | 0.258 | 0.157 |
| *MacroTermStemEvaluator* | ⟶ | 1.866 | 0.134 | 0.308 | 0.187 |
| *MicroTermEvaluator* | ⟶ | 1.566 | 0.127 | 0.297 | 0.178 |
| *MicroTermStemEvaluator* | ⟶ | 1.866 | 0.152 | 0.347 | 0.211 |
| ******* CRF ******* | 4.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.323 | 0.475 | 0.383 | 0.424 |
| *MacroTermStemEvaluator* | ⟶ | 2.422 | 0.496 | 0.399 | 0.442 |
| *MicroTermEvaluator* | ⟶ | 2.323 | 0.468 | 0.381 | 0.42 |
| *MicroTermStemEvaluator* | ⟶ | 2.422 | 0.485 | 0.397 | 0.437 |
| ******* ENSEMBLE ******* (MV) | 6.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.472 | 0.376 | 0.408 | 0.391 |
| *MacroTermStemEvaluator* | ⟶ | 2.521 | 0.383 | 0.416 | 0.399 |
| *MicroTermEvaluator* | ⟶ | 2.472 | 0.392 | 0.413 | 0.401 |
| *MicroTermStemEvaluator* | ⟶ | 2.521 | 0.399 | 0.421 | 0.409 |
| ******* ENSEMBLE ******* (WMV 1) | 6,5 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2,68 | 0,413 | 0,442 | 0,427 |
| *MacroTermStemEvaluator* | ⟶ | 2,806 | 0,433 | 0,463 | 0,447 |
| *MicroTermEvaluator* | ⟶ | 2,68 | 0,425 | 0,446 | 0,435 |
| *MicroTermStemEvaluator* | ⟶ | 2,806 | 0,447 | 0,467 | 0,456 |
| ******* ENSEMBLE ******* (WMV 2) | 6.1 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 2.609 | 0.43 | 0.43 | 0.43 |
| *MacroTermStemEvaluator* | ⟶ | 2.732 | 0.45 | 0.45 | 0.45 |
| *MicroTermEvaluator* | ⟶ | 2.609 | 0.438 | 0.434 | 0.436 |
| *MicroTermStemEvaluator* | ⟶ | 2.732 | 0.459 | 0.454 | 0.457 |

**Table 6.25:** Test 9 results - Events' descriptions dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | *A* | *P* | *R* | $F_1$ |
|---|---|---|---|---|---|
| ******* KEA ******* | 24.8 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.523 | 0.144 | 0.524 | 0.226 |
| *MacroTermStemEvaluator* | ⟶ | 3.542 | 0.145 | 0.527 | 0.227 |
| *MicroTermEvaluator* | ⟶ | 3.523 | 0.151 | 0.643 | 0.244 |
| *MicroTermStemEvaluator* | ⟶ | 3.542 | 0.152 | 0.643 | 0.245 |
| ******* KUSCO ******* | 11.4 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.011 | 0.268 | 0.445 | 0.334 |
| *MacroTermStemEvaluator* | ⟶ | 3.191 | 0.285 | 0.472 | 0.355 |
| *MicroTermEvaluator* | ⟶ | 3.011 | 0.293 | 0.426 | 0.345 |
| *MicroTermStemEvaluator* | ⟶ | 3.191 | 0.309 | 0.447 | 0.363 |
| ******* CRF ******* | 4.9 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.111 | 0.64 | 0.455 | 0.529 |
| *MacroTermStemEvaluator* | ⟶ | 3.111 | 0.64 | 0.455 | 0.529 |
| *MicroTermEvaluator* | ⟶ | 3.111 | 0.65 | 0.532 | 0.584 |
| *MicroTermStemEvaluator* | ⟶ | 3.111 | 0.65 | 0.532 | 0.584 |
| ******* ENSEMBLE ******* (MV) | 7.6 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.196 | 0.426 | 0.471 | 0.445 |
| *MacroTermStemEvaluator* | ⟶ | 3.196 | 0.426 | 0.471 | 0.445 |
| *MicroTermEvaluator* | ⟶ | 3.196 | 0.441 | 0.576 | 0.497 |
| *MicroTermStemEvaluator* | ⟶ | 3.196 | 0.441 | 0.576 | 0.497 |
| ******* ENSEMBLE ******* (WMV 1) | 7,7 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3,683 | 0,485 | 0,54 | 0,509 |
| *MacroTermStemEvaluator* | ⟶ | 3,692 | 0,486 | 0,541 | 0,51 |
| *MicroTermEvaluator* | ⟶ | 3,683 | 0,487 | 0,617 | 0,543 |
| *MicroTermStemEvaluator* | ⟶ | 3,692 | 0,49 | 0,622 | 0,547 |
| ******* ENSEMBLE ******* (WMV 2) | 5.4 | - | - | - | - |
| *MacroTermEvaluator* | ⟶ | 3.174 | 0.592 | 0.464 | 0.518 |
| *MacroTermStemEvaluator* | ⟶ | 3.174 | 0.592 | 0.464 | 0.518 |
| *MicroTermEvaluator* | ⟶ | 3.174 | 0.62 | 0.545 | 0.579 |
| *MicroTermStemEvaluator* | ⟶ | 3.174 | 0.62 | 0.545 | 0.579 |

**Table 6.26:** Test 9 results - Personalities' descriptions dataset. (***MV***): Majority Vote; (***WMV 1 & 2***): Weighted Majority Vote versions; (***A***): Average correct keys/file; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|
| ******* CRF ******* | 4.9 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.6517 | 0.4873 | 0.5574 |
| ******* ENSEMBLE ******* (WMV 2) | 6.1 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.5979 | 0.5394 | 0.5669 |

**Table 6.27:** Test 10 results (Events' descriptions dataset). (***WMV 2***): Weighted Majority Vote version; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

| *Application* | *Average keys extracted* | ***P*** | ***R*** | $F_1$ |
|---|---|---|---|---|
| ******* CRF ******* | 4.9 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.8522 | 0.6649 | 0.7449 |
| ******* ENSEMBLE ******* (WMV 2) | 5.4 | - | - | - |
| *MicroSegmentStemEvaluator* | ⟶ | 0.6306 | 0.7244 | 0.6726 |

**Table 6.28:** Test 10 results (Personalities' descriptions dataset). (***WMV 2***): Weighted Majority Vote version; (***P***): Precision ; (***R***): Recall; ($F_1$): F-measure.

#### Test 10 - MicroSegmentEvaluator revisited

Similarly to those presented in the previous section, results depicted in Table 6.27 and Table 6.28 used the events' and personalities' descriptions respectively.

Its yet worth noting that only the best results from those obtained are depicted (i.e., the best individual application and the best ensemble weighted setup).

### 6.2.3 Discussion

Several decisions were taken during the experimentation phase and, while some were already discussed in their respective test's sub-section for the sake of understanding, others did not and will be explained here.

Starting by the weights given to each classifier, two different combinations achieved best results, depending on the individual performance of CRF, as illustrated in Table 6.9. Nevertheless, attentive reader will notice that, despite KEA is achieving better $F_1$ scores than KUSCO in some of the tests shown, it is always given less vote weight. This can be explained due to the extracted keywords' type: KUSCO guarantees that each term it extracts is different from each other; with KEA, the same is not guaranteed. In fact, many of the terms extracted by KEA contain each other (e.g.: [example keyword extracted], [keyword extracted], [example keyword]), which gives an undesired focus to the *same* keyword when

the voting phase happens. To control that, the weight of a vote coming from KEA had to be lower than a vote coming from KUSCO. Yet, a question arises: why not just extracting less keywords with KEA and giving more weight to each of those? The answer is simple: because the best classifier, that of CRF, achieves more Precision (the keywords actually found) rather than Recall (the number of keywords found), the improvement that we see in the final results obtained come from enhancing this lack of keywords that the system can extract for some documents, so, the most keywords extracted by the applications, the better. Summarizing, because the keywords extracted by CRF are usually correct but are generally not many, the other two applications compensate that fact and this is one of the reasons why the ensemble achieves better results.

Following what have been said, the second configuration tested was the actual maximum number of keywords being extracted by each application that would give best results and trying to understand why. This second question was actually answered in the above paragraph already. The first question raised, concerning the maximum number of extracted keywords, the answer was *thirty*, to be precise. Note that the average of keywords extracted is actually lower than this limit for all the applications (even for KEA, although just for a small percentage), meaning that, for some documents, the applications could not extract as many keywords as desired. As referenced before, we cannot control the number of keys retrieved from KUSCO and CRF systems, those extract as many as they can.

Document structure was another factor influencing the results obtained. As one can see by the results shown, after using the OpenNLP Sentence Splitter to split the sentences of each document correctly, great improvement was observed.

Yet concerning structural issues, the number of true keywords present in the files seem to affect the performance of the applications, as well as the type (precisely in this case, if they contained digits) of keywords given as gold standard. Thus, removing unseen keywords and those which contain numbers resulted in better performances observed.

It can also be stated from the results obtained, since Test 2, that the *Simple Majority Voting* no longer improves the results of the best individual application (CRF). This can be explained due to the huge differences in the classifiers' reliability, turning some predictions way better than others. Thus being, attributing the same weight to all of them was obviously not the right thing to do. As so, giving

the appropriate weights to each of those predictions based on each classifier's individual performances, improved the results obtained as desired and expected.

Another objective for this dissertation was validating the results obtained with scientific datasets, with the non-scientific ones. From the results depicted in Table 6.25, concerning those of the Event descriptions, a performance improvement is still visible, similarly to what happened with the scientific datasets. Nevertheless, for the second non-scientific dataset here tested, that consisting in Personalities descriptions, the results using the ensemble did not improved those obtained by the CRF itself, despite the different configuration used for the Weighted Majority Vote. This can be explained because the superb performance that CRF system achieved with this dataset (depicted in Table 6.26) and indicates that above a certain limit, no further gains can be achieved: the difference between the classifiers' performances is to large, causing any keyword coming from the lower classifiers (KUSCO and KEA) being not good enough to improve the performance of the higher one (CRF).

The results presented here also seem to indicate something that had already been stated before: CRF is highly dependent on the document structure given. A closer look to each of these datasets (Events and Personalities), show that all the documents in the latter are very similar among each other, which seems to be the reason why CRF achieve even better performance in this case. These results are even more astonishing if we consider the new evaluation method here proposed. Comparing SegmentEvaluator with the other standard four Evaluators used, the results still the same: the ensemble achieves better performance than the individual applications for the Events' dataset, but it cannot surpass CRF system in the case of Personalities' dataset.

# Chapter 7

# Conclusions and Future Work

This last chapter of the dissertation consists of three sections. Section 7.1 presents concluding remarks. Section 7.2 summarizes the operation of the application here presented. Section 7.3 describes concrete ideas to be applied in the future.

## 7.1 Conclusions

The work presented in this dissertation concerns automatic keyword extraction from textual sources in general, once it was successfully applied to scientific and non-scientific domains.

The approach taken was that of supervised machine learning, that is, prediction models were obtained by training a priori the applications with pre-labelled documents and using an ensemble learning method to improve results, a method being recently focused by researchers, showing to be very effective in improving the results of single classifiers.

For combine the results of the individual models, two methods of majority voting are used: simple majority and weighted majority. While the former gives equal weight to all predictive models, the latter gives more weight to those who present better predictive performance.

The performance of the individual and ensemble applications is evaluated by the traditional metrics of Precision, Recall and F-measure, the latter being a combination of the formers. Equal weight is given to both Precision and Recall to compute the F-measure. Additionally, micro and macro-averaged versions of the results are also presented.

This dissertation has shown that combining models of different existing applications, instead of using a more traditional method (like bagging or boosting) to generate different models of an algorithm, is also a viable method to create an

ensemble application and the empirical results here obtained, which improved those of each the individual systems, attest that.

## 7.2 Summary

The keyword extraction application here proposed is developed for the English language, but the same principles should be valid for other languages as well. It was not however the focus of this dissertation to test that.

The main phases identified in the application's operation are listed below:

- Preprocessing the input;
- Training the models of the individual applications;
- Applying the gained knowledge from the models for extracting keywords from new documents;
- Combining the results of the individual applications using different majority voting schemes.

## 7.3 Future Work

During the development of this work, some ideas that could not be tested due to time constraints and should lead to performance improvements became evident.

- The preprocessing plays a crucial role in this kind of work, being of great importance for a more efficient operation by the applications. Thus, it is worth keep improving this part with new ideas that might enhance the results;
- Document structure and type clearly influenced the final output. Knowing this, an idea worth to be tested is applying a clustering algorithm that can find and group the documents that share common characteristics among them, training different models for each cluster found. That way, the obtained classifiers will become *specialists* in that type of document, possibly minimizing the labelling error and resulting in better performance;
- For this work, the applications were used as we got them. However, the CRF application can be further improved with new unused features. So, it may also be worth improving the CRF itself, adding new features that were

not present yet, but were considered important according to the empirical studies made by other researchers, like DEP (position of first appearance of the word) feature. Improving the CRF is then one of the paths to be taken in the future and very likely to improve the results as well;

- The great discrepancy between the individual applications' performance seems also to be a limiting factor to the results obtained. As observed, the ensemble gains were considerably higher when the individual performances were more alike (E.g. Test 1) and smaller gains were observed when one application achieved considerably better results than the others (Test 8). Another idea that can boost the application results is to use systems that share a closer performance. Three ideas can than be further explored: 1) adding one or more CRF models to the current application; 2) using a combination of CRF models and other existing algorithms that have better predictive performance than those used here; 3) using an ensemble of pure CRF models (the application that obtained the best individual results and is currently the state-of-the-art labelling method); each of these three approaches should also result in improvements.

# Bibliography


[Ali95] Kamal M. Ali. A comparison of methods for learning and combining evidence from multiple models, 1995.

[Alv11] Ana Alves. *Semantic Enrichment of Places - Understanding the Meaning of Public Places from Natural Language Texts*. PhD thesis, Faculty of Sciences and Technology of the University of Coimbra, Coimbra, Portugal, 2011.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[Coh95] Jonathan D. Cohen. Highlights: Language- and domain-independent automatic indexing terms for abstracting. *Journal of the American Society for Information Science*, page 114, 1995.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley-Interscience publication. John Wiley & Sons, 1973.

[Dom00] Pedro Domingos. Bayesian averaging of classifiers and the overfitting problem. In *IN PROC. 17TH INTERNATIONAL CONF. ON MACHINE LEARNING*, pages 223–230. Morgan Kaufmann, 2000.

[DP97] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Mach. Learn.*, 29(2-3):103–130, November 1997.

[FPW+99] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-specific keyphrase extraction. In *IJCAI*, pages 668–673, 1999.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm, 1996.

[FYZ12] Hong-Wei Xuan Feng Yu and De-Quan Zheng. Key-phrase extraction based on a combination of crf model with document structure. *2012 Eighth International Conference on Computational Intelligence and Security*, 0:406–410, 2012.

[GGL09] Maria P. Grineva, Maxim N. Grinev, and Dmitry Lizorkin. Extracting key terms from noisy and multitheme documents. In *WWW*, pages 661–670, 2009.

[Gia05] Michael J. Giarlo. A comparative analysis of keyword extraction techniques, 2005.

[Gup10] Jasmeen Kaur; Vishal Gupta. Effective approaches for extraction of keywords. *International Journal of Computer Science Issues*, 7:144–148, 2010.

[GW07] G.R. Guile and Wenjia Wang. Enhancing boosting by feature non-replacement for microarray data analysis. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 430 –435, aug. 2007.

[HKGM05] Yaakov HaCohen-Kerner, Zuriel Gross, and Asaf Masa. Automatic extraction and learning of keyphrases from scientific articles. In *Proceedings of the 6th international conference on Computational Linguistics and Intelligent Text Processing*, CICLing'05, pages 657–669, Berlin, Heidelberg, 2005. Springer-Verlag.

[Hul03] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proc. of the 2003 Conf. on Empirical Methods in NLP*, pages 216–223, 2003.

[Hul04a] A. Hulth. *Combining Machine Learning and Natural Language Processing for Automatic Keyword Extraction*. Report series / Department of Computer & Systems Sciences. Department of Computer and Systems Sciences [Institutionen för Data- och systemvetenskap], Univ., 2004.

[Hul04b] Anette Hulth. Enhancing linguistically oriented automatic keyword extraction. In *Proceedings of HLT-NAACL 2004: Short Papers*, HLT-NAACL-Short '04, pages 17–20, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[JHL09] Xin Jiang, Yunhua Hu, and Hang Li. A ranking approach to keyphrase extraction. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 756–757, New York, NY, USA, 2009. ACM.

[Jon72] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[KAM08] Mikalai Krapivin, Aliaksandr Autayeu, and Maurizio Marchese. Large dataset for keyphrases extraction. Technical Report DISI-09-055, DISI, Trento, Italy, May 2008. http://eprints.biblio.unitn.it/archive/00001671/01/disi09055-krapivin-autayeu-marchese.pdf.

[KR13] L. I. Kuncheva and J. J. Rodríguez. A weighted voting framework for classifiers ensembles. *Knowledge and Information Systems*, 2013. (in press).

[Kun04] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[KZ11] H. H. Kian and M. Zahedi. An efficient approach for keyword selection; improving accessibility of web contents by general search engines, 2011.

[LCZS11] Zhiyuan Liu, Xinxiong Chen, Yabin Zheng, and Maosong Sun. Automatic keyphrase extraction by bridging vocabulary gap, 2011.

[LD59] H.P. Luhn and International Business Machines Corporation. Advanced Systems Development Division. *Keyword-in-context index for technical literature (KWIC index)*. ASDD report. International Business Machines Corp., Advanced Systems Development Division, 1959.

[Lea78] E.E. Leamer. *Specification searches: ad hoc inference with nonexperimental data*. Wiley series in probability and mathematical statistics. Wiley, 1978.

[Liu09] B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer, 1st ed. 2007. corr. 2nd printing edition 2009, 2007-2009.

[LMP01] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[Lov68] Julie B. Lovins. Development of a stemming algorithm. June 1968.

[LPLL09] Feifan Liu, Deana Pennell, Fei Liu, and Yang Liu. Unsupervised approaches for automatic keyword extraction using meeting transcripts. In *HLT-NAACL*, pages 620–628, 2009.

[Luh58] H. P. Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, April 1958.

[MCSM11] Kristine Monteith, James L. Carroll, Kevin D. Seppi, and Tony R. Martinez. Turning bayesian model averaging into bayesian model combination. In *IJCNN*, pages 2657–2663. IEEE, 2011.

[Mel05] Prem Noel Melville. *Creating diverse ensemble classifiers to reduce supervision*. PhD thesis, Austin, TX, USA, 2005. AAI3217133.

[MFW09] Olena Medelyan, Eibe Frank, and Ian H. Witten. Human-competitive tagging using automatic keyphrase extraction. In *EMNLP*, pages 1318–1327, 2009.

[MI04] Yutaka Matsuo and Mitsuru Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–169, 2004.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MT04] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *EMNLP*, pages 404–411, 2004.

[MW06] Olena Medelyan and Ian H. Witten. Thesaurus based automatic keyphrase indexing. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '06, pages 296–297, New York, NY, USA, 2006. ACM.

[MW08] Olena Medelyan and Ian H. Witten. Domain-independent automatic keyphrase indexing with small training sets. *J. Am. Soc. Inf. Sci. Technol.*, 59(7):1026–1040, May 2008.

[NyK07] Thuy Dung Nguyen and Min yen Kan. Keyphrase extraction in scientific publications. In *In Proc. of International Conference on Asian Digital Libraries (ICADL ?07*, pages 317–326. Springer, 2007.

[Oel09] Iryna Oelze. Automatic keyword extraction for database search, 2009.

[OPTJS10] Roberto Ortiz, David Pinto, Mireya Tovar, and Héctor Jiménez-Salazar. Buap: An unsupervised approach to automatic keyphrase extraction from scientific articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 174–177, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[Oza00] Nikunj C. Oza. Online ensemble learning. In *AAAI/IAAI*, page 1109, 2000.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1999.

[PM06] Fuchun Peng and Andrew McCallum. Information extraction from research papers using conditional random fields. *Inf. Process. Manage.*, 42(4):963–979, July 2006.

[Por11] SemEval Portal. `http://aclweb.org/aclwiki/index.php?title=SemEval_Portal`, 2011. In ACLwiki.

[RHM97] Adrian E. Raftery, Jennifer A. Hoeting, and David Madigan. Bayesian model averaging for linear regression models. *Journal of the American Statistical Association*, 92:179–191, 1997.

[RMV94] Adrian Raftery, David Madigan, and Chris T. Volinsky. Accounting for model uncertainty in survival analysis improves predictive performance. In *In Bayesian Statistics 5*, pages 323–349. University Press, 1994.

[Rob04] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60:2004, 2004.

[Sew11] Martin Sewell. Ensemble learning. Research Note, January 2011.

[SM10] Charles Sutton and Andrew McCallum. An introduction to conditional random fields, 2010. cite arxiv:1011.4088Comment: 90 pages.

[SNG10] Kamal Sarkar, Mita Nasipuri, and Suranjan Ghose. A new approach to keyphrase extraction using neural networks. *CoRR*, abs/1004.3274, 2010.

[SP03] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 134–141, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[SR10] Catarina Silva and Bernardete Ribeiro. *Inductive Inference for Large Scale Text Classification: Kernel Approaches and Techniques*, volume 255 of *Studies in Computational Intelligence*. Springer, 2010.

[SRC10] Nick Cramer Stuart Rose, Dave Engel and Wendy Cowley. *Automatic keyword extraction from individual documents*. John Wiley & Sons, Ltd, 2010.

[SY73] Gerard Salton and Chu-Sing Yang. On the Specification of Term Values in Automatic Indexing. *Journal of Documentation*, 29:351–372, 1973.

[TKV10] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *In Data Mining and Knowledge Discovery Handbook*, pages 667–685, 2010.

[Tur99a] Peter Turney. Learning to extract keyphrases from text, 1999.

[Tur99b] Peter D. Turney. Learning algorithms for keyphrase extraction. *INFORMATION RETRIEVAL*, 2:303–336, 1999.

[TW99] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[vdPPRG04] Lonneke van der Plas, Vincenzo Pallotta, Martin Rajman, and Hatem Ghorbel. Automatic keyword extraction from spoken text. a com-

parison of two lexical resources: the edr and wordnet. *CoRR*, cs.CL/0410062, 2004.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[Wol92] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[WPF+99] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. Kea: practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries*, DL '99, pages 254–255, New York, NY, USA, 1999. ACM.

[WPHZ06] Jiabing Wang, Hong Peng, Jing-song Hu, and Jun Zhang. Ensemble learning for keyphrases extraction from scientific document. In *Proceedings of the Third international conference on Advances in Neural Networks - Volume Part I*, ISNN'06, pages 1267–1272, Berlin, Heidelberg, 2006. Springer-Verlag.

[WX08] Jun Wan and Jianguo Xiao. Single document keyphrase extraction using neighborhood knowledge. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 855–860. AAAI Press, 2008.

[XKS92] L. Xu, A. Krzyzak, and C. Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):418–435, May 1992.

[Yan99] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, 1999.

[YL95] Madigan D. Heuch I. York, J. and R.T. Lie. Estimation of the proportion of congenital malformations using double sampling: Incorporating covariates and accounting for model uncertainty. In *Applied Statistics*, volume 44, page 227?242, 1995.

[Zha08] Chengzhi Zhang. Automatic keyword extraction from documents using conditional random fields, 2008.

[Zha09] Chengzhi Zhang. Combining statistical machine learning models to extract keywords from chinese documents. In Ronghuai Huang, Qiang Yang, Jian Pei, João Gama, Xiaofeng Meng, and Xue Li, editors, *Advanced Data Mining and Applications*, volume 5678 of *Lecture Notes in Computer Science*, pages 745–754. Springer Berlin Heidelberg, 2009.