Barrier and Syntactic Features for Information Retrieval

Anita Alicante

March 2013

Dottorato in Scienze Computazionali e Informatiche XXV Ciclo

Universitá degli studi di Napoli “Federico II”

Tesi di Dottorato

Barrier and Syntactic Features for Information Retrieval

by

Anita Alicante

Barrier and Syntactic Features for Information Retrieval

Anita Alicante

March 2013

Abstract

Information Retrieval (IR) goal consists in *retrieving* all the *documents* in a collection that are relevant to a *given query*. A subtask of IR is *Information Extraction* (IE) which includes machine learning approaches automatically *extract* from the documents *information* about, for example, *entities* or *relations* or *events* etc. In this thesis a novel type of features, called *barrier features*, is introduced. They are based on PoS-tagging. We use these features to solve several IR and IE problems. In details we build several IR or IE systems and overcame both the state-of-art methods and baseline systems built without these features. Again exploiting syntactic information in the second part of this thesis we apply constituency and dependency parsing, to two different areas: to support *Concept Location* in Software Engineering and to study the *influence of the constituent order on the data-driven parsing* in Computational Linguistic. In the former we have evaluated the use of off-the-shelf and trained natural language analyzers to parse identifier names, extract an ontology and use it to support concept location; in the latter we use two state-of-the-art data-driven parsers to study influence of the constituent order on the data-driven parsing of Italian.

*"**The future belongs to those who believe in the beauty of their dreams.**"*

"Il futuro appartiene a coloro che credono nella bellezza dei propri sogni."

(Eleanor Roosevelt)

Con profondo affetto a Mio Papà e Mia Nonna, i miei avvocati in cielo,

e

a Mia Mamma e Mia Sorella, i miei angeli in terra!!!!

Publications

Part of the material in this thesis was published in the following articles:
Anita Alicante and Anna Corazza. **Barrier features for classification of semantic relations**. In Proceedings of the 8th Recent Advances in Natural Language Processing (RANLP 2011), pages 509-514, Hissar, Bulgaria, September. Association for Computational Linguistics.

Anita Alicante, Cristina Bosco, Anna Corazza and Alberto Lavelli. 2012. **A Treebank-based study on the influence of Italian word order on parsing performance.** In Proceedings the 8th Language Resources and Evaluation (LREC 2012), Istanbul, Turchia.

Surafel Lemma Abebe, Anita Alicante, Anna Corazza and Paolo Tonella. **Supporting Concept Location through Identifier Parsing and Ontology Extraction**. Submitted to Journal of Systems and Software

Contents

Chapter 1

Introduction

1.1 Background and motivations

Information Extraction (IE) is a subtask of Information Retrieval (IR) which extracts structured information from the unstructured or structured texts. For instance several applications involving text processing require to automatically extract from the text a set of entities or to build semantic relations between them. In Figure 1.1 some instances of entities and relations, informations which we could extract in an automatic way, are reported. Nowadays many approaches of text processing are develop to solve IE task,

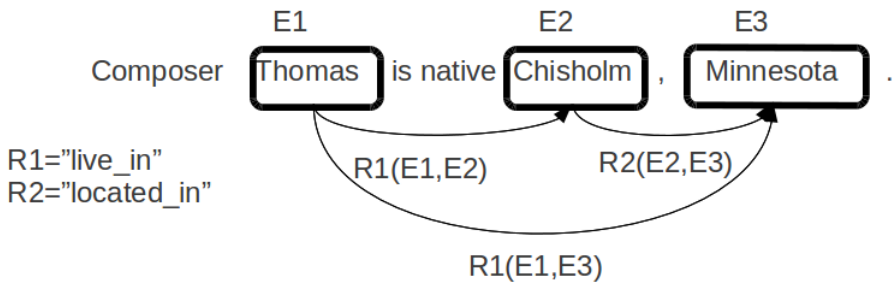

Figure 1.1: A sentence extracted by Roth and Yih annotated data set. $E1$, $E2$ and $E3$ are entities. $R1$ and $R2$ are respectively *"*$live\ in$*"* and *"*$located\ in$*"* relations.

extracting automatically information from the texts [11]. Such methods are based on machine learning techniques in which the input data (entity, tweets, relation, sentence etc.) are represented using *features*. The type of a feature is strictly connected with its value. In literature many types of features exist and the following ones are the most famous:

- *Natural features* have integer values corresponding, for example, to counts of occurrences of a pattern;

- *Real features* have values computed, for example, using the standard weighting schema called *tf/idf* (term frequency/inverse document frequency) [71];
- *Boolean features* assume only two values corresponding to the occurrence/absence, for instance, of words, full stop, comma and capitalized text etc.

Features describe the main characteristics of a input element and therefore the design of the set of features to include in the input representation is crucial in a machine learning approach.

Nearly all feature sets used to represent text in some machine learning task include unigrams of words. In addition to that, very often also bigrams are included, trying to capture some simple expressions involving the local context of each word. To include semantic information into this representation, we usually need manually developed lexical resources, such as the English Dictionary of Affect in Language [112]. Another possibility is to consider syntactic rather than semantic information, such as the parse tree of the sentence [2]. However, in case of informal language such as that used in tweets, it is very difficult to automatically obtain a complete parse tree, because sentences are often ungrammatical and errors are frequent.

In this thesis we define a new type of Boolean features, namely *Barrier Features* (BFs) based only on PoS Tagging, which describe a syntactic link between two tokens in a sentence based their Part-of-Speech (PoS) tags [35]. Moreover we use BFs in several machine learning approaches which automatically extract different kinds of information: entities, relations and sentiment expressed in a sentence.

In other machine learning approaches other syntactic information, as dependencies extracted with a dependency parsers, are employed to find the parts of document which describe a concept. For example in this thesis we use these dependencies to support a Software engineering task, namely *Concept Location*. On the other hand we exploit the syntactic analysis (constituency and dependency) to study the *influence of constituents* in a *Morphologically Rich Language*, as Italian.

1.2 Contributions of this Thesis

The main contribution of this thesis is the introduction of *Barrier Features* (BFs) which characterize a syntactic binding between two tokens in a phrase or sentence. Although BFs are based on PoS Tagging [35], a Natural Language Processing (NLP) task which assigns a lexical category (PoS tags) to each token in a sentence, BFs also seem likely to

be useful to capture the semantic bindings between tokens in a sequence. In fact their introduction in different system is very effective to extract, for instance, semantic relations between two entities contained in a sentence or to classify if a sentiment expressed in a tweet (message posted on famous social network Twitter) is positive or negative. Moreover another relevant contribution of this thesis is that we exploit the BFs in different machine learning approaches to solve several IE and IR tasks, such as entity and relation classification, relation extraction and the Twitter sentiment polarity classification.

Furthermore since BFs are Boolean features we propose several smoothing strategies to overcome data sparsity with unsupervised construction of a dictionary in the feature extraction phase or with the introduction of a special features named UNKNOWN and so on [54].

In the entity classification and relation extraction we give another one of our main contributions building a new probabilistic integration method based on Graphical Model which merge two information sources: the output of more usual classifier of entities and relations and logical constraints defined in a *Knowledge Base* (KB). To demonstrate the effectiveness of this new model we compare two systems: the *Pipeline System (PipeLS)* and *Jointly Entity and Relation Extraction System (JERES)*. The first one is composed by two modules: the entity classifier and the relation classifier combined in a pipeline and the output of entity classifier is the input of relation classifier. *JERES* build a probabilistic integration model based on the Graphical Model in which the output of entity and relation classifiers are integrated together with the logical constraints of a KB.

In the second part of this thesis we use other *syntactic information*, different from BFs, and based on syntactic analysis. In fact we propose to exploit some syntactic analyzers, the constituency and dependency parsing, in Software Engineering (SE) and Computational Linguistic (CL) tasks.

In details in the SE field the information captured in program identifiers can be extracted and used to support program comprehension by resorting to natural language parsing. More specifically, dependency parsers can determine the dependency relations that hold between the terms an identifier is composed of. Our contribution is based on training a natural language dependency parser to work directly on an identifier term list and we investigate the effect of training on concept extraction. We proposed four types of natural language analyzers to parse identifiers of a system. Two of the analyzers are adapted to directly work on identifiers through training while the other two are standard English analyzers. The training of the analyzers is conducted automatically using a training set constructed from the documentation of the corresponding system. Specifically, we evaluate such effect when the extracted concepts are used to facilitate the execution of a concept

location task. The concepts automatically extracted from the dependencies produced by the parser are represented in an ontology.

In CL we study the influence of the constituent order on the data-driven parsing of one of Italian, Morphologically Rich Language, using state-of-the-art data-driven parsers. The experiments are based on an Italian treebank, available in formats that vary according to two dimensions, i.e. the paradigm of representation (dependency vs. constituency).

1.3 Overview

This thesis is organized in seven chapters. Each one is meant as much as possible to be self-contained, thus providing an introduction to the problem, background and motivation, the proposed approach, the experimental assessment or case studies, conclusions and future works.

In *Chapter* 2 we report the definition of *Barrier Features* [3], a novel type of features which defines a link between two tokens in a phrase or sentence. In Chapters 3, 4 and 5 we applied the BFs in three different tasks to evaluate their effectiveness. Moreover in *Chapter* 3 BFs are use to train a classifier of semantic relations. We exploit these features in addition with other usual boolean features such as n-grams of PoS, word suffixes and prefixes, hypernyms from WordNet etc. In this Chapter we classify only semantic relation while the correct entities are given in input.

Furthermore in *Chapter* 4 we employed the same classifier of *Chapter* 3 to classify both the entities and relations. Using these classifiers we build two different systems: the *Pipeline System (PipeLS)* and J*ointly Entity and Relation Extraction System (JERES)*. The innovation of this Chapter is the second one system based on the Graphical Models.

In the *Chapter* 5 an approach for Twitter Sentiment Analysis is proposed based on a Maximum Entropy classifier trained with unigrams, bigrams and BFs extracted by a data set which contains tweets, messages posted on the social network Twitter. The system with BFs outperform other state-of-art methods demonstrating experimentally that these features, based on PoS tagging, are able to distinguish also semantic information to classify, as negative or positive sentiments expressed in a tweet [1].

In *Chapter* 6 we have evaluated the use of off-the-shelf and trained natural language analyzers to parse identifier names. As syntactic analyzer we employ a dependency state-of-the-art data-driven parsers. Moreover we use the natural language dependencies to

[1] Messages posted on social network Twitter.

extract different ontologies, one for each analyzers, and exploit it to support *Concept Location* which uses queries to narrow down the search space and identify the parts of a program that implement a concept of interest.

In *Chapter* 7 we study the influence of the constituent order on the data-driven parsing of one of Italian using state-of-the-art data-driven parsers.

Finally in *Chapter* 8 we discuss the goals achieved in this work and open questions to be addressed in the future.

Chapter 2

Barrier Features

An important original contribution of this Chapter is the definition of *Barrier Features* (BFs), presented for the first time in [3]. *Barrier Features* (BFs) are inspired by the *barrier rules* of the *constraint grammar framework* proposed in [56] for Part-of-Speech (PoS) tagging, but completely redesigned as Boolean features rather than rules. The basic idea of BFs is that to find a syntactic binding between two tokens of a sequence.

This chapter is devoted to BFs, and presents the main idea which inspired them. Indeed the background of BFs is reviewed and eventually we summarizes the IE and IR tasks in which the BFs are applied in this thesis.

2.1 Background of Barrier Features

The *barrier rules* of the *constraint grammar framework* were used in the parser developed by [56] to improve the PoS tagging, the first step of syntactic analysis. We use the PoS tag set of Penn Treebank [73] in this thesis. A list of PoS Tags contained in this set is shown in the appendix A.

A PoS tagger assigns a lexical category to each token: for example given a token w="*defect*" a PoS Tagger can label w with tag VB if w is a verb or with tag NN if w is noun. Obviously this choice strongly depends on the context of the token w. In the [56] a list of suitable PoS tags has been associated to each token (e.g. for w="*defect*" the list was $[VB,\ NN]$). Afterwards the *barrier rules* deleted all tags which can not occur to the context of the token. For instance if the PoS tagger associated to w both VB and NN tags, a barrier rule deleted the tag VB if there was a determiner (DT) in the context of w. A token context is usually determined by a window surrounding the token. In our definition of BFs we use

this idea of context based on a window surrounding the token. In fact the words around a token are strictly connected with it and indeed they determine the meaning of such token. Therefore the *barrier rules* were employed to solve some problems of ambiguity of the PoS tagger and they were defined between some pairs of PoS tags. In the definition of our BFs, presented in Section 2.2 we use the same pairs reported in Table 2.1.

In our work we redesigned the *barrier rules* as features. In general a feature is a measurable and important attribute which characterizes an input element of the problem to solve. A feature is usually represented by a pair (name, value). Its value can be numeric or structured (e.g. *string*, *set* etc.). For example in a Natural Language Processing application, given the sentence s "*The spy , high-ranking official Korean CIA Sohn Young wanted to defect*" in input we can infer that the sequence of two adjacent tokens $< Korean\ CIA >$ is *bigram* and then a feature of s is the pair *(bigram,* $< Korean\ CIA >$*)*.

To solve IE or IR tasks many machine learning approaches transform the input data of those problems into some new space of variables. This transformation is very important to make the problem easier to solve and it is a *pre-processing* step, namely *feature extraction*. For example sometimes the input data are represented in Vector Space Model (VSM) using vectors, called *feature vectors*. $\mathtt{x}_i \in \mathcal{T}$ is the $i-th$ element of the input data set $\mathcal{T}$ and $\vec{\mathtt{f}}_i$ is a vector of size m which describes $\mathtt{x}_i$.

$$\vec{\mathtt{f}}_i = \{f_{i,1}, f_{i,2}, ...f_{i,m}\}$$

is $\vec{\mathtt{f}}_i \in F \subseteq \mathbb{R}^m$ and each component $f_{i,j}$ is a *feature*. Each feature of vector $\vec{\mathtt{f}}_i$ describe an property (attribute) of the element $\mathtt{x}_i$. These feature vectors contains very significant patterns (features) which characterize the input elements [11]. In some approaches, for example useful to solve the text categorization task, the input data are represented by the vector which contains the occurrence of the features (n-grams of PoS, word suffixes and prefixes, hypernyms from WordNet etc.). In other methods the features are Boolean, meaning that for each input element, their value is *true* if the feature occurs in the sentence, *false* otherwise.

2.2 Definition of Barrier Features

Nearly all feature sets used to represent text in some machine learning task include unigrams and bigrams of words trying to capture some simple expressions involving the local context of each token. To include semantic information into this representation, we usually need manually developed lexical resources, such as thesaurus or dictionary. Another

possibility is to consider syntactic rather than semantic information, such as the parse tree of the sentence [2]. However it is not always possible to automatically compute a complete parse tree, for example in the informal language of tweets, because sentences are often ungrammatical and errors are frequent.

We therefore need some kind of features able to capture information on the structure of the sentence without imposing constraints on its grammaticality. BFs are based on hypothesis that a syntactic binding exists between two PoS tags ad this link is represented by the set of PoS's occurring between them. They are Boolean features and then their occurrence is as important as their absence; in fact their value is *true* if the BF occurs in the sequence, *false* otherwise. Given the following sequence σ of pairs $(token,\ PoS\ tag)$

$$\sigma =< (w_1, t_1) \dots (w_e, t_e) \dots (w_t, t_t) \dots (w_n, t_n) >$$

a BF of σ can be defined as a pair β :

$$\beta = ((t_t, t_e), \delta)$$

where t_t and t_e are PoS tags respectively of tokens w_t and w_e; the former is called *trigger* while the latter *endpoint*; δ is the BF value, which is the set of PoS's occurring between the trigger and endpoint. The choice of the pairs (t_t, t_e) has been inspired by the corresponding barrier rules of the constraint grammar framework as described in Section 2.1. Such (t_t, t_e) pairs are predefined and depend on the considered language and we reported all of them in Table 2.1. Actually, while this is the only knowledge based part in BF extraction, the tag pairs introduced for English are quite intuitive. For these reason we think that the porting of BFs in language different to English would be not too difficult.

Endpoint	**Trigger**
DT	NN or NNP
DT or MD or VB or VBP or VBZ or TO	VBD, VBN
IN	VB, VBP
PRP	NNS or VBZ
JJ	JJR or RBR

Table 2.1: *Endpoints and triggers of the barrier features.*

The definition of BFs is based on the set of PoS tags in a window surrounding a token (*trigger*) belonging to the sequence. The length of the window varies and is based on the PoS's of the corresponding tokens: for each token in the sequence (*trigger*), an *endpoint* token is chosen on the basis of the PoS of the trigger. In fact, for each token in the

considered string the corresponding endpoint is defined as the closest preceding token having one of the PoS associated with the PoS of the considered token.

In the our experiments all BFs only consider an endpoint token *preceding* the entity and therefore are extracted by just considering the left context of the considered token. However, this is not the only possible case, and the right context could also be included. Whenever we find an endpoint, we introduce a new BF corresponding to the set of PoS between the endpoint and this token. An token of a sequence can have several endpoints and a new BF corresponding to the set of PoS's between the endpoint and the token is introduced for every possible endpoint. If no endpoint is found before the trigger token, the set of all the PoS tags from the beginning of the sequence to this token are considered. As the BFs are based on *sets* of tags, order and possible repetitions of tags are not considered.

For the sake of clearness, Table 2.2 reports the BFs extracted from a entity of a sentence contained in the data set annotated by Roth and Yih for entity and relation classification (see Section 3.3.1 for more details about this data set) in experimental assessment of Chapter 3 and 4. Moreover Table 2.3 shows other BFs extracted from a tweet of Stanford Twitter Sentiment (STS) data set (see Section 5.3.1 for more details about this data set) exploited in experimental assessment of Chapter 5 for sentiment polarity classification.

(DT, NN, {})	The spy
(DT, NNP, { „, JJ, NN})	The spy , high-ranking Korean CIA
(DT, NN, { „, JJ, NN, NNP})	The spy , high-ranking Korean CIA official
(DT, NNP, { „, JJ, NN, NNP})	The spy , high-ranking Korean CIA official Sohn
(DT, NNP, { „, JJ, NN, NNP})	The spy , high-ranking Korean CIA official Sohn Ho
(DT, NNP, { „, JJ, NN, NNP})	The spy , high-ranking Korean CIA official Sohn Young

Table 2.2: *BFs extracted from th following PoS tagged entity: The/DT spy/NN COMMA/, high-ranking/JJ Korean/JJ CIA/NNP official/NN Sohn/NNP Ho/NNP Young/NNP*

In semantic relation classification and jointly entity and relation extraction system, described respectively in Chapter 3 and 4, we use only some pairs (t_t, t_e) of BFs since the syntactic context of tokens in entities involved in a relation (see example in in Tables 2.2 and 3.1), can only be nouns or adjectives. Therefore, we only considered patterns for this PoS, while completely disregarding other important PoS tags including verbs.

While in our approach for Twitter sentiment analysis described in Chapter 5 we use all pairs (t_t, t_e) reported in Table 2.1 (as we can observe analyzing the example shown in 2.3) probably because the verbs are very relevant to classify the sentiments expressed in tweets, messages posted on Twitter social network.

Finally BFs require PoS tagging of the considered texts, which can be automatically performed with very high accuracy [35].

(IN, VBP, {PRP, RB})	I firmly believe
(DT, NNP, {IN, PRP, RB, VBP})	I firmly believe that Obama
(DT, NN, {IN, NNP, PRP, RB, VBP})	I firmly believe that Obama /
(DT, NNP ,{IN, NN, NNP, PRP, RB, VBP,}	I firmly believe that Obama / Pelosi
(IN, VBP, {NN, NNP})	that Obama / Pelosi have
(DT, NNP, {IN, NN, NNP, PRP, RB, VBP}	I firmly believe that Obama / Pelosi ZERO
(DT, NN, {IN, NN, NNP, PRP, RB, VBP}	I firmly believe that Obama / Pelosi ZERO desire
(DT, VB, {IN, NN, NNP, PRP, RB, TO, VBP}	I firmly believe that Obama / Pelosi ZERO desire to be
(IN, VB, {NN, NNP, TO, VBP}	that Obama / Pelosi ZERO desire to be
(PRP, VBZ, {”}	It ’ s
(DT, NN, {}	a charade
(DT, NN, {}	a slogan
(IN, VBP, {”, „ CC, DT, NN, PRP, VBZ}	It ’ s a charade and a slogan , but they want
(DT, VB, {CC, NN, PRP, TO, VBP}	a slogan , but they want to destroy
(IN, VB, {”, „ CC, DT, NN, PRP, TO, VBP, VBZ}	It ’ s a charade and a slogan , but they want to destroy
(DT, NN, {„ CC, NN, PRP, TO, VB, VBP}	a slogan , but they want to destroy conservatism

Table 2.3: *BFs extracted from the following PoS tagged tweet: USER/deleted I/PRP firmly/RB believe/VBP that/IN Obama/NNP //NN Pelosi/NNP have/VBP ZERO/NNP desire/NN to/TO be/VB civil/JJ ./. It/PRP ’/” s/VBZ a/DT charade/NN and/CC a/DT slogan/NN ,/, but/CC they/PRP want/VBP to/TO destroy/VB conservatism/NN.*

2.3 Data Sparsity

A very large number of different BFs can occur therefore the choice of the smoothing strategy is crucial. In this section we illustrate the different smoothing steps which we apply to mitigate this problem of data sparsity in our proposed systems.

First of all, as the number of potential BFs is extremely large, it is very important to reduce their number to the ones which are possible in natural language. This is particularly the case because the BFs are Boolean and both their occurrence and their absence should be registered. Unlikely real value features, Boolean ones do not assign a weight to the event, but only register whether it occurs or not. Therefore, it is important to distinguish between features which are impossible in natural language from possible features which do not occur in the input we are considering. Indeed, only the second case should correspond to a null value for the feature, while the former should be simply not considered. For example, among all possible PoS bigrams, we will only consider some PoS pairs, while disregarding all the others as impossible or too rare to be significant for any classification task. On the other hand, for some of the considered types including PoS trigrams and BFs the number of potential features can be very large.

Therefore, if the BF we observe in the input to the classifier (as effectively happens in the tasks described in Chapter 3, 4 and 5) does not exist in the set of features collected on the training set, then we consider all BFs having the same (trigger PoS, endpoint PoS) pair and a PoS’s set including the considered one. A side effect of this strategy is that more

than one BF can be "true" at the same time. We verify experimentally in different tasks, reported in the following Chapters, that this choice based on the inclusion of the PoS's set, value of BF, is a smoothing strategy more effective to an exact matching between all BFs having the same (trigger PoS,endpoint PoS) pair.

2.3.1 Unsupervised Construction of Dictionary

To mitigate data sparsity it is therefore particularly important to include in the set of considered features all and only the ones which are both possible and not too rare. In other words, we are splitting the training step in two phases: *feature extraction* and *classifier training*. While the latter requires that the data are annotated with the correct class, the former only considers the characteristics of the considered language, in our case English, and therefore can be performed on text only labeled with PoS tags. On the other hand, feature extraction considers a very large number of potential features, and therefore require a very large data set. As a very accurate PoS tagging can be obtained automatically and is therefore quite cheap, the two training phases we are considering separately can be based on different data sets. For feature extraction a large collection of English texts automatically labeled with PoS tags. We can build this dictionary in off-line mode and in unsupervised manner. Afterward, we extract the features defined in the first step from the training and the test sets corresponding to the specific task we are considering and we use these data to respectively build and assess the classifier.

As discussed above we define the set of features on a large collection of English texts and we therefore assume that all features we consider are characteristic of the English language. However, we train the classifier on a set of sentences including one labeled relation, and such training set is likely to be much smaller. We therefore need an another smoothing strategy to deal with features having a small number of occurrences in the training set. We considered the smoothing strategy based on the introduction of an UNKNOWN label [54] for each type of feature: for example, an UNKNOWN bigram is considered in addition to all other bigrams. All occurrences of features appearing in the training set a number of times smaller than a given threshold increment the UNKNOWN counter. In other words, the UNKNOWN feature has been trained by considering as UNKNOWN all features that have a number of occurrences lower than a given threshold. Again in the case of bigrams, all bigrams occurring a number of times lower than that threshold value are collected in the same class UNKNOWN, and their occurrences contribute to the UNKNOWN statistics. To classify entity and relation, as describe in Chapter 3 and 4 three different thresholds have been considered: 3 for BFs, 100 for hyperonims

and 1000 for all other Boolean features: these values has been chosen with 10-fold cross-validation on the training set of all data sets. While in the sentiment classification (see Chapter 5) expressed in some tweets we adopted as threshold $K = 50$ only for BFs. In the last approach we chose the threshold only for BFs (we also use unigrams and bigrams in our twitter sentiment system) also because each token of a tweet is very significant since that the tweets are quite different (e.g. very short, very informal, containing a lot of misspelled words, etc.) from other texts like product reviews and news articles used, instead, in the annotated data set for entity and relation classification.

2.4 Applications of Barrier Features

Eventually we perform BFs in several IE and IR tasks to prove effectiveness of BFs in different application contexts. In the *Semantic relation classification*, described in Chapter 3 BF contribution to performance is always relevant and on average can be evaluated in an improvement in F_1 in the range $[6\%, 15\%]$ [1]. In *Twitter sentiment analysis* reported in Chapter 5 the performance of the sentiment classification improves in the F_1 of about 2% using the BFs.

We think that the favorable impact of BFs is connected to their complementarity to simpler features like bigrams and trigrams on one side and the complete parse tree on the other, since that their introduction is useful in different application contexts. BFs are characterized only with a set of PoS tags including among two determined PoS tags, trigger and endpoint. Although such set is very simple to build, it manages to capture useful information that help to improve the performance of several tasks. Indeed BFs seem enough to be simple and robust because they are a good compromise respect to the data sparsity.

[1] The value of F_1 improvement varies in that range because we consider three different datasets in our experimental assessment (see Section 3.3)

Chapter 3

Barrier Features for Relation Classification

Approaches based on machine learning, such as Support Vector Machines, are often used to classify semantic relations between entities. In such framework, classification accuracy strongly depends on the set of features which are used to represent the input to the classifier. To classify semantic relations we are proposing here the introduction of the *barrier features* (BFs), which we defined in Chapter 2. They can be used in addition to more usual features, such as n-grams of PoS, word suffixes and prefixes, hypernyms from WordNet etc., and to the parse tree of the whole sentence. BFs aim at giving a compact representation of the context of each entity involved in the relation. We use only some BFs in this task because the syntactic structure of an entity is a noun phrase; for this reason we consider in this Chapter only the BFs which do not involve PoS tags of verbal phrases. For the sake of clearness, we reported in Table 2.1 the pairs of trigger and endpoint already shown in Table 3.1 (see the appendix A for more details about PoS tags) but some pairs have been deleted since we don't use them in the semantic relation classification.

The effectiveness of the BFs for semantic relation classification is assessed on three data sets: documents from the TREC data set annotated by Roth and Yih [98] and two SemEval

Endpoint	**Trigger**
DT	NN or NNP
PRP	NNS
JJ	JJR or RBR

Table 3.1: *Endpoints and triggers of the BFs employed in the assessment presented in Section 3.3.*

data sets, namely the Task04 of SemEval2007 [36] and the Task08 of SemEval2010 [46]. The obtained results show not only that the performance of the proposed approach are state-of-the-art but also that such improvement is due to the introduction of the BFs in the resolution of semantic relation classification. The main contributions of this Chapter can be summarized as follows:

- the introduction of BFs in semantic relation classification;
- their contribution to final result.

This Chapter is devoted to the application of BFs in semantic relation classification. In details after an initial discussion to introduce the task, the focus of the Chapter is the evaluation of the contribution of BFs and above all their effectiveness in the improvement of global performance of the classification of semantic relations. We apply the approach with BFs on the three available data sets for semantic relation classification.

3.1 Background and Motivation

Semantic annotation of documents is one of the main need for a more semantic oriented information retrieval on the web. Usually concepts and relations employed in the annotation are taken from an ontology, which could have been automatically learned or at least automatically refined. Therefore, annotation techniques based on manually constructed information would hamper the possibility of designing a completely automatic processing chain. In addition to that, an effective solution should be easy to port on different languages, so that it can be used in a multilingual context such as the web.

Different NLP approaches have been proposed for the identification of concepts (also called entities) and relations in texts. However, among all techniques proposed to solve such problem, only approaches requiring a minimal manual human intervention are considered. Most are based on the cascade of two steps, the former considering entities, the latter relations. Our focus is on relation classification, and therefore we assume that entities are given to us. We assume that for every considered relation we are given a number of positive examples, that is, a number of sentences where entities, relation instances and relation labels are annotated. This could for example be the output of a ontology refinement process, where the relation has been extracted on the basis of a number of examples.

On the other hand, several approaches exist for entity recognition [115, 80, 30, 74]. In this Chapter we focus on the semantic relation classification, where a label taken from a

finite set is associated to each relation. From a machine learning point of view, for each possible relation label we have a binary classification problem and we use Support Vector Machines (SVMs) [109] for such classification. By applying SVMs with different kernel functions we can handle different kinds of information, both structure-based (the parse tree of the input sentence) and boolean (features extracted from the words surrounding each of the involved entities). Indeed, we applied tree kernels [77] to the whole sentence parse tree and linear kernels to the feature vector associated to the input entities.

We add to more classical features also the focus of our work, namely the BFs, introduced in Chapter 2. Furthermore, BFs are defined independently of the form assumed by the entities as long as they can be represented as a sequence of words. Preliminary experiments on the data set used by Roth and Yih [98] less affected by data sparsity gave encouraging results [3].

However, a characteristic of all Boolean features is that their occurrence is as important as their absence for classification. Therefore, the design of the set of features to include in the input representation is crucial. On the other hand, a large part of them, including BFs, are easily affected by data sparsity, as already discussed in Section 2.3. In the preliminary assessment of BFs on the Roth and Yih data set considered in [3] the data sparsity did not affect performance in relevant way probably because of the specific characteristics of that data set. From the comparison with the experimental results we are presenting in this Chapter, our opinion is that the explanation is twofold. First of all, in that case the number of positive examples is larger than on most other data sets. Moreover, most entities contain proper nouns and then the syntactic variety of entity contexts is lower. On the other hand, the large scale applicability of the technique requires to face data sparsity. This was the case for example in both versions of SemEval data sets, which represent the two most used freely available data sets. Therefore, we used them for assessment, namely the Task04 of SemEval2007 [36] and the Task08 of SemEval2010 [46]. The solution we propose uses a large number of English texts to collect all the features. In fact, we only included features occurring at least a minimum number of times, but we considered a very low threshold aiming at excluding errors and rare patterns. The effectiveness of the approach is proven by the experimental assessment described in Section 3.3.

Therefore, while in [3] a preliminary evaluation of BFs effectiveness was considered, with no strategy to define the actual set of Boolean features, in this Chapter a new way of collecting the set of features is presented and assessed. Crucially, this new methodology only uses automatically annotated texts. This allows applying the approach with state-of-the-art performance to more realistic data sets such as the data of SemEval2007 Task4 and SemEval2010 Task8 and also to data set annotated by Roth and Yih considered in [3].

Note that while the strategy has been made necessary by the introduction of barrier ones, it has been successfully applied also to all the other Boolean features.

As said, the focus of this Chapter is the usage of BFs to associate semantic labels to relations. As we already discussed in Chapter 2, all features used in the assessment are Boolean features. The definition of such set is based on the automatic analysis of a large collection of plain texts, different from and much larger than the labeled sets used for training the classifiers. From this point of view, our approach can be compared to semi-supervised classification [18]. However, our approach is different in a crucial aspect, namely the fact that unannotated data are used to collect features independently from the considered classifier.

On the other hand, the approach we are proposing deal with the semantic relation classification, which has been considered by several recent works. Most of the systems aiming at extracting and classifying semantic relations are based on two steps: they first extract and label entities and afterwards relations. An important exception to the two pass approach is represented by [99], where entity and relation extraction and classification are integrated. Most relation classification systems are based on some machine learning approaches, and build a classifier which associates a label to a representation of the input sentence. In such approaches the representation of the input is crucial, as only the information it contains can be used for classification. Nearly all such systems consider some form of parsing: the complete parse tree of the input sentence is considered, among the others, by [75], which is based on a lexicalized, statistical parser, and [55], which considers both constituency and dependency parse trees obtained by a maximum entropy statistical parser. Systems that instead of the complete parse tree only consider some form of shallow parsing include [38] and [116].

Many of the best performing systems use kernel functions. The system presented in [85] is based on distributional kernels to compare co-occurrence probability distributions. Eventually, the system presented in [55] includes different kinds of features from several different sources e.g. Word Net, gazetteers, output of other semantic taggers etc.., applied to the specific task. It has been assessed on the Automatic Content Extraction (ACE) data set[1] , where it obtained competitive results, but performance on other data sets is not reported. When the syntactic information is represented by a tree, tree kernels represent the most direct option. In [26] relations between entities in the ACE corpus of news articles are detected and classified by applying tree kernels and SVMs to the dependency parse tree of the input sentence. This approach is characterized by the use of different features such as WordNet hypernyms, PoS, and entity types and of a dependency tree kernel.

[1] The ACE data set is not freely available.

One of the systems with which we compare our performance on the Task 4 of SemEval2007 is the University of Illinois at Urbana-Champaign (UIUC) system [7], which is based on SVMs together with a Radial Basis Function (RBF) kernel. The main characteristic of this approach is that it is characterized by a knowledge intensive feature set. Indeed, the features it employs are divided in three subsets: *core*, *context* and *special features*, and their construction is based on different external resources. Depending on this characteristics, the system results difficult to port on different domains and languages. Moreover, training has been performed by using additional examples for all except the *instrument-agency* relations.

In addition to the UIUC system, we also compare our performance with the FBK-irst system on the Task04 of SemEval2007 [37]. They assess both their approach to relation extraction and classification and the effect of automatic named-entity recognition on its performance. Their approach is based on shallow linguistic features, which are combined with semantic information, such as WordNet hypernym relations of the candidate entities. Kernels are employed to combine two different information sources: the global context where the two entities appear and (independently) the two local contexts of the entities. A specific kernel function is associated with each of the different types of information. Furthermore, we consider the HUJ system, presented in [27] which is the system which obtains the best performance on the Task04 of SemEval2007 without using the manually provided WordNet sense disambiguation tags. Their approach is based on a first clustering step, applied to entities, followed by classification of relations. They did not use the manually provided WordNet sense disambiguation tags.

In [97] semantic relations are classified by means of SVM by using features to describe context, semantic role affiliation, and relations between the nominals extracted by TextRunner. Some of these features are extracted by PropBank and FrameNet, two language dependent resources available only for English. The approach has been assessed on one of the data sets we used in our assessment, namely the Task 8 of SemEval2010.

In [51] a new method for semantic relation classification is presented using automatically-derived grammar rule clusters as a robust knowledge source for relation classification. In details a features set is built using all POS-tags, syntactic structure, and Cluster ID features come from the Berkeley Parser [90]. In their experimental assessment they employ the SemEval 2010 data set, one of data sets used in our assessment. Moreover we don't compare our approach with their system because their performance is lower than that of the approach presented in [97], winner of SemEval2010 competition.

3.2 Proposed Approach

In the relation classification problem two input entities, E_i and E_j, are connected by a relation $R_{i,j}$ and a label $l \in L$ is associated with $R_{i,j}$. We consider a binary version of the relation classification problem: a binary classifier is associated with every possible relation label $l \in L$, to discriminate if the entity pairs for both data sets are connected by the relation R labeled with l or not. Furthermore, the input representation of our classifier is composed by two parts: a vector of Boolean features and the input sentence parse tree. The former refer to the two input entities and are therefore called *entity features* (local context), while the latter refers to the whole sentence (global context) using a standard statistical parser to calculate the complete parse tree.

In the sentence s_2 in Table 3.2 the relation corresponding to the entity pair (e_1, e_2) is labeled as *work for*. As entity e_2 is composed by two tokens ("Korean CIA") the corresponding feature vector results from the OR combination of the features corresponding to each token. If the entity were composed by only one token, the feature vector would only contain 1's in correspondence of the features computed for this token.

Thus, features based on words are extracted from the window "The/DT spy/NN ,/, high-ranking/JJ Korean/JJ CIA/NNP". BFs construction is based on a window whose length is not predetermined, but depends on the PoS's of the tokens preceding the one we are considering, in this case "CIA". Since "CIA" PoS is NNP, we apply the first rule reported in Table 2.1 the endpoint is the closest determiner preceding the token *CIA*, namely *The*. In this case the endpoint does not belong to the entity, but this is not always so. The resulting BF is then given by the set *{JJ, NN, ,}*, and contains, as discussed, only one repetition of *JJ*, corresponding to the tokens *high-ranking/JJ Korean/JJ* , *spy/NN* and *,/,*.

For the sake of clarity, let us consider the example sentence s_1 of Table 3.2 extracted by the TREC data set annotated by Roth and Yih . It contains four different relations containing six entities, namely (e_1, e_2) with label "work for", (e_2, e_3) with label "orgbased in", (e_4, e_5) labeled as "work for", and (e_5, e_6) for "orgbased in". Indeed, entities e_2 and e_5 are involved in two different relations.

Let us introduce the following example which is taken from the Task04 of SemEval2007 data set. The sentence S ="The $\langle e_1 \rangle$device$\langle /e_1 \rangle$ uses the newest personal antenna $\langle e_2 \rangle$technology$\langle /e_2 \rangle$." The relation corresponding to the entity pair ($\langle e_1 \rangle$device$\langle /e_1 \rangle$, $\langle e_2 \rangle$technology$\langle /e_2 \rangle$) is labeled as *Instrument-Agency*. The second entity, namely $\langle e_2 \rangle$technology$\langle /e_2 \rangle$ is composed by only one token, and therefore the feature vector only contains 1's in correspondence of the features computed for this token. If the

s_1	Also being considered are $\langle e_1 \rangle$*Judge Ralph K. Winter* $\langle /e_1 \rangle$of the $\langle e_2 \rangle$*2nd U. S. Circuit Court of Appeals*$\langle /e_2 \rangle$ in $\langle e_3 \rangle$*New York City* $\langle /e_3 \rangle$ and $\langle e_4 \rangle$*Judge Kenneth Starr* $\langle /e_4 \rangle$ of the $\langle e_5 \rangle$*U. S. Circuit Court of Appeals*$\langle /e_5 \rangle$ for the $\langle e_6 \rangle$*District of Columbia*$\langle /e_6 \rangle$, said the source, who spoke on condition of anonymity.
s_2	The/DT spy/NN ,/, high-ranking/JJ $\langle e2 \rangle$Korean/JJ CIA/NNP$\langle /e2 \rangle$ official/JJ $\langle e1 \rangle$Sohn/NNP Ho/NNP Young/NNP$\langle /e1 \rangle$,/, wanted/VBD to/TO defect/VB ....

Table 3.2: *Example sentences taken from the Roth and Yih data set used for assessment.*

entity were composed by several tokens, the feature vector would have resulted from the OR combination of the features corresponding to each token. Thus, features based on tokens and PoS bigrams are extracted from the window "uses the newest personal antenna technology". In this case, the two entities are composed by just one token each, and therefore we take into account only one BF for each of them. BFs always depends on the PoS's of the tokens preceding the one we are considering, in this case "technology". Since "technology" PoS is NN, we apply the first rule reported in Table 2.1: the endpoint is the closest determiner preceding the token *technology*, namely *the*. In this case the endpoint does not belong to the entity, but this is not always so. The resulting BF is then given by the set *{JJ, NN}*, and contains, as discussed, only one repetition of *JJ*, corresponding to the two tokens *newest* and *personal*.

3.2.1 Feature Extraction using Unsupervised Dictionary Construction

In this subsection we describe the approach to build a vector of Boolean features to represent the context local to entities. As already described in Section 2.3.1, in our approach we split the training step in two step: *feature extraction* and the *classifier training*. Using text only labeled with PoS tags, namely *unsupervised dictionary*, the set of Boolean features can be built on it. In any case, the feature dictionary construction is performed in an unsupervised manner, and the crawled texts are processed in a completely automatic way. These texts are extracted from Wikipedia [2]. Before used them for feature extraction we apply a PoS Tagging.

Afterward, we extract the features to build the entity vector of Boolean features using the *unsupervised dictionary* from the training and the test sets, corresponding to the semantic relation classification, we are considering and we use these data to respectively build and assess the relation classifier.

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

Moreover to extract the features we use the smoothing strategies described in Chapter 2. The entity vector of features has been obtained by merging the feature vectors corresponding to each token in both entities involved in the relation we want to classify. As discussed in Chapter 2, all features are Boolean, taking a value *true* of *false* and the merging applies an OR operation of each feature.

Entity features are extracted from the substrings of tokens corresponding to the two entities and include *BFs*, word and PoS unigrams, PoS bigrams and trigrams, word prefix and suffix, word length, and a set of word features indicating whether the initial letter is upper case, whether all letters are upper or lower case, whether the token contains a period or number or hyphen. Furthermore, entity features also include a WordNet[3] [31] sense tag for each token involved in the entity. However, while the SemEval2007 Task04 data set comes with the correct WordNet sense tag, this is not the case for the SemEval2010 Task08 data set and in the data set annotated by Roth and Yih where we consider the most likely sense associated to the token. In addition to that, we also include in the features all the hypernyms. All features we consider in addition to the sentence parse tree are Boolean, indicating whether the corresponding event occurs in the input considered for classification.

More precisely, let $E_i = w_{i,1}, ..., w_{i,k_i}$ and $E_j = w_{j,1}, ..., w_{j,k_j}$ be the two entities we are considering. To obtain the feature vector used to classify the relation, the OR of the feature vectors for $w_{i,1}, ..., w_{i,k_i}$ and $w_{j,1}, ..., w_{j,k_j}$ is computed. For each of these tokens, we defined a windows made of the 5 preceding tokens and the 2 following ones. Similar number are often considered in analogous tasks, including PoS tagging, named entity recognition, etc..

3.2.2 Classification

To perform the classification we adopted SVM's [109] which in its basic definition is a binary linear classifier. As it tries to maximize the margin between the two classes, such classification approach attains a good generalization and usually has a good performance on different tasks. Moreover, it is able to face non linear cases by adopting kernels. In our case, as the classifier must be able to process an input composed by two different kinds of representation, namely a vector of binary values and a tree structure, kernels represent a smooth way to combine them. Indeed, the problem can be solved by combining a tree kernel ([78]) with a more traditional one, in our case a linear kernel. Tree kernels

[3] `http://wordnet.princeton.edu/`

evaluate the similarity between two trees in terms of the number of fragments they have in common.

To build the input representation in addition to the feature extraction step discussed in the previous section, we applied a statistical parser to the input sentence. We chose the Stanford Parser [58, 59][4], with the English grammar distributed together with the parser.

3.3 Experimental Assessment

3.3.1 Data Sets

For experimental assessment we used three data sets: (i) the data set used by Roth and Yih [98], derived from TREC corpus[5], which is freely available; (ii) the data of Task04 of SemEval2007 *Classification of Semantic Relations between Nominals of SemEval2007* described in [36]; (iii) the data of Task08 of SemEval2010 *Multi-Way Classification of Semantic Relations Between Pairs of Nominals* described in [46].

Roth and Yih data set

The data set used by Roth and Yih [98] includes three types of entities, namely PER (person), LOC (location) and ORG (organization) and the five types of binary relations reported in Table 3.3.

Relation	**Example**	**agent**	**target**
work for	employ-company	PER	ORG
kill	murderer-victim	PER	PER
live in	Clinton-USA	PER	LOC
located in	Rome - Italy	LOC	LOC
orgbased in	Harvard -USA	ORG	LOC

Table 3.3: List of relations with the type of the involved entities in the Roth Yih Data set.

The Roth and Yih data set is not divided in training and test set. Therefore assessment is performed by following the 5-fold cross validation protocol, as in [38, 99].

[4] The parser can be freely downloaded from `http://nlp.stanford.edu/software/lex-parser.shtml`.

[5] The annotated data are freely available at `http://l2r.cs.uiuc.edu/~cogcomp/Data/ER/conll04.corp`

SemEval data sets

The Task04 of SemEval2007 data set includes seven types of relations [36] as reported in Table 3.4 together with an example given for each relation.

Relation	Example
Cause-Effect	virus-flu
Instrument-Agency	laser-printer
Product-Producer	honey-bee
Origin-Entity	rye-whiskey
Theme-Tool	soup-pot
Part-Whole	wheel-car
Content-Container	apple-basket

Table 3.4: Task04 SemEval2007 Data set statistics.

Analogously, Table 3.5 refers to the SemEval2010 Task08 data set, which includes only nine types of relations [46]. Both SemEval2007 and SemEval2010 data sets do not include PoS's tagging and therefore we use the PoS tagging output by the parser also to construct the features.

Relation	Example
Cause-Effect	infection-inflammation
Component-Whole	elements-configuration
Entity-Destination	People-downtown
Entity-Origin	lawsuits-fans
Member-Collection	student-association
Message-Topic	citation-reasons
Content-Container	lawsonite-platinum crucible
Instrument-Agency	drugs-Adults
Product-Producer	products-industry
Other	sand-beach

Table 3.5: The SemEval2010 task08 Data set

3.3.2 System tuning and kernel choice

The classification was performed by using the SVM package SVMLight-TK[6] [78], which is based on SVMLight [52], but also includes tree kernels, and offers the possibility of combining them with other kernels to combine trees with a feature vector. To decide

[6] The package is available from `http://dit.unitn.it/~moschitt/Tree-Kernel.htm`.

which kernel if any should be used to process the entity feature vector, we compare the performance of different combinations of tree kernels alone or together with other kernels. These experiments have only be performed on the training set of SemEval2007 by using a 10-fold cross validation protocol: the best performance has obtained by linear kernel which has been then adopted in experiments on data sets.

3.3.3 Relation classification assessment on Roth and Yih data set

As discussed above, the first set of experiments has been performed on the Roth and Yih data set. Assessment considers five classifiers, one for each relation. The data set is divided in subsets corresponding to the different relations. For each relation, training has been performed by considering gold positive examples for the considered relation while negative examples are represented by all the other pairs of entities having labels compatible with the relation. In this way, the number of negative examples is much larger than for positive examples. The SVM implementation we used allows to balance the number of positive and negative examples by a cost factor. We set it to the rate between the number of negative and positive examples. Table 3.6 reports the comparison between the performance of our system and the results presented in [38] for the $M_{O|K}$ system. With the only exception of the *located in* relation, our system has an F_1 larger than $M_{O|K}$ both on single relations and on average. Although we are not able to estimate the statistical significance of such comparison because we do not have the output of that system on each sentence, we think that this consistency is quite convincing. Note however that in two cases their precision is better than ours, and in three cases their recall is better. However, the average values are always better for our system. Although we are not reporting the exact results here, we noticed that the WordNet features (hypernyms of each entity tokens) do not give any significant improvement on performance. This is probably due to the fact that most entity tokens are proper nouns, and therefore the WordNet sense does not add any information. An example entity taken from the data set is *Micheal Minns* which plays the role of agent in the relation *LiveIn*(*Micheal Minns*, *England*).

3.3.4 Relation classification assessment on the SemEval datasets

Relation classification assessment on SemEval2007 Task04

The experiments discussed in this and in the next section aim at comparing the performance of our system with published results of other systems. Therefore, they have been designed by strictly following the protocols of the experiments we want to compare our

	OurSystem			$M_{O|K}$		
Relation	**P**	**R**	**F1**	**P**	**R**	**F1**
kill	**92.39**	75.63	**83.17**	82.80	81.00	81.89
live in	74.69	**73.39**	**74.33**	78.00	65.8	71.38
work for	76.38	**86.18**	**80.99**	76.80	80.00	78.37
located in	70.00	75.40	72.60	79.60	76.00	77.76
orgabased in	**86.58**	**77.70**	**81.90**	74.30	77.20	75.72
average	**80.01**	**77.66**	**78.54**	78.30	76.00	77.02

Table 3.6: Comparison of performance of our system (OS) and the best performing one on the Roth and Yih data set. Bold cases correspond to the best performance.

results with. Performance for the SemEval2007 Task04 data set is reported in Table 3.7 where we consider the two systems with higher performance in the competition, that is the UIUC system [7], the FBK-irst system [37].

	Our Results			FBK-irst			UIUC		
Relation	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Cause-Effect	68.50	81.70	74.52	67.3	**90.2**	**77.1**	**69.5**	**100.0**	**82.0**
Instrument-Agency	73.65	73.82	73.73	**76.9**	**78.9**	**77.9**	68.2	**78.9**	73.2
Product-Producer	72.21	98.15	83.21	**76.2**	77.4	76.8	**84.5**	**79.0**	81.7
Origin-Entity	63.70	66.14	64.90	62.2	63.9	63.0	**86.4**	52.8	**65.5**
Theme-Tool	68.33	74.81	71.42	62.1	65.5	**85.7**	41.4	55.83	59.4
Part-Whole	80.24	67.17	73.13	65.5	73.1	69.1	**70.8**	65.4	67.99
Content-Container	77.41	67.69	72.23	78.8	68.4	**73.2**	**93.1**	**71.1**	**80.6**
average	71.11	76.97	73.48	70.9	73.4	71.8	**79.74**	69.8	72.4

Table 3.7: Comparison of the performance obtained by our system on the SemEval2007 data set with systems using more knowledge: the cases in which some other system obtained better performance than ours are in bold.

As discussed in Section 3.1, the UIUC system is a very knowledge intensive system, difficult to port on different domains and languages. Nevertheless, their global performance is slightly worse than our system, which is only based on features which can be extracted automatically. As the UIUC system is based on information useful for disambiguation, its precision is quite high, often higher than the one obtained by our system, which, on the other hand, has a better recall (bold figures corresponds to the UIUC values outperforming our system). Also on average, UIUC precision is higher than ours, while their recall is lower. The F_1 values are very close, but our system performs better on average.

Eventually, we report FBK-irst system performance, the second best result in SemEval2007, which is more similar to ours for the type of information used, but with (at least) one important difference: they only use shallow parsing, while we use a complete statistical parser. In any case, even if some performance figures are better for FBK-irst system, our

	Our Approach			UTD		
Relation	**P**	**R**	**F1**	**P**	**R**	**F1**
Cause-Effect	87.82	87.60	87.71	**89.63**	**89.63**	**89.63**
Component-Whole	74.43	80.52	77.36	74.34	**81.73**	**77.86**
Content-Container	82.12	86.34	84.18	**84.62**	85.94	**85.27**
Entity-Destination	80.22	92.73	86.02	**88.22**	**89.73**	**88.96**
Entity-Origin	82.42	81.00	81.70	**83.77**	80.62	**82.21**
Instrument-Agency	76.76	80.62	78.64	71.83	65.83	68.46
Member-Collection	84.51	85.23	84.87	84.30	**87.55**	**85.89**
Message-Topic	82.34	84.16	83.24	81.02	**85.06**	82.99
Product-Producer	83.77	81.13	82.43	82.38	74.89	78.46
Average	81.60	84.37	82.91	**82.25**	82.28	82.19

Table 3.8: Comparison of the performance obtained by our system on the SemEval2010 data set with systems using more knowledge: the cases in which some other system obtained better performance than ours are in bold.

performance is globally better. FBK-irst system looks for the best combination of kernels to combine different information sources at the best, while we think that the strength of our system is due to the BFs coupled with the strategy we adopted to cope with data sparsity. Thus, it would probably be a good idea to try combining their choice of kernels with this new kind of features, in the hope that performance can still improve.

Relation classification assessment on the SemEval2010 Task08 data set

In Table 3.8 we report the performance for the SemEval2010 Task08 data set and compare it with the system UTD [97], the winner of the SemEval2010 Task08 competition. For most relations UTD performs better (e.g. Cause-Effect precision and recall): an explanation could be that they use a word sense disambiguation system while we use the most likely sense. In addition to that they use semantically annotated resources like PropBank and FrameNet to extract semantic role features which are language dependent and are not easily portable to other languages. Also in this case, although F_1 values are very close, our system performs better.

3.3.5 Barrier Features contribution

Last but not least, we tried to understand the contribution of BFs to the global system performance. In order to obtain a numerical estimation, we run exactly the same experiment with and without BFs. In fact we build the two classifiers (the classifier are described in details in 3.2.2); the former is trained with an entity vector without BFs and parse tree

of the whole sentence the latter with a feature vector including BFs and with the same parse tree. Results on the Roth and Yih data set are reported in Table 3.9 and show that their contribution to performance is always relevant and on average can be evaluated in an improvement in F_1 of nearly the 15%.

One of the main difference of the approach we are proposing with respect to the other systems proposed in literature consists in the introduction of an approach to overcome data sparsity when using Boolean features. As in addition to the assessment of the complete system, we want to evaluate BFs contribution, we compare performance on these two data sets with and without them. Results, reported in Table 3.10 for the SemEval2007 Task04 and in Table 3.11 for the SemEval2010 Task08, show that their effect is definitely positive and their contribution to F_1 can be evaluated in about 6.6 % for SemEval2007 Task04 and about 7.6 % for the SemEval2010 Task08 sets.

3.4 Conclusion and Future Works

In this chapter we applied the BFs, defined in Chapter 2 to a new task, the classification of semantic relations and shown how they are effective in improving classification performance. Experimental assessment on the Roth and Yih data set and on two other data sets, the SemEval2007 Task04 and the SemEval2010 Task08, not only shows that the performance are state of the art, but also that their contribution is relevant. In other words the experimental results we obtained on these data sets demonstrated that the BFs captured information necessary to classify the relations. Furthermore we have proposed a new approach for the classification of semantic relations which overcame data sparsity when using Boolean features. Indeed, features are constructed on the basis of a large automatically tagged English text. By this approach also more precise Boolean features such as BFs can be effectively adopted and results are definitely competitive with other approaches.

Note that the whole system can be easily ported to different languages as long as a statistical parser and PoS tagger are available and a few language specific barrier rules are defined. Indeed, even if automatic parsing and tagging can introduce errors, the experimental results shown the approach robustness towards them. Note that we used some WordNet based features in order to compare our system against other state-of-the-art approaches. However, we plan to port the system on Italian by using a WordNet version for the target language.

	Without BFs			With BFs		
Relation	**P**	**R**	**F1**	**P**	**R**	**F1**
kill	70.30	71.29	70.79	**92.39**	**75.63**	**83.17**
live in	73.26	63.65	68.12	**74.69**	**73.39**	**74.33**
work for	66.22	63.12	64.63	**76.38**	**86.18**	**80.99**
located in	61.53	72.23	71.88	**70.00**	**75.40**	**72.60**
orgabased in	68.13	66.33	67.22	**86.58**	**77.70**	**81.90**
average	69.89	67.32	68.53	**80.01**	**77.66**	**78.54**

Table 3.9: Comparison of performance of our system on the Roth and Yih data set with and without BFs. Bold cases correspond to the best performance.

	No BFs			BFs		
Relation	**P**	**R**	**F1**	**P**	**R**	**F1**
Cause-Effect	61.00	80.54	69.42	**68.50**	**81.70**	**74.52**
Instrument-Agency	62.70	72.48	67.24	**73.65**	**73.82**	**73.73**
Product-Producer	71.00	97.91	82.31	**72.21**	**98.15**	**83.21**
Origin-Entity	62.20	63.23	62.71	**63.70**	**66.14**	**64.90**
Theme-Tool	67.85	63.00	65.34	**68.33**	**74.81**	**71.42**
Part-Whole	62.12	66.31	64.15	**80.24**	**67.17**	**73.13**
Content-Container	71.87	67.00	69.35	**77.41**	**67.69**	**72.23**
average	65.53	72.92	68.64	**71.11**	**76.97**	**73.48**

Table 3.10: Comparison of performance on SemEval2007 with and without BFs. Bold cases correspond to the best performance.

	No BFs			BFs		
Relation	**P**	**R**	**F1**	**P**	**R**	**F1**
Cause-Effect	73.82	85.60	79.27	**87.82**	**87.60**	**87.71**
Component-Whole	70.43	73.52	71.94	**74.43**	**80.52**	**77.36**
Content-Container	72.25	80.34	76.08	**82.12**	**86.34**	**84.18**
Entity-Destination	70.22	89.23	78.59	**80.22**	**92.73**	**86.02**
Entity-Origin	75.35	79.20	77.23	**82.42**	**81.00**	**81.70**
Instrument-Agency	71.23	78.23	74.57	**76.76**	**80.62**	**78.64**
Member-Collection	70.00	79.25	74.34	**84.51**	**85.23**	**84.87**
Message-Topic	80.21	80.23	80.22	**82.34**	**84.16**	**83.24**
Product-Producer	74.23	80.15	77.08	**83.77**	**81.13**	**82.43**
Average	73.08	80.64	76.59	**81.60**	**84.37**	**82.91**

Table 3.11: Comparison of performance on SemEval2010 with and without BFs. Bold cases correspond to the best performance.

Chapter 4

Jointly Entity and Relation Extraction using Graphical Model

Information Extraction (IE) aim is the extraction of structured or unstructured information from raw texts of a data set or a corpus, as, for example, entities and relations etc., as we already discussed in Chapter 3. In Figure 4.1 for example some instances of entities and relations are reported. The sentence is extracted from the data set annotated by Roth and Yih and used also in the assessment of Chapter 3.

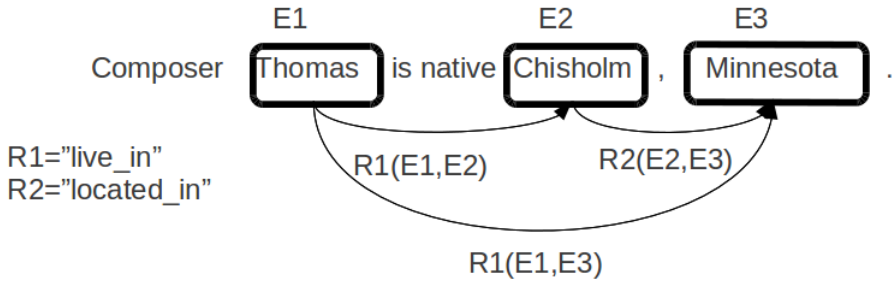

Figure 4.1: A sentence extracted by Roth and Yih annotated data set. $E1$, $E2$ and $E3$ are entities. $R1$ and $R2$ are respectively "$live\ in$" and "$located\ in$" relations.

In Chapter 3 we introduced a new approach to classify only semantic relations (*Kill*, *Located In*, etc.) between two entities given in input [1]. In this Chapter we propose two new approaches: the former, called *Pipeline system (PipeLS)*, is based on two steps where in the first step entities are classified and after these entities are the input of relation classifiers; the latter is a *Jointly Entity and Relation Extraction System (JERES)* in which

[1] Gold standard entities are employed during training and testing phases of relation extraction.

logical constraints, extracted by a predefined *Knowledge Base* (KB) (ontology or conceptual map) are used together with the classifier outputs to build a probabilistic model based on Graphical Models (GMs) [11, 53]. The classifiers used in both systems are based on the same system illustrated in Section 3.2, binary classifiers trained with classical features and BFs (see details in Chapter 2) and with the parsetree of each sentence.

The pipeline approach *PipeLS* is composed by two main modules: the former for the entity classification and the latter to classify the relations between entities. Furthermore the *PipeLS* have some limitations as, for example, error propagations between the first and second phase; indeed in the relation classification (second step of the pipeline) we can not evaluate again the choices taken by the entity module to change and to correct any errors. For these reasons we proposed the second system *JERES* based on GM which integrate the classifier behaviors with logical constraints of KB to overcome the problems of *PipeLS*.

KB-based IE systems are also employed to support the development of Semantic Web tasks [113]. In fact in several approaches [91, 113, 81] the ontologies (KB) are used to help the IE processes using its knowledge representation by means of logical constraints defined in it.

In experimental assessment 4.3 we apply the two systems, *PipeLS* and *JERES*, on the data set described in 3.3.1. In Chapter 3 we also use for relation classification other two data sets introduced in Section 3.3.1. We can not use these data sets in this Chapter because the entities of these corpora, involved in the relations (summarized in Tables 3.4 and 3.5) have not been annotated with specific entity types. (e.g. location or organization). Then we can not create an entity classifiers without an annotated data set with tags for entity types.

The main contributions of this Chapter can be summarized as follows:

- a approach to classify the entities is presented based on the classifier already described in Chapter 3 for the relation classification;
- *PipeLS*, a pipeline system, is introduced to extract entities and relations;
- the introduction of a Jointly Entity and Relation Extraction System (*JERES*) to build a probabilistic model for each sentence of a corpus, a novel integration between logical constraints extracted from an ontology or KB and the classifier output, based on GMs;
- the assessment of these two approaches on the data set annotated by Roth and Yih.

Eventually in this Chapter we present the two systems, namely *PipeLS* and *JERES* and we experimentally demonstrate that the second one based on the Graphical Model obtained better performance than the first one. Assessment of the systems are performed on the same data set annotated by Roth and Yih, already used in the semantic relation classification described in Chapter 3.

4.1 Background and Motivation

Most research approaches to entity and relation extraction are developed using two different architectures: pipeline approach or jointly entity and relation extraction. In the former, which we call *PipeLS*, each entity is determined independently and the output of entity extraction is the input to relation extraction; in the latter, called *JERES*, the extraction of a part of the information (entity extraction) is influenced by extraction from the other part (relation extraction) and vice-versa during all information extraction process.

Therefore in the *JERES* the usage of the logical constraints, extracted by KB or ontology or conceptual map, are useful to correct some errors of the pipeline systems. Based on the level of formality, an ontology or KB can vary from a simple taxonomy with almost no formalization, to one which uses a rigorously formalized theory (see [108]). In this Chapter KB includes logical constraints which are some properties and axioms enjoyed by the elements of the reference domain (instance of entities and relations). Indeed in this work we use the words KB and ontology as synonyms.

In the following paragraphs we summarize some methods proposed in literature and based on two types of architecture.

In [38], in addition to the method already described in Section 3.1, another pipeline approach, called $M_C|K_{SL}$ has been presented. This system is composed by two modules: the first one is the entity classifier based on Conditional Random Fields (CRFs); the second one takes in input the classified entities, computed in the previous step and classifies the relations. The second module is an SVM classifier trained with several kernel functions, one for each information source, as we already discussed in Section 3.1. At the best of our knowledge, their system represents the state-of-the-art on dataset annotated by Roth and Yih. For this reason in Section 4.3 we compare our system with $M_C|K_{SL}$.

A new approach for joint entity and relation extraction using a graph is presented in [57], namely "Card-Pyramid (CP)". All possible entities and relations in a sentence are encoded in such graph; in this way jointly labeling each node of this data structure they jointly extract the entities and relations. The labeling algorithm is very similar to the

parsing one and is based on dynamic programming. In the experimental assessment their joint system overcomes in performance the pipeline standard system proposed by [99], but not our joint approach JERES, as shown in Section 4.3.

In [114] another joint discriminative probabilistic model with arbitrary graphical structure is proposed to solve the problem of entity identification and relation extraction from encyclopedia articles. They introduce a unified framework based on undirected, conditionally-trained probabilistic GMs to optimize all relevant subtasks jointly. Moreover, a new inference method, called collective iterative classification (CIC), is presented to find the most likely assignments for both entities and relations. We can not compare our performance with theirs because it has been obtained on a different data set.

An approach for the joint extraction of entities and relations for opinion recognition and analysis is introduced in [19]. This system aims is to find links between opinion-related entities, expressions of opinions and sources of opinions. Then this links represent their relations which have to extract. An integer linear programming method is exploited to solve the joint opinion recognition task. Moreover the performance of both relation extraction and the extraction of opinion-related entities significantly improves using constraint-based inference and with the integration of a semantic role labeling. Our approach has not been compared with this system because although it is very similar at JERES, it solves a task a little different of ours. Indeed they use an annotated data set for Opinion Recognition which we do not use in our experiment assessment.

In the [110] a collective IE approach combining three tasks, Named Entity Recognition (NER), Relation Extraction (RE) and Coreference Resolution (CR), is proposed based on linear-chain conditional random fields. They presented an iterative training and labeling algorithm which uses new iterative and semantic features, based on the positions and the offsets of words. They build a probabilistic model using an arbitrary structured Condition Random Field (CRF) in which these features have been encoded. Assessment of their approach is performed on a dataset written in Slovene language then we do not perform our approach on it since *PipeLS* and *JERES* have been tested on an English data set.

Another joint approach to information extraction is proposed in [91]. This method is a single integrated inference process in which in a citation matching domain all records and entity resolution are performed together using Markov logic and the MC-SAT algorithm. In their solutions they write the appropriate logical formulas and combine them using Markov logic, a representation language that integrates probabilistic GMs and first-order logic. In their assessment they use the CiteSeer and Cora citation matching datasets. Though the *JERES* approach is very similar to that described in [91] we do not perform it

on the CiteSeer and Cora citation matching dataset because they are used to solve another task for retrieving links between the article citations.

Last but not least a general IE system based on logical and statistical rule that exploit Markov Logic Networks is proposed in [81]. The work focus is the system scaling to large datasets and the definition of generally applicable rules. In details a compiler finds the specialized algorithms automatically to solve NER, RE and CR and their task scheduler can combine all three IE tasks. The specialized algorithms is encoded in the model as operators and the approach is rule-based. We do not compare our system performance with their approach because we propose a pipeline method and an integrated probabilistic model for only Entity and Relation Extraction and we do not solve a CR task. Moreover their system is performed on a different dataset that we do not employ in our experimental assessment.

4.2 Proposed Approach

4.2.1 Pipeline system for Entity and Relation classification

While in Chapter 3 we only classify different kinds of relations (e.g. *Kill*, *Located In* etc.) since the correct entity pairs have been given in input, in this Section we introduce an approach to predict also the types of the entities, namely Location (Loc), Organization (Org) and Person (Peop). Afterwards the relation classification system extracts relation between the predicted entities.

To extract entities we use the same classifier illustrated in Section 3.2.2. Moreover to perform the classification we adopted SVM's [109] a binary linear classifier which has an input composed by two different kinds of representation, namely a feature vector of binary values (entity features) and a tree structure (sentence features)[2]. Kernels represent a smooth way to combine them. We train each classifier using a combination of a tree kernel [78] with a linear kernel, two different kernel functions to elaborate the information sources, respectively, sentence parsetrees and feature vectors. Afterwards we employ the predicted entities, outputs of entity classifiers, to classify the relations between them. A system schema is shown in Figure 4.2 and is used in experimental assessment on data set annotated by Roth and Yih described in 3.3.1. In the following two sections we describe in details entity classifier and relation classifier, modules of *Pipeline system* (*PipeLS*).

[2] This classification system is the same employed in Chapter 3.

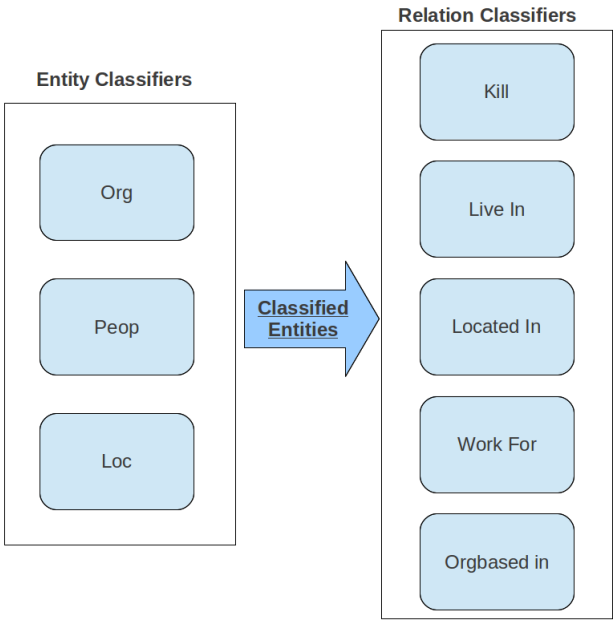

Figure 4.2: Pipeline system schema for data set annotated by Roth and Yih.

Entity Classifier

The $C_{\mathcal{E}}$ set is defined as follows:

$$C_{\mathcal{E}} = \{c_1, c_2, c_3, ...c_k\}$$

where $C_{\mathcal{E}}$ is the set of classes (types) to which the entities contained in the sentence s may belong and $E_j \in \mathcal{E}$ is a generic entities of s. Furthermore the correct entity boundaries are given as input. To solve the entity classification task we build an SVM classifier $\mathbb{C}_{c_i}$ for each class $c_i \in C_{\mathcal{E}}$. The classifier input are the *parse tree* of the sentence and the *feature vector* which represents the entity to be classified (E_j). The parse tree is computed using the Stanford Parser [58, 59] and the feature vector contains several kinds of features including for example unigrams of words, unigrams, bigrams and trigrms of PoS, lexical features, BFs etc. (see section 2 for more details about the BFs) . Each SVM classifier processes the two different inputs using a combination of two kernel functions: tree kernel [78] for parse tree and a linear kernel for the feature vector. For the sake of clarity, let us consider the example sentence s extracted by data set annotated by Roth and Yih and shown also in Figure 4.1:

"Composer **Thomas**$_{|E_1}$ is native **Chisholm**$_{|E_2}$, **Minnesota**$_{|E_3}$."

which contains three entities, $\mathcal{E} = \langle E_1, E_2, E_3 \rangle$, and we suppose that

$$C_{\mathcal{E}} = \{\textit{Loc},\ \textit{Peop},\ \textit{Org},\ \textit{Other}\}$$

[3] is the set of classes (types) to which the entities may belong. We build a classifier for each entity type $C_{\mathcal{E}}$ except for *Other* for which we can assert the following statement: E_i belongs to *Other* if and only if E_i do not belongs to neither *Loc* nor *Org* nor *Peop*.

Then the set of classifiers is:

$$\mathbb{C}_{\mathcal{E}} = \{\mathbb{C}_{\textit{Loc}},\ \mathbb{C}_{\textit{Peop}},\ \mathbb{C}_{\textit{Org}}\}$$

. Each classifier is learned independently and we use them to classify E_1, E_2 and E_3. If the output value of each classifier $\mathbb{C}_{c_i}$ is greater than 0 then c_i is the type of entity E_j .

Since each classifier is learned independently in the *PipeLS*, each of them can compute a positive margin related to entity E_j. Then this means that E_j can belong to more than one types $c_i \in C_{\mathcal{E}}$, that is, E_j can be in the same time a *Location*, a *Person* or an *Organization*. For instance in the sentence s for the entity E_1 the margin computed by $\mathbb{C}_{\textit{Peop}}$ is equal to 0.527 and the margin calculated by $\mathbb{C}_{\textit{Loc}}$ is equal to 0.368 by $\mathbb{C}_{\textit{Loc}}$ then E_1 belongs to both types *Peop* and *Loc*. On the contrary the *JERES* solve this semantic ambiguity as we describe in 4.2.2, choosing suitably the types most likely to E_1.

Relation Classifier

As already described in Chapter 3, we consider only the relations between entities within the same sentence; in other words we do not solve the cross-sentence relation classification task because the relations between entities in different sentences are not annotated in the corpus used for evaluation. Moreover all relations automatically extracted by the data set are binary and anti-reflexive. No entity is in relation with itself, that is, $\forall E_i \in \mathcal{E}, \neg R_{i,i}$ (self-relations). Furthermore two entities involved in relation R play two different roles (*Agent* or *Target*) given by the position of entities in the relation ($R_{i,j} \neq R_{j,i}$).

$$C_R = \{c_1, c_2, c_3, ...c_{k'}\}$$

is the set of classes (types) to which the relations in a sentence may belong. Each entity involved in a relation is tagged using entity classifier with one or more types contained in

[3] *Loc*, *Peop* and *Org* stand for respectively *Location*, *Person* and *Organization*

the $C_{\mathcal{E}}$ set. As described for entities in the previous Section, we also build an SVM binary classifier $\mathbb{C}_{c_z}$, one for each class $c_z \in C_R$ for relation classification. We train the classifier using two kernel functions, employed to combine two different information sources: the global context where the two entities appear, parse tree of the sentence which contains the relation and a vector of Boolean features to represent the local context of entities (vector built as reported in section 3.2.1). If the output margin of $\mathbb{C}_{c_z}$ classifier is greater than 0 then a relation R between two entities exists and it is the c_z type. For the sake of clarity, let us consider again the example sentence extracted by the data set annotated by Roth and Yih and shown also in Figure 4.1:

"Composer **Thomas**$_{|E_1}$ is native **Chisholm**$_{|E_2}$, **Minnesota**$_{|E_3}$."

$$C_R = \{\textit{Kill},\ \textit{Located In},\ \textit{Work For},\ \textit{OrgBased In},\ \textit{Live In},\ \text{No Rel}\}$$

is the set of classes to which the relations contained in the sentence s may belong.

$$C_{\mathcal{E}} = \{\textit{Loc},\ \textit{Peop},\ \textit{Org},\ \textit{Other}\}$$

is the set of types, possible labels of the entities involved in the relation. From now on if a relation $R_{i,j}$, defined between two entities (E_i, E_j), is tagged with a class $c_z \in C_R$ then we say that the entity pair (E_i, E_j) is involved in a relation of c_z type. We suppose that not all relation types can be expressed between some pair of entity types and in particular each kind of the relation is compatible only with a pair of entity types. Moreover an entity tagged with label *Other* can not be involved in any relation. In the Table 3.3 we show the constraints of entity types for each kind in the relation of data set used in the experimental assessment. Eventually, to perform relation classification we build one classifier for each class $c_z \in C_R$ but not for *No Rel* class since the elements of this class are all relations which do not belong to any other class.

The set of SVM relation classifier is defined as follows:

$$\mathbb{C}_R = \{\mathbb{C}_{\textit{Kill}},\ \mathbb{C}_{Live\ In},\ \mathbb{C}_{Work\ For},\ \mathbb{C}_{Located\ In},\ \mathbb{C}_{OrgBased\ In}\}.$$

Each classifier is applied independently from the other ones and it has for input the *parse tree* of the sentence and the *feature vector* which represent one of the following pairs of entities: (E_1, E_2), (E_2, E_1), (E_2, E_3), (E_3, E_2), (E_1, E_3), (E_3, E_1). For instance if the margin computed by $\mathbb{C}_{\textit{Located In}}$ for the entity pair (E_2, E_3) is equal to 0.143 then E_2 and E_3 are involved in relation *Located In*.

As for the entity in *PipeLS* an entity pair can belong to one or more classes in the same time since each relation classifier is applied independently from the other ones. For example, the entity pairs (E_2, E_3) can belong to *Located In* and *Work For* classes because $\mathbb{C}_{\textit{Located In}}$ and $\mathbb{C}_{\textit{Located In}}$ computed two positive margins for pair (E_2, E_3). This is another ambiguity problem of *PipeLS* which we solve using the integration probabilistic model built by *JERES*. As we presented in Section 4.2.2 we define a logical constraint which assert that the instance of entity and relation in a sentence can only belong to one class. Therefore if the entity pair (E_2, E_3), for instance, belongs to more than one class considering only the outputs of classifiers (in *PipeLs*), the new probabilistic model based on GMs (built by the *JERES*) combines these outputs with the defined logical constraint. Then in *JERES* the entity pair (E_2, E_3) will be ssociated to the more likely class, deleting the ambiguity problem.

4.2.2 Jointly Entity and Relation Extraction system using Graphical Models

The use of classification methods, described in the previous paragraph, to solve IE tasks is not sufficient to fix some problems inherent to natural language such as:

- *language ambiguity*: in the same sentence one or more language ambiguities may arise related to different levels of analysis (lexical, syntactic, semantic). Individual words or phrases can often have more than one interpretation, and although this does not represent a crucial issue for humans, instead it is a problem for the development of an automatic approach;
- *the incompleteness of information*: the information, contained in a sentence, is not always expressed in a complete and explicit way. It often happens that some concepts are not extensive because they are non-functional in the speech or implied since they are logical consequence of others. When a concept is non-functional in the speech, the information is completely absent and not deducible from the context of the sentence;
- *the presence of "negative information"*: the information, contained in a sentence, can be expressed either by a concept or through its negation;
- *the presence of contradictory information*: there are some cases in which the information, contained in the sentence, expresses concepts that are in contrast ones with others.

Moreover the *PipeLS* system has some problems which also depend on the pipeline architecture of the system. The errors of entity classifiers (e.g. the same entity or relation belongs to two or more types) are propagated in the relation classification and besides they can not be corrected in the relation classification, changing the predictions (output values) computed by entity classifiers.

Furthermore to overcome the limits of machine learning classifiers we combine two information sources in a probabilistic model: the output of each classifier and the logical constraints defined in the ontology (KB). We propose a new method, namely *JERES*, for joint entity and relation extraction using GMs [11, 53] and in particular we use *Markov Random Fields* [22], with undirected graphs. Each graph node corresponds a random variable $\mathcal{X}_k$ which describes the behavior of a classifier $\mathbb{C}_{p_k}$ where p_k is a unary or binary predicate as we explain in the following paragraph. Each edge in the model is induced by a constraint of a KB (ontology or conceptual map).

Definition of Ontological Constraints

Given a sentence s we consider $\mathcal{E} = \langle E_1, E_2, ...E_N \rangle$ the sequence of entities contained in s on which several predicates may be expressed, $\{P^{(i)}\}_{i \in I}$, with $I = 1, 2, 3, ...$, identifying properties and/or relations. In this work we only exploit unary and binary predicates, $P^{(1)}$ and $P^{(2)}$, the former ones express entity properties and the latter ones specific the relation between two entities. Let us consider again the running example shown also in Figure 4.1:

"Composer **Thomas**$_{|E_1}$ is native **Chisholm**$_{|E_2}$, **Minnesota**$_{|E_3}$."

in which the entities $\mathcal{E} = \{E_1, E_2, E_3\}$ are contained in s. We must extract two types of information split in two distinct sets of predicates:

$$P^{(1)} = \{\textit{Loc}(\cdot), \textit{Org}(\cdot), \textit{Peop}(\cdot)\}$$

a set of unary predicates which contains entity types and

$$P^{(2)} = \{\textit{Located In}(\cdot,\cdot), \textit{Kill}(\cdot,\cdot), \textit{OrgBased In}(\cdot,\cdot), \textit{Live In}(\cdot,\cdot), \textit{Work For}(\cdot,\cdot)\}$$

a set of binary predicates whose elements are the types of relation between two entities. The binary predicates have the semantic properties described in the Table 4.1:

Relation	Entity Type 1	Entity Type 2	Properties
Kill	Peop	Peop	AR, AS, NT
Live In	Peop	Loc	AR, AS, NT
Work For	Peop	Org	AR, AS, NT
Located In	Loc	Loc	AR, AS, T
OrgBased In	Org	Loc	AR, AS, NT

Table 4.1: List of properties of each relation. The acronyms AR, AS, NT, T mean respectively, Anti-reflexive, Antisymmetric, Negative Transitive and Transitive.

- an entity can not belong to two or more different types (classes) in the same time (in a sentence):
  for each $E_i \in \mathcal{E}$ and $p_z \in P^{(1)}$

$$p_z(E_i) \ \wedge \bigwedge_{p_j \in P^{(1)} \ con \ j \neq z} \neg p_j(E_i)$$

- all relations are anti-reflexive:
  for each $E_i \in \mathcal{E}$ and for each $p \in P^{(2)}$

$$\neg p(E_i, E_i)$$

- all relations are antisymmetric:
  for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$, and for each $p \in P^{(2)}$

$$p(E_i, E_j) \rightarrow \neg p(E_j, E_i)$$

- all relations, except *Located In*, are negative transitive:
  for each $E_i, E_j, E_k \in \mathcal{E}$, with $E_i \neq E_j \neq E_k$, and for each $p \in P^{(2)}$, such that $p \neq$ *Located In*$(\cdot, \cdot)$

$$p(E_i, E_j) \wedge p(E_j, E_k) \rightarrow \neg p(E_i, E_k)$$

- the relation *Located In*$(\cdot, \cdot)$ is transitive:
  for each $E_i, E_j, E_k \in \mathcal{E}$, with $E_i \neq E_j \neq E_k$

$$p(E_i, E_j) \wedge p(E_j, E_k) \rightarrow p(E_i, E_k)$$

  with $p =$ *Located In*$(\cdot, \cdot)$

- each type of relations is only compliant to one pair of entity type
  for each $E_i, E_j \in \mathcal{E}$,with $E_i \neq E_j$

$$\textit{Kill}(E_i, E_j) \rightarrow \textit{Peop}(E_i) \wedge \textit{Peop}(E_j)$$

$$\textit{Live In}(E_i, E_j) \rightarrow \textit{Peop}(E_i) \wedge \textit{Loc}(E_j)$$

$$\textit{Work For}(E_i, E_j) \rightarrow \textit{Peop}(E_i) \wedge \textit{Org}(E_j)$$

$$\textit{Located In}(E_i, E_j) \rightarrow \textit{Loc}(E_i) \wedge \textit{Loc}(E_j)$$

$$\textit{OrgBased In}(E_i, E_j) \rightarrow \textit{Org}(E_i) \wedge \textit{Loc}(E_j)$$

Given the properties of the elements of domain, the consistency constraints contained in the considered fragment of ontological knowledge are formalized as follows:

1. $p_z(E_i) \rightarrow \neg p_j(E_i)$
   for each $E_i \in \mathcal{E}$ and $p_z, p_j \in P^{(1)}$, with $z \neq j$
2. $p(E_i, E_j) \rightarrow \neg p(E_j, E_i)$
   for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$, and $p = \textit{Located In}(\cdot, \cdot)$ or $p = \textit{Kill}(\cdot, \cdot)$
3. $\textit{Located In}(E_i, E_j) \wedge \textit{Located In}(E_j, E_k) \rightarrow \neg\textit{Located In}(E_k, E_i)$
   for each $E_i, E_j, E_k \in \mathcal{E}$, with $E_i \neq E_j \neq E_k$
4. $\textit{Kill}(E_i, E_j) \rightarrow \textit{Peop}(E_i) \wedge \textit{Peop}(E_j)$
   for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$
5. $\textit{Live In}(E_i, E_j) \rightarrow \textit{Peop}(E_i) \wedge \textit{Loc}(E_j)$
   for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$
6. $\textit{Work For}(E_i, E_j) \rightarrow \textit{Peop}(E_i) \wedge \textit{Org}(E_j)$
   for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$
7. $\textit{Located In}(E_i, E_j) \rightarrow \textit{Loc}(E_i) \wedge \textit{Loc}(E_j)$
   for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$
8. $\textit{OrgBased In}(E_i, E_j) \rightarrow \textit{Org}(E_i) \wedge \textit{Loc}(E_j)$
   for each $E_i, E_j \in \mathcal{E}$, with $E_i \neq E_j$

Note that:

- since all the predicates in the set $P^{(1)}$ are mutually exclusive, we can infer first the constraint;

- the second constraint describes the antisymmetric property only for *Located In* and *Kill* relations because they are only two relations in which the type of the entities involved in the relation is the same;
- the third constraint is logical entailment of antisymmetric and transitive properties of *Located In* relation;
- since all predicates in $P^{(2)}$ have the anti-reflexive property, the pairs (E_i,E_i) can not involved in any relations.

Probabilistic Integration Model based on Markov Random Fields

To integrate the classifier output and logical constraints in a probabilistic model based on Markov Random Fields we have to convert the SVM margins [109], outputs of entity and relation classifiers, to probabilities normalizing the absolute value of sigmoid function as follows:

$$\sigma(M) = \frac{1}{1 - e^{-M}}$$

where M represented the absolute value of margin. The integration model is a probabilistic model based on Markov Random Fields. For each sentence s in data set we build in *JERES* an undirected graph which contains a variable for each entity $E_i \in \mathcal{E}$ and a variable for each instance of binary predicates (relation) $p_k \in P_s$, namely respectively *entity variable* and *relation variable*. The last one will be represented by the pairs $((E_i, E_j), \mathcal{R})$ in which the first element is a ordinated pair of two entities $E_i, E_j \in \mathcal{E} : i \neq j$ and the second element $\mathcal{R}$ is the relation type in which the entities are involved. since in our task each classifier is applied independently, we do not use the *Bayesian Network* with which we should decide an a-priori ordering to apply the different classifier $\mathbb{C}_{p_k}$ to classify the instance of $p_k \in P_s$. Each variable X_i patterns an application of a classifier and its probability distribution describe the classifier behavior. The *relation variables* shape exactly the binary classifiers, built for relation classification and presented in the paragraph 4.2.1. On the contrary each *entity variable* models a multi-class classifier ($\mathbb{C}_{E_i}$ with four classifiers $C = \{Loc, Peop, Org, Other\}$) assembled using a right combination of outputs of three binary classifiers ($\mathbb{C}_{Loc(E_i)}, \mathbb{C}_{Org(E_i)}$ and $\mathbb{C}_{Peop(E_i)}$) for entity classification (paragraph 4.2.1). Finally the *relation variables* are all binary and they undertake values in the set {*True*, *False*}, while the default alphabet {*Loc*, *Org*, *Peop*, *False*} contains the values of the *entity variables*. The last ones represent the entity types (the *False* value corresponds to *Other* type of entities) and they are mutually exclusive. Every entity type can be associated to each entity $E_i \in \mathcal{E}$ with a probability ≥ 0 and a different probability distribution. The edges which link the nodes of the model are induced only by the constraints

of KB (ontology). Furthermore a link between two variables exits if there is a constraint in the ontology such that the value of a variable depends on a value of another one, that is, two variables are not statistically independent from probabilistic point of view. Indeed each relation node $((E_i, E_j), \mathcal{R})$ which involve the E_i and E_j, is linked with entity nodes E_i and E_j. No edge exits between two entity nodes because the related *entity variables* are statistically independent.

For sake of clarity, let us consider the following sentence s [4]:

"Composer **Thomas**$_{|E_1}$ is native **Chisholm**$_{|E_2}$. "

which contains two entities ($\mathcal{E} = E_1, E_2$) and the set of logical constraint included in the KB is defined as follows:

1. Located In$(E_1, E_2) \rightarrow \neg$Located In$(E_2, E_1)$;
2. Kill$(E_1, E_2) \rightarrow \neg$Kill$(E_2, E_1)$;
3. Kill$(E_1, E_2) \rightarrow$ *Peop*$(E_1) \wedge$ *Peop*(E_2);
4. Kill$(E_2, E_1) \rightarrow$ *Peop*$(E_1) \wedge$ *Peop*(E_2);
5. Live In$(E_1, E_2) \rightarrow$ *Peop*$(E_1) \wedge$ *Loc*(E_2);
6. Live In$(E_2, E_1) \rightarrow$ *Peop*$(E_2) \wedge$ *Loc*(E_1);
7. Work For$(E_1, E_2) \rightarrow$ *Peop*$(E_1) \wedge$ *Org*(E_2);
8. Work For$(E_2, E_1) \rightarrow$ *Peop*$(E_2) \wedge$ *Org*(E_1);
9. Located In$(E_1, E_2) \rightarrow$ *Loc*$(E_1) \wedge$ *Loc*(E_2);
10. Located In$(E_2, E_1) \rightarrow$ *Loc*$(E_1) \wedge$ *Loc*(E_2);
11. OrgBased In$(E_1, E_2) \rightarrow$ *Org*$(E_1) \wedge$ *Loc*(E_2);
12. OrgBased In$(E_2, E_1) \rightarrow$ *Org*$(E_2) \wedge$ *Loc*(E_1)

Note that the constraints of mutual exclusion related to the entity types are not included in the KB because this property is encoded with the construction of the multi-class classifier of entities. The *Markov Random Fields* related to previous sentence s is represented in Figure 4.3: where:

- the edges 'a', 'm' were induced by constraint 8;

[4] In this example we only consider a part of the sentence shown in Figure 4.1 because we want to paint the readable graph in Figure 4.3

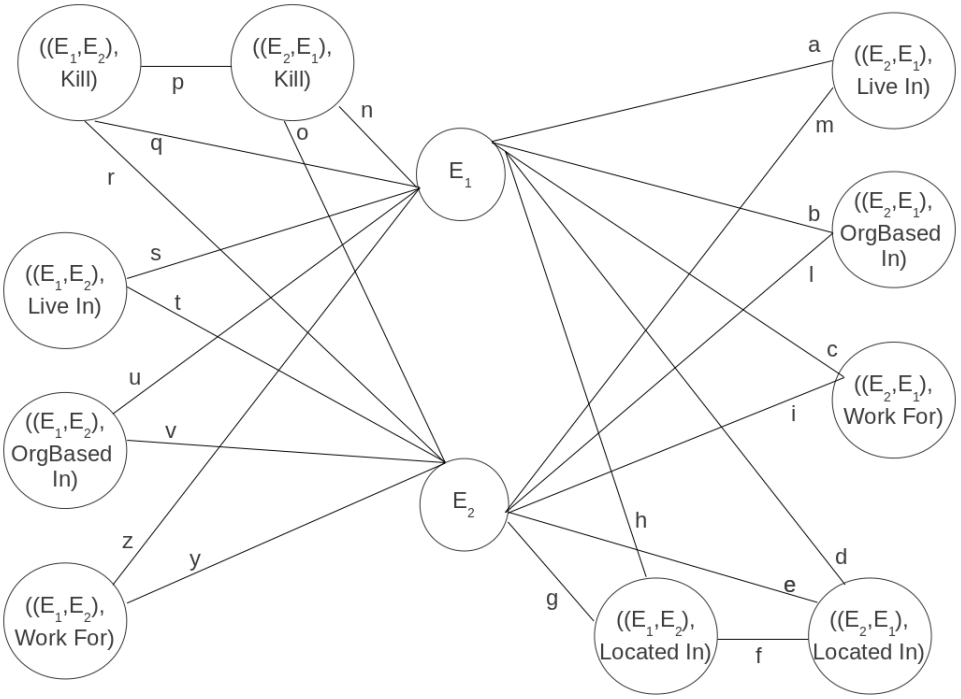

Figure 4.3: Example of Construction of Graphical Model related to a sentence extracted by data set annotated by Roth and Yih in 2007 and contained two entity E_1 and E_2.

- the edges ‘b’, ‘l’ were induced by constraint 14;
- the edges ‘c’, ‘i’ were induced by constraint 10;
- the edges ‘d’, ‘e’ were induced by constraint 12;
- the edges ‘h’, ‘g’ were induced by constraint 11;
- the edges ‘n’, ‘o’ were induced by constraint 6;
- the edges ‘q’, ‘r’ were induced by constraint 5;
- the edges ‘s’, ‘t’ were induced by constraint 7;
- the edges ‘u’, ‘v’ were induced by constraint 13;
- the edges ‘z’, ‘y’ were induced by constraint 9;
- the edge ‘f’ was induced by constraint 1;
- the edge ‘p’ was induced by constraint 2.

Since the model in Figure 4.3 we can observe that an edge corresponds to a single constraint, but a constraint may induce one or more edges. The *JERES* builds this probabilistic model for each sentence extracted by data set used in experimental assessment.

Maximization of the Joint Probability

To extract the information, entities and relations contained in a sentence, in the *JERES* an inference algorithm is applied to compute the assignment of all random variables in the graph which maximizes the joint probability $\Pr(\mathcal{X})$. After some preliminary experiments we choose the *Loopy Belief Propagation* [79, 88, 105], as inference algorithm. The joint probability $\Pr(\mathcal{X})$ can be decomposed as:

$$\Pr(\mathcal{X}) = \frac{1}{Z}\prod_{\mathcal{C}} \psi_{\mathcal{C}}(\mathsf{X}_{\mathcal{C}})$$

in which $\psi_{\mathcal{C}}(\mathsf{X}_{\mathcal{C}})$ is a potential function defined on *maximal cliques* ($\mathsf{X}_{\mathcal{C}}$) of the graph. To calculate the joint probability the first step consists in identifying the maximal cliques of the graph. Each one of them "activates" a number of logical constraints of KB, whose number and type depend, respectively, by the number of the edges in the clique and the variables that connect these edges. A specific assignment of the variables in the clique is considered "valid" only if all the activated constraints are satisfied by the current assignment. If an assignment is not valid it means that the expressed information is not consistent. Each clique is associated to a number of assignments which is the product of the number of values that each variable can take. Given a valid assignment a score, namely *potential*, is assigned which depends on the relevance of the expressed information: if such score is high then most likely the variables take locally the current value. The potential is computed using a potential function $\psi_{\mathcal{C}}$. It is strictly positive function and then it can be defined as an exponential function:

$$\psi_{\mathcal{C}}(\mathsf{X}_{\mathcal{C}}) = \exp\{-\mathbb{E}(\mathsf{X}_{\mathcal{C}})\}$$

where $\mathbb{E}(\mathsf{X}_{\mathcal{C}})$ is energy function defined as follows:

$$\mathbb{E}(\mathsf{X}_{\mathcal{C}}) = \begin{cases} +1 \cdot \sum_{\mathsf{X}_i \in \mathcal{C}} \alpha(\mathsf{X}_i), & \text{if the current assignment} \mathcal{C} \text{ is valid} \\ -1 \cdot \sum_{\mathsf{X}_i \in \mathcal{C}} \alpha(\mathsf{X}_i), & \text{else} \end{cases}$$

The α value strictly depends on the value which the variable takes in the current configuration ($\alpha(X_i = val = \gamma)$ and can overlap with:

- log probability ($Pr(X_i = val)$) given in output by the classifier which corresponds to X_i variables, if the predicate classified is unary (entity variable);

- log probability of the following product:

$$\Pr(\mathsf{X}_i = \mathrm{val}_i) \cdot \Pr(\mathsf{X}_j = \mathrm{val}_j) \cdot \Pr(\mathsf{X}_z = \mathrm{val}_z)$$

  if the predicate classified which defines the variable X_i is binary and where $\Pr(\mathsf{X}_j)$ and $\Pr(\mathsf{X}_z)$ represent the probabilities that the entities, arguments of predicates, are some type (relation variable).

We calculate two previous different log probably because the probability, computed using the output score of a classifier built to classify a binary predicate, is a conditional probability of that predicate given the entity types which are its arguments. Furthermore we choose the values of the energy function so that if the current assignment of the *clique* is not valid the potential value associated with it is between 0 and 1, otherwise it is greater than 1.

4.3 Experimental Assessment

In *PipeLS* (a schema is shown in Figure 4.2) each classifier, module of pipeline approach, is implemented with the SVM package SVMLight-TK[5] [78] because these software offers the possibility of combining tree kernels with other kernels (in our case linear kernel) to manage two information sources: parse tree and a feature vector. The parse tree is computed using Stanford Parser [58, 59][6] The Roth and Yih data set is not divided in training and test set. Therefore assessment is performed by following the 5-fold cross validation protocol, as in [38, 57]. The Roth and Yih data set has already been described in details in section 3.3.1.

In *JERES* to build the probabilistic model based on Markov Random Field for each sentence of the Roth and Yih data set we use GRMM[7], a toolkit for computing inference and learning in GMs of arbitrary structure. We also employ the JAVA implementation of GRMM of Loopy Belief Propagation, the inference algorithm Loopy Belief Propagation to compute the maximal joint probability of each graph built for each sentence contained in the Roth and Yih data set. In Figure 4.4 a schema of *JERES* is shown. The *JERES* system is composed by the following modules: entity and relation classifiers, a constructor to build the probabilistic model using the classifiers outputs and the KB logical constraints

[5] The package is available from `http://dit.unitn.it/~moschitt/Tree-Kernel.htm`.

[6] The parser can be freely downloaded from `http://nlp.stanford.edu/software/lex-parser.shtml`.

[7] The software is freely downloaded by `http://mallet.cs.umass.edu/grmm/index.php`

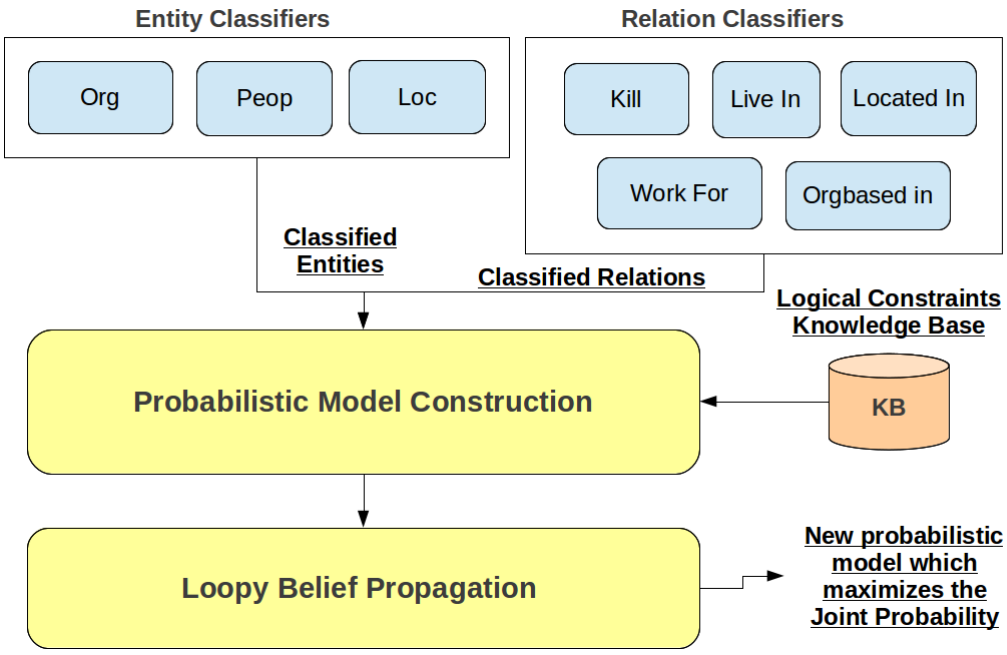

Figure 4.4: The schema of JERES for entities and relation extraction

and the inference algorithm Loopy Belief Propagation to compute the maximal joint probability of each graph.

4.3.1 Discussion

In this section the experimental results are reported and discussed for entity and relation classification. Following related works, performance is evaluated by computing Precision (P), Recall (R) and F_1 as usual. In Tables 4.2 and 4.3 we show the comparison between pipeline system (*PipeLS*) based on the SVM classifiers (our baseline introduced in Section 4.2.1) and *JERES*, based on the probabilistic model, obtained with the integration of classifier outputs and logical constraints using Markov Random Fields (presented in Section 4.2.2). The *JERES* performance is better than that of the *PipeLS* for both entity and relation classification, except for Precisions of *Loc* and *Located In* and accordingly the micro and macro average precision. Then the experimental results demonstrate that the novel probabilistic integration model based on GMs corrects some mistakes of the pipeline system. On the other hand, when we can run both systems we are comparing and we apply approximate randomization to evaluate statistical significance. Average performance is always significantly better for *JERES*, with the only exception of precision of entity classification, where there is not statistical significance.

	Peop			*Org*			*Loc*		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
PipeLS	94.83	90.01	92.35	93.57	78.35	85.29	**92.97**†	88.05	90.44
JERES	**95.03**†	**90.53**†	**92.73**†	**93.92**†	**78.54**†	**85.54**†	92.09	**89.68**†	**90.87**†

	micro-average			macro-average		
	P	**R**	**F1**	**P**	**R**	**F1**
PipeLS	**93.78**	87.00	90.11	**93.79**	85.47	89.36
JERES	93.53	**88.00**†	**90.48**†	93.68	**86.25**†	**89.71**†

Table 4.2: Experimental results to evaluate the Graphical Model contribution on entity classification task (*JERES* system). Bold cases correspond to the best performance. Symbol † indicates that the ρ-values, computed by approximate randomization, are smaller than 0.05 level.

	Kill			*Live In*			*Work For*		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
PipeLS	69.18	**85.45**	76.46	70.36	66.99	68.63	72.70	71.07	71.88
JERES	**69.82**†	**85.45**†	**76.85** †	**72.03**†	**68.85**†	**70.40**†	**73.30**†	**72.57**†	**72.93**†

	Located In			*OrgBased In*		
	P	**R**	**F1**	**P**	**R**	**F1**
PipeLS	**71.94**†	69.63	70.77	75.95	70.58	73.17
JERES	71.82	**71.11**†	**71.46**†	**76.92**†	**70.80**†	**73.73**†

	micro-average			macro-average		
	P	**R**	**F1**	**P**	**R**	**F1**
PipeLS	72.08	72.00	71.80	72.03	72.74	72.18
JERES	**72.88**†	**73.00**†	**72.75**†	**72.78** †	**73.75**†	**73.08**†

Table 4.3: Experimental results to evaluate the Graphical Model contribution on relation classification task (*JERES*). Bold cases correspond to the best performance. Symbol † indicates that the ρ-values, computed by approximate randomization, are smaller than 0.05 level.

In the Tables 4.4 and 4.5 the performance of baseline system (*PipeLS*) and *JERES* is compared to $M_C|K_{SL}$ and Card-Pyramid (CP) approaches, presented respectively in in [38] e [57]. The performance of *JERES* for entity classification is less than that obtained by $M_C|K_{SL}$; on the contrary the integration system *JERES* based on *Markov Random Fields* for relation classification overcome the state-of-art systems in most cases, except in precision and F1 of *Kill* and *Work For* relation and precision of *Located In*. However our *JERES* has an F_1 larger than $M_{C|K}$ and CP approach on average.

	Peop			*Org*			*Loc*		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
PipeLS	94.83	90.01	92.35	93.57	78.35	85.29	92.97	88.05	90.44
JERES	**95.03**	90.53	92.73	**93.92**	78.54	85.54	92.09	89.68	90.87
$M_C \vert K_{SL}$	94.80	**96.60**	**95.70**	91.90	88.50	**90.20**	**94.20**	**94.40**	**94.30**
CP	92.10	94.20	93.20	90.50	**88.70**	89.50	90.80	94.20	92.40

	micro-average			macro-average		
	P	**R**	**F1**	**P**	**R**	**F1**
PipeLS	**93.78**	87.00	90.11	**93.79**	85.47	89.36
JERES	93.53	**88.00**	**90.48**	93.68	86.25	89.71
$M_C \vert K_{SL}$	-	-	-	93.63	**93.17**	**93.40**
CP	-	-	-	92.27	92.36	91.70

Table 4.4: Experimental results to compare the performance of our system *JERES* for entity extraction based on GM and State-of-the-Art System (SoAS). Bold cases correspond to the best performance.

	Kill			*Live In*			*Work For*		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
PipeLS	69.18	**85.45**	76.46	70.36	66.99	68.63	72.70	71.07	71.88
JERES	69.82	**85.45**	76.85	**72.03**	**68.85**	**70.40**	73.30	**72.57**	72.93
$M_C \vert K_{SL}$	81.10	79.90	**80.50**	71.80	60.70	65.80	**75.50**	70.80	**73.10**
CP	**91.60**	64.10	75.20	66.40	60.10	62.90	73.50	68.30	70.70

	Located In			*OrgBased In*		
	P	**R**	**F1**	**P**	**R**	**F1**
PipeLS	71.94	69.63	70.77	75.95	70.58	73.17
JERES	71.82	**71.11**	**71.46**	**76.92**	**70.80**	**73.73**
$M_C \vert K_{SL}$	**73.50**	64.90	68.90	65.10	69.50	67.20
CP	67.50	56.70	58.30	66.20	64.10	64.70

	micro-average			macro-average		
	P	**R**	**F1**	**P**	**R**	**F1**
PipeLS	72.08	72.00	71.80	72.03	72.74	72.18
JERES	**72.88**	**73.00**	**72.75**	72.78	**73.75**	**73.08**
$M_C \vert K_{SL}$	-	-	-	**73.40**	69.16	71.10
CP	-	-	-	73.04	62.66	66.36

Table 4.5: Experimental results to compare the performance of our system *JERES* for relation extraction based on GM and State-of-the-Art System (SoAS). Bold cases correspond to the best performance.

4.4 Conclusion and Future Works

In this Chapter we proposed two systems to extract entities and relations: the former is a pipeline system, namely *PipeLS*, composed by several binary classifiers and the latter based on a novel probabilistic integration model based on Markov Random Fields, called *JERES*. Experimental assessment on the Roth and Yih data set shows that the *JERES* performance is better than that of *PipeLS* and indeed the *JERES* relation extraction approach overcome the performance of state-of-art system. In the future we would perform the

JERES system on other data set in which a structured knowledge base is defined, as for instance GENIA [49].

Chapter 5

Twitter Sentiment Analysis using Barrier Features

In recent years sentiment analysis over Twitter has acquired great importance offering to companies and organizations an efficient and effective way to understand the audience feelings (negative or positive) towards their brand, products, business etc. A wide range of features have then been proposed for training polarity sentiment classifiers. The design of features is crucial for classification performance, as the classifier can only use the information they convey. In addition to that, features should be designed in such a way that they can be automatically extracted. In this work we consider the introduction in a Twitter sentiment analysis classifier of *barrier features* (BFs), defined in Chapter 2.

Empirical tests show that such approach overcomes published state-of-the-art performance by only using unigrams and bigrams in addition to BFs with a Maximum Entropy classifier.

The main contributions of this Chapter can be summarized as follows:

- a new application of BFs in the Twitter sentiment polarity classification;
- the assessment of this new approach on three available data sets for Twitter sentiment polarity classification.

The goal of this Chapter is the description of the new system for Twitter sentiment polarity classification based on BFs. Experimental results, discussed in Section 5.4 show that the usage of BFs improve the performance of both state-of-art system and a system without BFs.

5.1 Background and Motivations

Sentiment analysis (SA) [87], or opinion mining, is mainly about finding out the thoughts of people from data such as product reviews and news articles. Nowadays this process is very important for both consumers and vendors: the former can look for advices about a product to make informed decisions in the consuming process and the latter are investing increasing attention to on line opinions about their products and services. For these reasons many research communities, including machine learning, data mining and natural language processing, are working to develop methods to extract these opinions/sentiments in an automatic way.

In practice, most methods adopt a two-step strategy for SA [87]. In the subjectivity classification step, the target is classified to be subjective or neutral (objective), and in the polarity classification step, the subjective targets are further classified as positive or negative. Therefore, two classifiers are trained for the whole SA process, one is called subjectivity classifier, and the other is called polarity classifier.

Polarity is an aspect of sentiment analysis which can be faced as a three-way classification problem, associating to each tweet either a positive, negative or neutral polarity. Actually, most approaches only concentrate on the distinction between positive and negative, disregarding the neutral polarity. Many supervised learning approaches to classify positive or negative sentiment are proposed in literature, namely polarity classifiers.

Twitter is a social network based on microblogging service, and its messages, called "tweets", are quite different from other texts like product reviews and news articles. First of all, tweets are very informal, containing a lot of misspelled words, slang, modal particles and acronyms. Because of that and of the fact that length can not be greater than 140 characters, expressions in tweets are often ambiguous. All in all, the characteristics of the employed language are very different from more formal documents and we expect statistical methods trained on tweets to automatically adapt to such specificities and therefore to perform well. On the other hand, tweets are public and therefore a huge amount of unlabeled material can be freely downloaded and employed to build data sets to train unsupervised learning devices. By exploiting such material, we want to reduce as much as possible manual labeling by replacing it with automatic processing. In this way, we aim at constructing systems which can be effectively ported on new domains and also on languages different from English. Of course, we need labeled tweets to train the classifier, but we feel that this is unavoidable because performance of unsupervised approaches in classification would not be acceptable. However, classifier training do not need a huge

quantity of data to be trained and the required labeling can be performed by anyone knowing the tasks, rather than experts in linguistics and knowledge representation.

Most automatic machine learning approaches recently applied to Twitter sentiment polarity classification belong to two streams: some of them try new ways to run the analysis, such as performing sentiment label propagation on Twitter follower graphs and employing social relations for user-level sentiment analysis [103]. Others, not differently from the one we are proposing here, investigate new sets of features to train the model for sentiment identification, such as, for instance, microblogging features including hashtags, emoticons etc. [6, 60]. Indeed, we are proposing to add BFs [3], defined in Chapter 2, to unigrams and bigrams and input them to a Maximum Entropy (MaxEnt) classifier.

A growing request comes from companies and organizations for systems using SA of texts extracted from Twitter to analyze the audience feelings towards their brand, business etc.. Consequently, research efforts have been concentrated on the task and, in the last years, several approaches have been considered. Most of them adopt machine learning techniques, where the input is represented by a feature vector to which a classifier is applied. In such framework, both the choice of the classifier and the feature design are issues to be addressed. Most of the existing approaches exploit three types of features, namely *lexicon features* (based on entry of lexical resource as WordNet [31]), *PoS features* (based on PoS tagging, see the appendix A for more details about PoS tags), and *microblogging features* (e.g. re-tweets, hashtags, emoticons and so on) for sentiment analysis.

In [2] a novel set of features, based on PoS tags, lexicon and microblogging information, is proposed. The feature set is organized in a hierarchy described as follows. It can be split into three subsets depending on the type of the value assumed by features: i) *natural features* have integer values corresponding to counts of occurrences; ii) *real features* have values given by the frequency of use of emoticons included in a dictionary; iii) *Boolean features* correspond to the occurrence/absence of words, exclamation marks and capitalized text. Each of the previous subsets is furtherly divided in two categories: *polar* and *non-polar features*. A feature is polar whenever it is associated to an entry in the English Dictionary of Affect in Language [112] (extended through WordNet [31]); otherwise, it is non-polar. Polar feature values belong to the interval between 1 (negative) and 3 (positive) indicating the degree of polarity of the corresponding word. *PoS features* are defined among both polar and non-polar features as those associated to statistics about PoS tags. Features which are not PoS are included into a generic category *other*. The experimental assessment considers two classification tasks, namely the two-way classification Positive versus Negative and the three-way Positive versus Negative versus Neutral. In both, the best performing features are those com-

bining prior polarity of words with their PoS tags. More in detail, the best performance is obtained when these features are combined with unigrams of words. In other words, they consider lexical information taken from an external knowledge source to integrate unigrams. As better discussed in the following sections, in this feature taxonomy the BFs would be classified as PoS features. Unfortunately we can not directly compare our system performance with the one in [2] as the dataset they used is not available.

While [86, 2] stress the importance of features based on PoS tagging, the approach discussed in [6, 60] emphasizes the use of microblogging features. The results presented in [6] demonstrate that features based on word n-grams introduce a noise in the model due to the large amount of rare words occurring in tweets. So they use microblogging features such as re-tweets, hashtags, replies, punctuations, and emoticons instead. Assessment, performed on a dataset which has not been released, shows that a Support Vector Machine (SVM) classifier trained on the new microblogging features overcame by 2.2% the accuracy of SVMs trained on only unigram features. A similar approach is presented for Twitter sentiment classification in [60] again on an unreleased data set. However, the microblogging features they adopt are different from those in the preceding work and include emoticons, abbreviations and the presence of intensifiers such as all-caps and character repetitions. The best performance is obtained by using n-grams together with microblogging and lexicon features, where words are tagged with their prior polarity as in [2]. In their case, PoS features did not have any positive effect. On the other hand, we used BFs in place of microblogging and lexicon features, obtaining an improvement.

A completely different kind of input representation is introduced in [103] where microblogging features are not directly considered, but part of them (e.g. hashtags and emoticons) are associated to the nodes of a graph together with other information like users, tweets, word unigrams and bigrams. An edge is inserted between two nodes whenever a link between the corresponding labels can be found (e.g., users are connected to tweets they posted on social network; tweets are linked to word unigrams that they contain and so on). Then label propagation is applied to the graph. In particular, sentiment labels are propagated from a small set of nodes seeded with some initial label information throughout the graph. In their experiments, they showed that the label propagation method outperformed a MaxEnt classifier applied to an input which also includes microblogging features. Although their approach obtains an accuracy of 84.7%, this performance has not been estimated on the available Stanford Twitter Sentiment (STS) test set, but only on a portion of it [39]. Direct comparison is therefore not possible because they do not release this smaller test set.

Moreover in [86] another feature-based method is presented to automatically collect a corpus and use it to train a sentiment classifier. The approach considers a multinomial Naive Bayes (NB) classifier and N-gram and POS-tags as features. They use the same unavailable test set as in [39], but the training set is composed by Twitter posts actually crawled and hand-annotated by the authors. Also in this case, direct comparison is not possible because of the unavailability of the test set they used. In addition to that, they consider the three-way classification (neutral/positive/negative) while we only consider positive versus negative polarities.

The other important issue in these systems, namely the choice of the classification approach, is considered in [39], where NB, MaxEnt and SVMs are compared by adopting a standard feature set represented by n-grams of both words and PoS tags. The conclusion of such comparison showed that the best performing combination was composed by the MaxEnt classifier applied to an input represented by unigrams and bigrams of words. In fact, we consider a system based on the same approach as our baseline, referred to as WithOut Barrier Features System (WOBFS).

The approach discussed in [100] introduces semantic information into a system also based on NB classification. This is performed in two ways, namely by interpolation with a unigram model and through a new kind of semantic feature, referring to the abstract concept (e.g. President of United States) corresponding to each entity in a tweet text (e.g. Obama, Bush, Clinton). The value associated to such features gives its correlation with the positive or negative tweet polarity. Assessment was performed on the same data sets we consider in Section 5.3. At the best of our knowledge, their system represents the state-of-the-art, on the three most widely used data sets, namely STS, HCR and OMD. While discussing the results, the authors remark that their approach with semantic features obtains better recall and F_1 in the negative sentiment classification, and better precision but worse recall and F_1 on positive sentiment.

Last, but not least, a novel language model has been very recently proposed in [69], the emoticon smoothed language model (ESLAM), based on a probabilistic framework whose parameters are estimated from manually labeled data. Smoothing considers the noisy emoticon data, which are automatically labeled. The experimental assessment showed that the integration of the two kinds of data in the training set outperformed each of them when applied separately. However, we could not use this data set because authors have not released the randomly chosen test set and the training noisy data.

5.2 Proposed Approach

The approach we are proposing follows the usual framework in two steps, the former constructing a vector of features from the input and the latter applying a classifier. In our case, in addition to word unigrams and bigrams, we are also considering BFs which have been recently introduced for a different task in [3] and described in details in Chapter 2. For the sake of clearness, Table 5.1 reports again the BFs extracted from a tweet taken from the STS data set described in Section 5.3.1. Note that the mention USER derives from the preprocessing step and is deleted before PoS tagging.

(IN, VBP, {PRP, RB})	I firmly believe
(DT, NNP, {IN, PRP, RB, VBP})	I firmly believe that Obama
(DT, NN, {IN, NNP, PRP, RB, VBP})	I firmly believe that Obama /
(DT, NNP ,{IN, NN, NNP, PRP, RB, VBP,}	I firmly believe that Obama / Pelosi
(IN, VBP, {NN, NNP})	that Obama / Pelosi have
(DT, NNP, {IN, NN, NNP, PRP, RB, VBP}	I firmly believe that Obama / Pelosi ZERO
(DT, NN, {IN, NN, NNP, PRP, RB, VBP}	I firmly believe that Obama / Pelosi ZERO desire
(DT, VB, {IN, NN, NNP, PRP, RB, TO, VBP}	I firmly believe that Obama / Pelosi ZERO desire to be
(IN, VB, {NN, NNP, TO, VBP}	that Obama / Pelosi ZERO desire to be
(PRP, VBZ, {"}	It ' s
(DT, NN, {}	a charade
(DT, NN, {}	a slogan
(IN, VBP, {", „ CC, DT, NN, PRP, VBZ}	It ' s a charade and a slogan , but they want
(DT, VB, {CC, NN, PRP, TO, VBP}	a slogan , but they want to destroy
(IN, VB, {", „ CC, DT, NN, PRP, TO, VBP, VBZ}	It ' s a charade and a slogan , but they want to destroy
(DT, NN, {„ CC, NN, PRP, TO, VB, VBP}	a slogan , but they want to destroy conservatism

Table 5.1: *BFs extracted from th following PoS tagged tweet: USER/deleted I/PRP firmly/RB believe/VBP that/IN Obama/NNP //NN Pelosi/NNP have/VBP ZERO/NNP desire/NN to/TO be/VB civil/JJ ./. It/PRP '/" s/VBZ a/DT charade/NN and/CC a/DT slogan/NN ,/, but/CC they/PRP want/VBP to/TO destroy/VB conservatism/NN.*

Also the following example is taken from the STS data set, and shows the BF definition when no endpoint tag precedes the trigger. In this case, all PoS tags from the beginning of the sentence are included in the feature.

$$Now/RB\ \ I/PRP\ \ can/MD\ \ see/VB\ \ why/WRB\ \ Dave/NNP\ \ Winer/NNP\ \ screams/NNS\ \ about/IN\ \ lack/NN\ \ of/IN\ \ Twitter/NNP\ \ API/NNP\ \ ,/,\ \ its/PRP\$\ \ limitations/NNS\ \ and/CC\ \ access/NN\ \ throttles/NNS\ \ !/.$$

Two BFs are extracted from this tweet, namely (DT,VB,{MD, PRP, RB}) and (IN,VB,{MD, PRP, RB}), both corresponding to "Now I can see".

Also in this approach, as in those presented in Chapters 3 4 and as already discussed in Section 2.3 when applying the classifier, we set the BFs at *true* whenever the trigger and the endpoint PoS tags are the same and the PoS tag set is *included* in the one of the

corresponding dictionary feature built on training set. More than one BF can be set at *true* at the same time.

In this Chapter the BFs are not been built using unsupervised dictionary of tweets, but in the future we plan to add it in the system to improve the performance of the classification.

since the BFs are based on PoS tags, input is tagged by using SVMtool [35][1] because it is a SVM-based tagger achieves a very competitive accuracy of 97.2% for English.

5.2.1 System architecture

The global architecture of the approach we are proposing follows the schema depicted in Figure 5.1. All tweets are preprocessed before feature extraction. In fact, they follow a

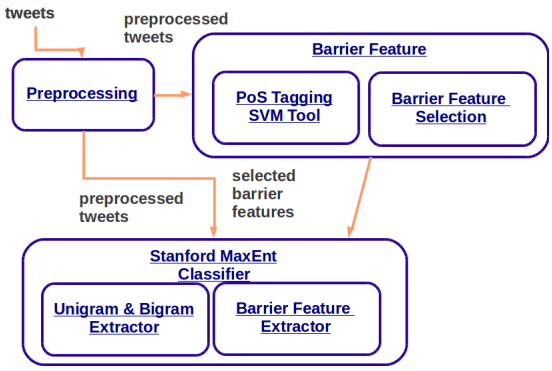

Figure 5.1: Our System Schema for Twitter Sentiment Classification. MaxEnt Classifier is trained with a feature set represented by unigrams and bigrams of word and the BFs.

specific set of conventions, including a maximum of 140 characters for length, the use of emoticons to express mood, of target to mention other tweet users and of hashtags to mark topics. When extracting standard features, which in our case are represented by word unigrams and bigrams, not differently of other works in the field [39], we consider the following preprocessing steps:

1. all emoticons are deleted[2];
2. all HTML entities are removed;
3. all strings composed by three or more repetitions of the same letter are replaced by a string including only two repetitions;

[1] The software can be freely downloaded `http://www.lsi.upc.edu/~nlp/SVMTool/`
[2] Actually, STS dataset comes without emoticons.

4. all URLs are replaced by the token `URL`;
5. all mentions, characterized by the syntax "@⟨ username ⟩" are replaced by the token `USER`;
6. all retweets characterized by the syntax "RT : @⟨ username⟩" are deleted;
7. hashtags, characterized by the syntax "#⟨ some string⟩" are left untouched for unigrams and bigrams.

Hashtags are left because they are consistently used, and therefore behave like content words in normal language. However, tweets are PoS tagged to build BFs. As placeholder tokens such as `URL` or `USER` would not correspond to any PoS, they are deleted before tagging too.

After that, all BFs corresponding to the (endpoint, trigger) in Table 2.1 are collected from the training set together with the number of their occurrences. They are then ranked in descending order of number of occurrences and the top K are inserted in the dictionary. In the assessment described in this Chapter we adopted $K = 50$ as already discussed in Section 2.3.

The Stanford MaxEnt classifier [70][3] is then applied to the feature vector obtained by integrating unigrams, bigrams, and BFs. The choice of this approach is supported by both literature [39] and the results of some preliminary tests not reported in this paper.

5.3 Case studies

This section aims at describing the design of the case study and the datasets used to assess the proposed approach. In the perspective of using our results to realize an actual system for Twitter sentiment analysis, we aim not only at verifying global performance with respect to the state-of-the-art, but also at trying to identify the strength and weakness points of the approach. In particular, we want to minimize the quantity of the training material necessary and to estimate the contribution of BFs to the global results. All in all, we consider the following research questions:

RQ1 Which is the minimum training data size which gives the best performance?

RQ2 Is the performance of the proposed approach better than state-of-the-art on the three considered data sets?

[3] The classifier can be freely downloaded from `http://nlp.stanford.edu/software/classifier.shtml`

RQ3 Which is the contribution of barrier features to the system performance?

Following related works, performance is evaluated by separately computing Precision (P), Recall (R) and F_1 for negative and positive sentiments and averaging the two resulting values.

5.3.1 Data Sets

Experiments are performed on the three following data sets as they are those generally considered for this task:

Stanford Twitter Sentiment (STS) The original Stanford Twitter Sentiment (STS) from [39] [4] is composed by 1.6 million general tweets and has been annotated by using a scraper that periodically queried the Twitter API. The obtained tweets containing positive emoticons, such as *:), :-), :), :D and =)*, have been annotated as positive instances, while the other ones containing emoticons, such as *:(,:-(, or : (*, have been annotated as negative instances. The training set comes with all these emoticons stripped off, though. As we need a development set to answer RQ1 without risking overfitting, we randomly extracted $1,000$ tweet from the training set. The test set adopted by [100] contains $1,000$ tweets, of which 527 are negative, and 473 positive.

Health Care Reform (HCR) The Health Care Reform (HCR)[5] corpus contains the tweets crawled in March 2010 and the hashtag "#hcr"(health care reform) is included in them. This dataset is annotated in [103]. A subset of HCR is labeled with three polarity tags: positive, negative and neutral. Since the focus of our work is the classification of positive and negative tweets, we consider the two-polar dataset released by [100][6]. In fact, this two-polar version of the HCR dataset is composed by $1,354$ tweets, divided 957 negative and positive 397 instances, obtained by merging the training set and the test set provided by [100].

Obama-McCain Debate (OMD) The Obama-McCain Debate (OMD) dataset is composed by 3,238 tweets posted on Twitter in September 2008 during the first U.S. presidential TV debate [101]. Amazon Mechanical Turk is used to obtain sentiment ratings of these tweets where each tweet was classified by one or more voter

[4] The corpus can be freely downloaded at `http://help.sentiment140.com/for-students`

[5] This data can be freely downloaded at `https://bitbucket.org/speriosu/updown/src/5de483437466a9b134ef7c3e886f5b2b0fdf2fff/data?at=default`.

[6] This data has been freely downloaded at `http://www.tweenator.com/index.php?page_id=8` in November 2012

as either positive, negative, mixed, or other which are tweets without any associated rate. Tweets rated by at least three voters with half of the votes being either positive or negative have then been included in the data set[7], which in total includes $1,081$ tweets, split in 393 positive and 688 negative ones. OMD is not divided in training and test set.

5.3.2 Experimental set-up

In this section we describe the different systems involved in assessment. The Barrier Features System (BFS) implements the approach we are proposing. The WOBFS (WithOut Barrier Features System) is identical to BFS, except that BFs are not considered. Thus, its feature vector is composed by unigrams and bigrams of words and it adopts MaxEnt classification. Eventually, the system presented in [100] shows the best performance on all three data sets. It is based on NB classification and a feature set composed by unigrams of words together with a new set of semantic features discussed in Section 5.1. We refer to this system as the SoAS (State-of-the-Art System).

RQ1 requires the definition of a development set and of a series of training sets of increasing size. Thus, it can be addressed only for the STS data set, because the others are too small. In general, we use a protocol based on a training set distinct from the test set, and from the development set when used, only for STS. In this case, training sets are randomly extracted from the set of data available for training. To minimize possible bias connected to the random choice, we repeated each experiment 20 times and averaged performance.

For HCR and OMD, tests are based on a 5-Fold Cross Validation (5-FCV) experimental protocol, where the data set is randomly split into five partitions. Each 5-FCV experiment is repeated 10 times and the results are averaged. Note that in [100] 5-FCV was used only for OMD, while for HCR a split between training and test sets was adopted, but not released. Furthermore the sizes of the retrieved sets are a bit smaller[8] than the ones exploited in experimental phase of [100].

As every experiment is repeated 10 or 20 times, we can compute confidence intervals at 95% to decide whether differences with the SoAS are statistically significant. On the other hand, when we can run both systems we are comparing, such as for RQ3, we apply approximate randomization to evaluate statistical significance.

[7] This data have been freely downloaded at `http://www.tweenator.com/index.php?page_id=8` in November 2012.

[8] 655 instead of 839 for training set and 699 instead of 839 for test set

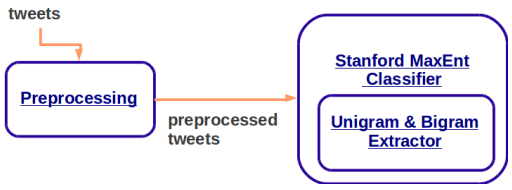

Figure 5.2: Baseline System Schema, called WithOut Barrier Feature (WOBF). In this system, after the preprocessing step, a MaxEnt classifier is used by adopting a feature vector of unigrams and bigrams of words.

RQ3 aims at estimating BF contribution and therefore we compare the performance of our system with a system which is identical except for this point. However, as discussed in Section 5.1, our baseline system WithOut Barrier Features (WOBF) is implemented by following [39], that is a MaxEnt classifier applied to a feature set including unigrams and bigrams of words. WOBF is therefore composed by three modules, namely preprocessing, feature extraction and sentiment polarity classification, as depicted in 5.2.

5.4 Results

In this section the experimental results are reported and discussed for each research question.

5.4.1 Barrier Features contribution

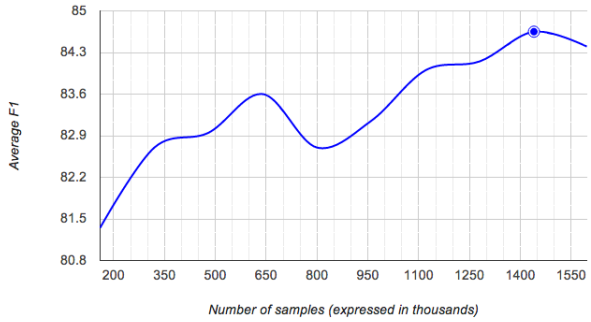

Figure 5.3: Learning curve for STS on the development set.

As discussed in the preceding section, the minimum training set size necessary to obtain the best performance has been chosen on a development set, to avoid overfitting. From the resulting learning curve, plotted in Figure 5.3, after a first maximum between 60K and 65K, F_1 saturates at a size of about $1,440$K. Therefore, the training set adopted for BFS has this latter size.

Table 5.2: *Experimental results to compare the performance of our system and State-of-the-Art System (SoAS). Bold cases correspond to the best performance, while symbol ‡ indicates statistical significance of the comparison, namely that the SoAS performance is external to the confidence interval at* 95% *of BFS performance.*

Data Set	System	Positive Sentiment			Negative Sentiment			Average		
		P	R	F1	P	R	F1	P	R	F1
STS	SoAS	**85.80**	79.40	82.50	82.70	**88.20**	85.30	84.25	83.80	83.90
	BFS	84.05‡	**89.06**‡	**86.48**‡	**89.56**‡	84.73‡	**87.08**	**86.80**‡	**86.90**‡	**86.78**‡
HCR	SoAS	53.80	**47.20**	50.30	**84.50**	87.60	86.00	69.15	67.40	68.15
	BFS	**71.62**‡	46.82	**56.61**‡	80.71‡	**92.29**‡	**86.11**	**76.16**‡	**69.56**	**72.71**‡
OMD	SoAS	68.90	**75.60**	71.70	**87.10**	82.40	82.20	77.65	79.00	78.20
	BFS	**82.13**‡	74.25	**77.99**‡	86.05‡	**90.77**‡	**88.35**‡	**84.09**‡	**82.50**‡	**83.16**‡

Table 5.3: *Experimental results to evaluate the BF contribution. Bold cases correspond to the best performance. Symbol † indicates that the* ρ*-values, computed by approximate randomization, are smaller than* 0.05 *level.*

Data set	System	Positive Sentiment			Negative Sentiment			Average		
		P	R	F1	P	R	F1	P	R	F1
STS	WOBFS	83.71	88.60	86.09	89.14	84.42	86.67	86.42	86.51	86.38
	BFS	**84.05**†	**89.06**†	**86.48**†	**89.56**†	**84.73**	**87.08**†	**86.80**†	**86.90**†	**86.78**†
HCR	WOBFS	69.57	45.51	55.02	80.23	91.73	85.59	74.90	68.62	71.62
	BFS	**71.62**†	**46.82**†	**56.61**†	**80.71**	**92.29**†	**86.11**†	**76.16**†	**69.56**†	**72.71**†
OMD	WOBFS	81.42	70.79	75.73	84.30	89.62	86.88	82.86	80.20	81.30
	BFS	**82.13**†	**74.25**†	**77.99**†	**86.05**†	**90.77**†	**88.35**†	**84.09**†	**82.50**†	**83.16**†

Table 5.2 compares BFS's performance on the test set with state-of-the-art. Average performance is always significantly better for BFS, with the only exception of recall for HCR, where there is not statistical significance. Even when considering that for HCR the comparison involves two different protocols, namely training and test sets for SoAS and 5-FCV for BFS, the improvement in terms of F_1 for BFS is quite important. Moreover, when we separately consider positive and negative sentiments, F_1 is never worse for BFS than SoAS, although in some cases precision or recall can show a discording trend. All in all, we can conclude that on these data sets BFS has state-of-the-art performance.

However, for STS, SoAS is trained on a much smaller training set, composed of only 60K tweets [100]. As they did not release the training set they used, we trained 10 systems on training sets composed by 60K tweets randomly selected from the initial training data and averaged the performance. BFS performed $F_1 = 82.39 \pm 1.87$, with a confidence interval at 95%, to be compared with $F_1 = 83.90$ for SoAS. Also in this case, then, our system performance is not significantly worse than the SoAS, even if it employs a significantly smaller knowledge based contribution.

To answer RQ3 we estimate the contribution of BFs by comparing performance on each data set of the system with and without them. Results, reported in Table 5.3, show how the barrier features contribution always improve performance both in terms of precision and recall, and therefore also in F_1. Furthermore, such improvement is statistically significant in all but one case.

5.5 Conclusion and Future Works

Feature design is crucial in any classification problem as it represents the means for introducing in the system the necessary information. On the other hand, automatic construction of this information greatly improves the applicability of the approach. We therefore explored the effectiveness of BFs for sentiment polarity classification in Twitter posts and we showed on three different data sets that they can be very effective to obtain a classification system which attains state-of-the-art performance with minimum need of manual intervention.

Indeed, only the pairs (target, endpoint) should be designed by an expert, but in fact they seem to be very intuitive and even more importantly, they depend on the language rather than on the task. We plan to explore whether a different choice of these pairs gives better performance, and to find criteria to design such pairs also for languages different from English.

Interestingly, two of the considered data sets are quite small. Therefore BFs do not need a large training set to attain good performance. This is not contradicted by the STS case, where, even if a better accuracy was obtained by using more training data, the system shows a competitive accuracy also with a small training set.

In the future we plan to extract the BFs using an unsupervised dictionary, built employing unlabeled Twitter posts taken from the same domain, as in our approach for the entity and relation classification described in tha Chapters 3 and 4.

Chapter 6

Training Natural Language Parsers to improve Concept Location

Identifier names play a key role in program understanding and in particular concept location. Programmers can easily "parse" identifiers and understand the intended meaning. This, however, is not trivial for tools that try to exploit the information in the identifiers to support program understanding. To address this problem, we resort to natural language analyzers, which parse tokenized identifier names and provide the syntactic relationships (dependencies) among the terms composing the identifiers. Such relationships are then mapped to semantic relationships.

In this Chapter, we have evaluated the use of off-the-shelf and trained natural language analyzers to parse identifier names, extract an ontology and use it to support concept location. In the evaluation, we assessed whether the concepts taken from the ontology can be used to improve the efficiency of queries used in concept location. We have also investigated if the use of different natural language analyzers has an impact on the ontology extracted and the support it provides to concept location. Results show that using the concepts from the ontology significantly improves the efficiency of concept location queries. The results indicate that the efficiency of concept location queries is not affected by the differences in the ontologies produced by different analyzers.

The main contributions of this Chapter as compared are:

1. An approach to train natural language analyzers for use with identifiers.
2. A comparison among the different natural language analyzers investigated in this work in terms of differences between the extracted ontologies in addition to their support to concept location.

In the following Sections we describe: the different types of natural language analyzers used in this study, the mapping of natural language dependencies and ontological relations are presented, the steps involved in the concept location task. Section 6.5 presents the case study, including procedure, results and discussion to compare the different natural language analyzers in terms of differences between the extracted ontologies

6.1 Background and Motivations

During program understanding, source code exploration in search for a specific concept is a typical activity. One of the key source code elements which affects this activity is identifiers. Identifiers serve as a link between the intention of a concept and its extension in the source code ([94]). Different approaches which take advantage of this fact have been proposed by various authors to improve code search and support program understanding (see [72, 48, 33, 1, 96] and [102]).

The intention of a concept in an identifier is reflected in the terms chosen, their relative order and the relationships among them. For example, an identifier name constructed from the terms *record* and *event*, might convey two different meanings to the reader when the terms are used in a different sequence: *recordEvent*, which might mean logging an event, and *eventRecord*, a type of record. While this difference is easy to grasp for a human, it is not obvious for tools that are designed to support program understanding. To address this problem, we resort to natural language dependency analyzers (see Section 7.3 for details about dependency parsing), which can be used to extract the semantic relationships among the identifier terms.

Natural language dependency analyzers present the syntactic relationship between the identifier terms through natural language dependencies (e.g. *noun-specifier* or *direct-object*) among them. Though these relations are syntactic, very often the terms they connect are semantically related. Of course this is not always the case, and, for example, the relation between a determiner and the corresponding noun is not semantically relevant. However, the relation between a verb and its object usually has a semantic nature. Therefore, we focus on some categories of the dependencies extracted from the analyzer in order to obtain relevant semantic relations for our aims.

In the work ([1]), such information has been exploited to extract an ontology from identifiers and support program understanding. The extraction of the ontology is conducted by parsing sentences constructed from identifiers. In this work, in addition to the natural language dependency analyzer used in [1], we have considered analyzers which are

trained on a training set that closely resembles the structure of identifiers. The training set is constructed from sentences and phrases automatically extracted from the textual documentation of the system which comes with the source code and is available on-line. The documentation we considered consists of source code comments, user manuals, system documentation, and FAQs describing howtos, when available. The extracted sentences and phrases are automatically converted to identifiers that resemble true code identifiers. This is achieved by applying a predefined set of rules for identifier construction from natural language sentences. For these artificially constructed identifiers we know the correct parse trees, obtained from the parsing of the original sentences or phrases. Hence, we can use them as a training set.

The natural language dependency trees generated by the analyzers are used to build ontologies, which can support program understanding. Based on the level of formality, an ontology can vary from a simple taxonomy with almost no formalization, to one which uses a rigorously formalized theory (see [108]). Ontology in our context is a "lightweight ontology" which is in between these two extremes and does not include axioms supporting formal reasoning, but only considers concepts and relations connecting the concepts. A lightweight ontology which is built using only concepts and relations connecting the concepts without any formalization is sometimes referred to as "concept map". In this Chapter we refer to such lightweight ontology simply as *ontology*. This definition is opposite to that given in Chapter 4 in which the axioms supporting formal reasoning were very important in the definition of probabilistic integration model for jointly entity and relation extraction.

In this study, we have assessed the benefits of using ontologies extracted from identifiers in several subject applications and we have investigated the impact of using different analyzers to generate the ontologies. The assessment was conducted in the context of a program understanding task, namely *concept location*, which uses queries to narrow down the search space and identify the parts of a program that implement a concept of interest.

Concept location/assignment problem as described by [9] is a problem related to discovering human oriented concepts and assigning them to their implementation instances within a program. In the literature various approaches which exploit different information such as dynamic and textual information are proposed to address this problem. A comprehensive survey of the approaches can be found in [28]. In this section, we discuss approaches which exploit textual information in the source code to facilitate/improve concept location.

[72] and [33] used information retrieval (IR) based approaches to reduce the effort required to understand and locate the part of the source code that needs to be changed. In their approach, they used Latent Semantic Indexing (LSI) to convert source code documents (composed of identifier terms) and user query to their respective mathematical representations. Such formalizations are then used to compute the similarity between them and get a ranked list of source code documents (by decreasing similarity). The results of these approaches are dependent on the quality of user queries. To assess the quality of queries priory to using them and reduce the effort and time required to assess the results, [44, 45] have proposed query assessment metrics. The metrics are used to evaluate and classify the query as *high-performing query* and *low-performing query* prior to its execution. [21] have proposed an approach to expand queries using a language model. The approach uses information flow and term co-occurrence information in the system documentation to identify terms which can be used to expand the queries.

To reduce the developers' effort in locating concepts using IR techniques, [92] have combined Formal Concept Analysis (FCA) with LSI. The approach produces a concept lattice using the most relevant attributes (terms) selected from the top n ranked documents (methods). The evaluation of their approach has shown that the concept lattice is effective in grouping relevant information. [93] have integrated the Google Desktop Search Engine[1] into Eclipse, to take advantage of the engine's features and to facilitate searching of the source code. [40] have also proposed automated concept location using Independent Component Analysis (ICA). In this approach, the authors use ICA to identify statistically independent signals which correspond to concepts. The concepts are then mapped to methods which are related in functionality.

In other works, parts-of-speech (PoS) of the terms composing the identifiers and natural language dependencies among them are used to extract information from the source code and improve the quality of the queries formulated to locate a concept. [48] have generated noun, verb and preposition phrases from the signatures of program elements to support context based (re)formulation of queries which are used to search the source code. [102] have exploited the action oriented identifier graph, which is a natural language representation of the source code ([32]), to expand user queries and locate a concept. [10] have employed templates to improve identifier name PoS tagging (see the appendix A for more details about PoS tags) and have defined rules to improve the structure of field names. In the work ([1]), they have used the PoS of the terms and the natural language dependencies to extract an ontology from the source code. The ontology is then exploited to expand the user queries which are used for concept location. In this work, we have extended it by

[1] http://googledesktop.blogspot.com/

considering three types of natural language analyzers, including analyzers that require training. To support such training, we have developed a novel approach, presented in this Chapter. Using this approach we can train a natural language analyzer so that it directly works on identifier term lists, instead of artificial sentences constructed from identifier term lists. In the present work we also extend the experimental evaluation of the approach with additional subject systems, more research questions and additional metrics and data collected.

In [89] authors have proposed an approach where programmer queries used in concept location are (re)formulated using the knowledge acquired while manually refining fragments of an ontology related to the concept being searched. Our work is similar to theirs in that we both utilize ontologies to reduce the search space. However, the ontology they used is manually constructed while ours is reverse engineered from the identifiers of the source code using Natural Language Processing (NLP) techniques. [96] have also proposed an approach to extract a domain specific ontology from a graph based representation of the public interfaces of similar APIs. To extract the domain specific ontology, they have used a graph matching algorithm. The main difference from our approach is that we aim at producing a project-specific ontology, while Ratiu et al. try to generate a cross-project domain ontology. As a consequence, we rely on completely different techniques.

6.2 Identifier Parsing

Identifier names are one of the ways in which developers' communicate the intention of the source code, by representing it with a carefully chosen name. To extract and exploit this information, we use NLP techniques (see [1]). The approach uses a natural language dependency analyzer to retrieve a set of dependencies between the different tokens composing each identifier name. These dependencies are then used to identify ontology concepts and relations among them. For the sake of clarity, in the following the term *analyzer* refers to the tool taking an input sentence, that is a string of words, and returning a syntactic analysis, while the word *parser* only refers to the last part of this analysis which takes an input sentence tagged with Parts-of-Speech (PoS). As we will discuss in the following, the parser can also take as input a set of tagged hypotheses and then choose among them. The output of the analyzer can be hierarchically organized phrases, if a constituency based approach is adopted, or a set of dependencies between word pairs composing a directed graph in case of a dependency based approach. In the latter case, by *dependency* relationship we mean an asymmetric binary relationship between a token called *head*, and another token called *modifier* (see [68]).

While natural language analyzers are mainly conceived to work with full sentences, the term lists which are obtained by splitting identifiers are different from sentences. In the work ([1]), they have presented some heuristics to convert an identifier term list into a sentence, so that it can be handled by a general purpose parser which we call as Standard English Analyzer (SEA). In the present work, in addition to the analyzer used by [1], we consider an analyzer constructed to directly work on an identifier term list.

In the following sub-sections, we describe the syntactic analysis approaches followed in our study with the corresponding analyzers and training sets used. The summary of the steps involved in the construction of the sentences used in the work ([1]) are also described below.

6.2.1 Syntactic analysis

In NLP it is well known that a syntactic analysis is necessary to reconstruct the meaning of the input sentence. In our case, the relations between the entities (concepts) included in the identifier term list depend on the syntactic and semantic role of each token. A syntactic parse is therefore necessary for further processing.

The construction of the syntactic analysis for an input identifier can be performed in different ways and requires several steps. The first step in all cases, however, is *tokenization*. *Tokenization* is the process of splitting a text into words or linguistic elements called *tokens* or *terms*. Identifier names are composed of one or more terms. In order to identify the composing terms and tokenize identifier names, we have taken advantage of the commonly used term separators, such as camel casing (e.g. *FileItem*) and underscore (e.g. *file_item*). This can also be achieved using more sophisticated techniques proposed in [63]. When the terms used to construct the identifiers are abbreviations or contractions, they can be expanded using the approaches described by [61, 64] and [47]. For example, by tokenizing the identifier name *fileItem*, we get the term list <file, item>.

In our approach, the syntactic analysis includes two modules: *PoS tagging* and *parsing*. The former assigns a label corresponding to the function of the word in the sentence, such as *noun, verb*, and so on, to each token, while the latter constructs a syntactic analysis of the whole sentence. In our case we consider a *dependency parser* where the analysis is formed of dependencies between pairs of input words. The *dependency parser* is chosen over the *constituency parser* because it allows a more direct reconstruction of relations between concepts.

The two modules can be organized to work in a *pipeline* or *integrated*. While in the pipeline analyzer PoS tagging is completed before the syntactic parse is built, in the integrated schema a list of possible PoS tags is associated to each token in the input by consulting the lexicon, and the choice of the best PoS tags is performed during parsing. In other words, in the integrated analyzer the parser is also involved in the decision of the PoS tag which is more likely in the considered sentence.

While in the work ([1]) a complete English sentence was constructed from the tokenizer output and used as an input to a Standard English Analyzer (SEA), in the present approach we also consider the string of tokens produced by the tokenizer as a direct input to the analyzer. To directly process the sequence of tokens, three different NLP systems are considered. These systems use a data-driven natural language parser which requires a training phase to learn how to process the input text. The advantage of such data-driven parsers is that they can easily learn how to parse different (novel) languages and their variants from a collection of suitable parse trees, called *treebank*.

The first NLP system, which is applied to the token sequence, is similar to the analyzer of the standard English, SEA. Its two analyzer modules are trained on a largely employed English treebank, namely the PennTreebank (see [73]). Both remaining NLP systems involve retraining the two analyzer modules to adapt them to the identifier language, and only differ in the architecture, being *pipeline* or *integrated*.

Training is performed on an annotated set which should be as similar as possible to the actual input set, hence, in our case, to tokenized identifiers. Indeed, the string of tokens extracted from an identifier is very different from a natural language sentence as identifiers usually do not correspond to complete sentences. In addition to that, they also have a different structure depending on their function: method names are more likely to describe actions and therefore their structure resembles the Verbal Phrases (VPs), while attribute and class names usually aim at indicating things, in a way similar to Noun Phrases (NPs). This is, for example, the case of the two examples reported in Figure 6.1 which shows the analysis of two identifiers: *TextPanel* and *removeFile*. The former corresponds to a class identifier while the latter to a method name. Such distinction is expected to affect the syntactic analysis, but not the PoS tagging. Therefore, we will consider a unique PoS tagger, but a different parser for each of the two classes of identifiers, namely a VP-parser for method names and an NP-parser for all the others (classes and attributes). Consistently, we construct two different training treebanks, one for each parser. In Section 6.2.3, we present details of the training set construction.

All in all we therefore consider four NLP systems:

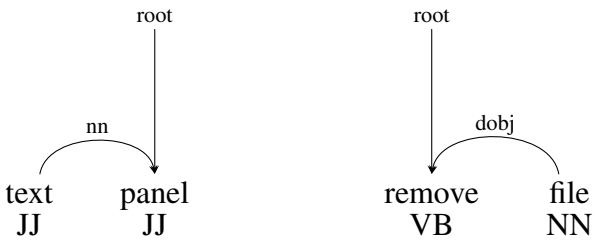

Figure 6.1: Examples of dependency analysis for two identifiers: (a) the class name composed by two tokens, **text panel**, and (b) the method name formed by two tokens, **remove file**. Each graph node is labeled by one token of the identifier and the PoS tag assigned to the token. The edge labels are dependency relationships extracted by the analyzer, namely **dobj** - direct object and **nn** - noun-noun specifier.

1. *Untrained Integrated Analyzer (UIA)*: consists of an SEA integrating PoS tagging and dependency analysis, applied to complete sentences built by padding the token sequence produced by the tokenizer. The system architecture is shown in Figure 6.2;

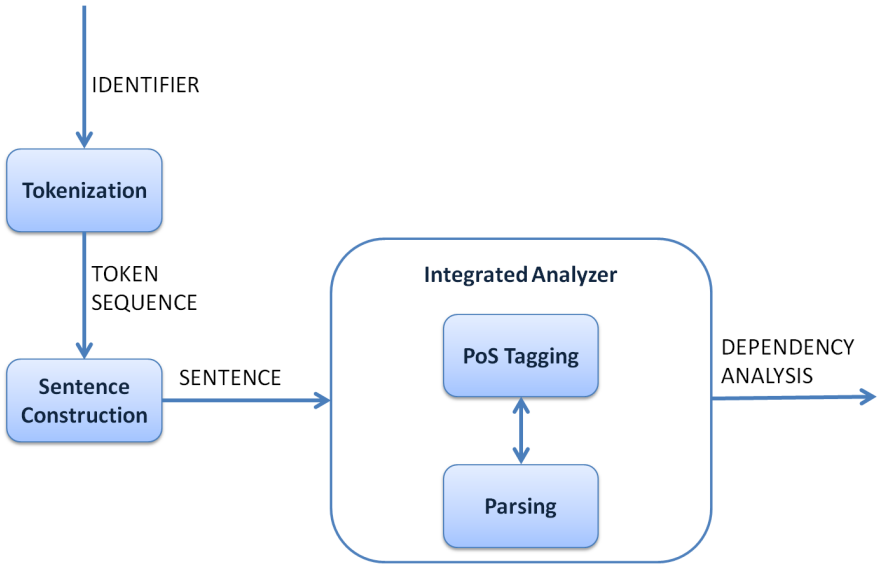

Figure 6.2: Architecture of Untrained Integrated Analyzer (UIA).

2. *Untrained Pipeline Analyzer (UPA)*: to overcome the need for complete English sentences, a pipeline composed of standard English PoS tagger and parser is directly applied to the token sequence extracted from the input identifier: the analysis in this case is obviously more difficult than in the previous case and a more accurate analyzer is required. The pipeline architecture is depicted in Figure 6.3;

3. *Trained Pipeline Analyzer (TPA)*: both syntactic analysis modules, namely PoS tagger and parser, are retrained to adapt them to the token language and then they are combined in a pipeline and directly applied to the tokenizer output (see Figure 6.4).

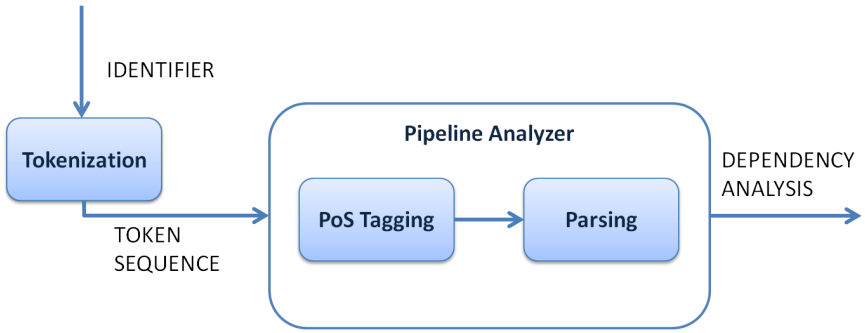

Figure 6.3: Architecture of Untrained Pipeline Analyzer (UPA): the two analyzer modules are applied as distributed, without retraining, and in a pipeline.

Note that in this case two different parsers (i.e. VP and NP) are used, depending on the function of the input identifier;

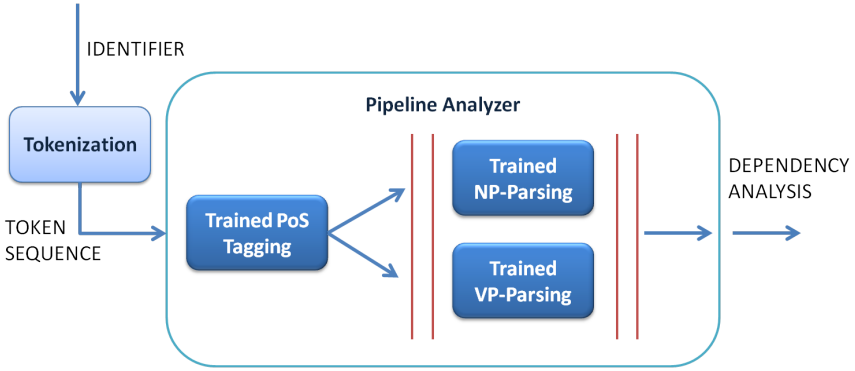

Figure 6.4: Architecture of Trained Pipeline Analyzer (TPA): a pipeline of the two retrained modules is directly applied to the tokenizer output.

4. *Trained Integrated Analyzer (TIA)*: this system is identical to the previous one, except that the two syntactic analysis modules are integrated together to improve robustness towards PoS tagging errors, as depicted in Figure 6.5.

The sentence construction module is adopted only in the UIA, while in all other cases the analyzer is modified to directly process the tokenizer output. In [1] the steps involved in the sentence construction of UIA is described in details.

6.2.2 Syntactic analyzers

As our analyzers, we use two tools, namely *Minipar* which has an integrated PoS tagger and the *Malt* parser, which we employ together with the *SVMTool* PoS tagger. Minipar is used in UIA while Malt/SVMTool are used in UPA, TPA and TIA. Minipar is quite robust with respect to natural language variability but is available as is, and can not be

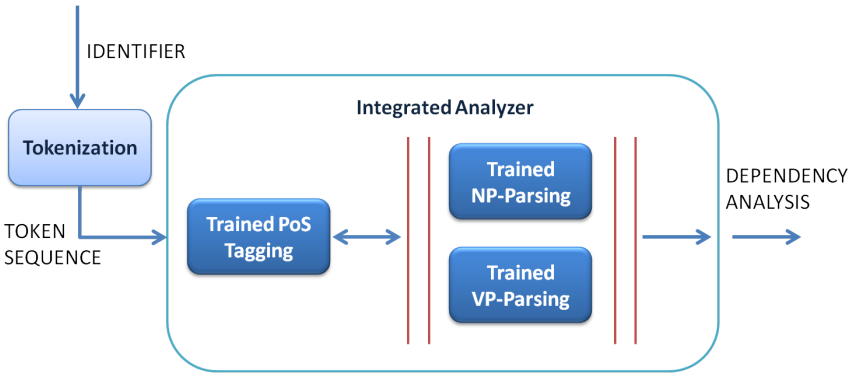

Figure 6.5: Architecture of Trained Integrated Analyzer (TIA): the two modules are re-trained and applied in an integrated modality.

adapted in any way to new tasks. Since we aim at adapting the analyzer to identifier analysis, we consider the combination of the latter two state-of-the-art tools: Malt parser and SVMTool. Malt parser and SVMTool are based on data-driven NLP approaches. We have applied them both in their standard English version, referred to as *untrained*, and after re-training on a text which is similar to the token sequences generated from identifiers. Details of these tools are presented in the following sub-sections.

Minipar

Minipar[2] is a broad-coverage principle based parser for the English language (see [67]), in which the grammar is represented as a network. It adopts an integrated strategy: a list of possible PoS tags is associated to each word in the lexicon and the resulting tag is chosen during parsing. After parsing a sentence, Minipar outputs information about the individual components of the sentence and the structural relation between such components, including their mutual dependencies. In addition to specifying the relationship between terms, Minipar labels each term with one of the PoS based on its role in the sentence.

The PoS which are of interest to us, to extract concepts and relations, and to build the ontology, are nouns (N), verbs (V) and adjectives (A). Minipar generates a list of tuples. Each tuple provides information about the term, w, represented by the node, its category (N, V, A, etc.), the head (*root*) term it modifies, and the dependency relationship between the modified term (the head) and w (see [67]). In this study we are mainly interested in the dependency relations between verbs and their respective objects, and the nouns and their modifiers. The former dependency relation is referred as object relation (*obj*)

[2] http://webdocs.cs.ualberta.ca/ lindek/minipar.htm

in Minipar, while the latter as a modifier (*mod*) or noun-noun specifier (*NN*) relation. Figure 6.6 shows a graphical representation of the tuples generated by Minipar for the sentence *Subjects get size*.

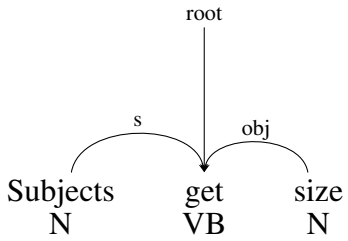

Figure 6.6: Parse tree for ***Subjects get size*** which is generated using SEA (Minipar). The edge labels are dependency relationships extracted by the analyzer, namely **obj** - object and **s**, subject.

SVMTool PoS tagger

The PoS tagger SVMTool presented in [35] is based on Support Vector Machines (SVMs) by [109], a machine learning approach largely adopted because of its good performance on a large set of tasks. The SVMtool tagger for standard English achieves a very competitive accuracy of 97.2%, as reported by [34]. However, the system can be efficiently trained to be adapted to different languages. In fact, SVMTool is composed of two modules: SVMLearn and SVMTagger. The former is used to train the models and it is based on SVM^{light}, a library for SVM implemented by Thorsten Joachims. The latter is used to tag the input sentences, and in our case it is applied to tokenized sentences. In the UPA NLP system, the standard English tagger is applied to the token sequence output by the tokenizer, while in the TPA and TIA systems it is applied after training the models on the training set, as discussed in Section 6.2.3.

Malt parser

Malt parser[3] ([84] and [82]) is a data-driven dependency parser, which, similarly to SVMTool, can be applied either with a standard English model or trained on a training set. In this case, however, the training set is represented by a collection of sentences annotated with the corresponding analysis, called *treebank*. While Minipar also includes a PoS tagger, the input to Malt parser must be tagged. The final output of Malt parser is quite similar to that of Minipar.

[3] Malt parser can be freely downloaded from `http://maltparser.org/download.html`

6.2.3 Training

In NLP tools based on machine learning, a model is learned from a training set and then it is applied to the input. This is the means we employ in this work to adapt generic natural language tools to process the list of tokens extracted from an identifier. The crucial point in this approach, however, is the construction of a training set which can describe the task at hand. Each training set should be collected from a domain as similar as possible to the one considered, and then annotated with the information necessary for the model. In our case, annotation should include both PoS tagging and dependency analysis. We looked for collections of identifiers available together with their analysis (PoS tags and parse trees). The closest data we got is the class identifier data set built by [17]. It is a treebank which has been employed for class and attribute names. This data set contains 120,000 class identifier names extracted from 60 Java open source projects. It has been used in [17] to understand the Java class identifier naming conventions used in practice. Since the first step of training is identifying grammatical patterns in names based on a part of speech (PoS) tagging, we use the identifier names of this data set, which are already tokenized and tagged using the Stanford Log-linear PoS tagger[4].

As larger training sets are usually better than smaller ones, a good training set is obtained as a trade-off between the need for a large amount of data and the requirement that such data accurately describe the task at hand. Unfortunately, manual annotation is a very expensive process, and therefore it is very difficult to obtain large training sets for tokenized identifiers. Furthermore no large collection of program identifiers annotated with PoS tags and associated with the respective parse trees is publicly available. We have therefore designed automatic procedures to construct the necessary annotations without manual intervention. We used natural language texts available from the documentation of the considered software projects to build our training sets. Such documentation typically includes comments extracted from the source code and user manuals.

As already mentioned above, while only PoS annotations are needed to train the PoS tagger, for the parsers we need a *treebank*. In a similar way to syntactic parsers, treebanks can also follow the *constituency* or *dependency* framework. The latter suits well ontology construction, as it builds dependency relations between words. On the other hand, the constituency approach produces a sentence parse tree, which is easier to transform to obtain the kind of simplified sentences corresponding to identifiers. We have used both approaches during the construction of the training sets for both the PoS tagger and the dependency parser. In fact, the transformations necessary to build a potential identifier from

[4] http://nlp.stanford.edu/software/tagger.shtml

a natural language sentence are more intuitively expressed in the constituency framework. Eventually, the so obtained constituency treebank has been transformed into an equivalent dependency one, necessary to train Malt parser.

The construction of the training set by means of transformations applied to a natural language treebank has also the advantage of allowing a stronger adaptation to the considered software system. Indeed, it can automatically be applied to any natural language description of the system, such as comments and documentation. While designing the transformations to be applied to these texts in order to simulate identifiers, we have considered the fact that identifiers have different structures depending on their function. For example, method names are more likely to describe actions and therefore their structure resembles VPs, while attribute and class names usually aim at indicating things, in a way similar to NPs. Such distinction is expected to affect the syntactic analysis, but not the PoS tagging. Therefore, a unique training set including all sentences is considered to train the PoS tagger, while two different, disjoint training sets (VP-like sentences for method names and NP-like sentences for class/attribute names) are considered to train two parsers, a VP-parser and an NP-parser.

First of all, the natural language sentences available from the project documentation are PoS tagged using SVMTool with the language model for the standard English distributed with the tagger. Afterwards a constituency parser, namely the Stanford parser discussed later in Section 6.2.3, is employed to build the constituency parse trees of each sentence. Although automatic PoS tagging and parsing can introduce errors, it is very cheap and the error rates of both tools are low enough to be sure that the introduced errors will not deteriorate too much the resulting treebank. All determiners are then deleted from the treebank, since no determiner is included in identifiers. Although a similar transformation could also be applied to other PoS tags, only this one resulted to be effective in some preliminary tests. This is probably due to the fact that the other infrequent PoS tags are nearly absent from the simplified text which we use for training the parser. However, the deletion must be performed in such a way that a consistent parse tree is produced even after the transformation. An important property of parse trees is that only leafs are labeled with PoS tags. Therefore, after deletion, every internal node must still have one or more children.

As noted before, we aim at obtaining two different parsers, namely a VP-parser to apply to method identifiers and the NP-parser for class/attribute identifiers. Therefore, we need two different training sets, one containing only parse trees of VP's and the other of NP's. As identifier structures are usually quite simple, we also impose that all NP's and VP's composing our training treebanks are minimal in the sense that they do not contain any

other subtree with the same root. Such trees are called *non-recursive*. Then, all non-recursive VP subtrees are collected from the parsed project documentation to form the VP training set, while all non-recursive NP subtrees form the NP training set. Eventually, the parse trees are converted into equivalent dependency graphs, used to train the data-driven dependency parser

Eventually, we use the whole treebank to train SVMTool, while each of the two treebanks is used to train the VP-parser and the NP-parser respectively. The so obtained modules are then introduced in the two trained NLP systems, namely TPA and TIA, as showed in Figures 6.4 and 6.5 respectively.

Stanford parser

The constituency parser used for the construction of the training treebank is the Stanford parser ([58, 59]) with the English grammar distributed together with the software. It is based on probabilistic context-free grammars whose probabilities are estimated during training and used during parsing to output the most probable derivation with the Viterbi algorithm. This package is implemented in Java and can be freely downloaded from `http://nlp.stanford.edu/software/lex-parser.shtml`. To build the training treebanks for NP-parser and VP-parser we use another tool distributed by the Stanford lab, namely the Tsurgeon[5], a tree transformation tool which maintains the consistency of parse trees when non-recursive subtrees are extracted.

6.3 Ontology extraction

The ontology of a program is extracted by exploiting the linguistic information captured in the program identifiers. The *concepts* of the ontology are retrieved from the nouns or noun phrases referred in the parse trees of identifiers, or, in some cases, directly from the class or program names. The ontological *relations* are obtained from the natural language dependencies found in the parse trees of the identifiers and the verbs used in method names. In our study, we have defined four types of ontological relations which are used to show the semantic relationships among the concepts in the extracted ontology. Below we describe each ontological relation in detail. Our examples are taken from the case studies and the parse trees are represented using an XML notation. In the XML notation, dependencies are specified as the role attribute of the XML nodes of the parse trees.

[5] The tool can be freely download from `http://nlp.stanford.edu/software/stanford-tregex-2012-07-09.tgz`

For instance, *subj* indicates the subject, *dobj* (Malt parser, used in UPA, TPA and TIA) or *obj* (Minipar, used in UIA) the direct object, *det* a determiner and *nn* a noun-noun specifier.

***isA* ontological relation:** a relation holding between a specific and a general concept. This relation is mapped to *nn* (*noun noun*) or *mod* (*modifier*) natural language dependency relations found in the parse trees generated by UIA (Minipar), while when the parse tree is generated using UPA, TPA and TIA (Malt parser), it is mapped to *nn*, *amod* (*adjectival modifier*) or *partmod* (*participial modifier*) natural language dependencies. In both cases, the concepts extracted and connected by this relation are the root noun (the most general concept) which is modified/specified and all the descendant (more specialized) sub-concepts, which are obtained by incrementally adding all specifiers/modifiers down the parse tree. For example, from the UIA and TIA parse trees shown in Figure 6.7, we can extract the ontological relation *isA(text panel, panel)*, using the natural language dependency *nn*.

```
<root>
  <C id="E0" pos="0">
    <VBE id="3" pos="3" role="i" phrase="is" base="be">
      <N id="2" pos="2" role="s" phrase="panel">
        <N id="1" pos="1" role="nn" phrase="text"/>
        </N>
      <N id="5" pos="6" role="pred" phrase="thing">
        <N id="E2" pos="4" role="subj" base="panel"
          antecedent="2"/>
        <Det id="4" pos="5" role="det" phrase="a"/>
      </N>
    </VBE>
  </C>
</root>
```

```
<root>
  <NN id="2" pos="0" role="null" phrase="panel">
    <NN id="1" pos="2" role="nn" phrase="text">
    </NN>
  </NN>
</root>
```

Figure 6.7: Parse trees generated by UIA (top) and TIA (bottom) for the sentence ***text panel*** *is a thing* and ***text panel***, respectively. ***TextPanel*** is a JEdit class identifier.

***<verb>* ontological relation:** a context-specific relation holding between a concept, usually the doer, and the object on which the verb acts. This relation is mapped to a verb term which is found in the parse tree generated for a method name. The concepts involved in the relation are those represented by the class name and the object of the verb, if this exists. Otherwise, the relation is between the concept represented by the program name and the class name (i.e., the object of the action is considered to be the implicit `this` parameter of the method invocation). When it exists, the object of the verb is referred by UIA using the *obj* (*object*) dependency relation while UPA, TPA, and TIA refer to it us-

ing either *dobj* (*direct object*) or *pobj* (*preposition object*). This ontological relation is not extracted if the verb is an accessor (`get` or `set`). In method names, the most frequently appearing prepositions are *on* and *to*. *On* is usually used to mean *handle* and refers to an event, while *to* usually means *convert to*. Hence, we map these two prepositions to the verb ontological relations *handle* and *convertTo*, respectively.

Figure 6.8 shows the parse trees generated by UIA and TIA for the method name *removeFile* of class *DirectoryCache* found in *FileZilla*. From these parse trees we can extract the ontological relation and concepts: *remove(directory cache, file)*.

```
<root>
  <C id="E0" pos="0">
    <V id="2" pos="2" role="i" phrase="remove">
      <N id="1" pos="1" role="s" phrase="Subjects"
          base="subject"/>
      <N id="E2" pos="3" role="subj" base="subject"
          antecedent="1"/>
      <N id="3" pos="4" role="obj" phrase="file"/>
    </V>
  </C>
</root>
```

```
<root>
  <VB id="1" pos="0" role="null" phrase="remove">
    <NN id="2" pos="1" role="dobj" phrase="file">
    </NN>
  </VB>
</root>
```

Figure 6.8: Parse trees generated by UIA (top) and TIA (bottom) for the sentence *Subjects* ***remove file*** and ***remove file***, respectively. ***removeFile*** is a FileZilla method identifier.

***hasProperty* ontological relation:** a relation holding between a concept and its properties. This relation is extracted in a similar way as the $< verb >$ relation but only for the verbs *get* and *set*. The concepts connected by this relation are the class name and the object of the verb, reported as an *obj* or *dobj* dependency by UIA or the other analyzers, respectively.

***hasState* ontological relation:** a relation holding between a concept and its state properties. This relation is mapped to a *be* dependency in the parse tree generated by UIA. The corresponding dependency relations of UPA, TPA, and TIA, which are mapped to this ontological relation, are *auxpass* (*passive auxiliary*), *cop* (*copula*), *acomp* (*adjectival complement*), and *neg* (*negation modifier*). The concepts connected by this relation are the class name and the predicate verb with the corresponding object, when available. For example, from the parse trees generated for the method name `isClosed` of JEdit class `View`, we can extract the ontological relation *hasState(view, closed)* (see Figure 6.9).

```
<root>
  <C id="E0" pos="0">
    <V id="3" pos="3" role="i" phrase="closed"
      base="close">
      <N id="1" pos="1" role="s" phrase="Subject"/>
      <be id="2" pos="2" role="be" phrase="is"
        base="be"/>
      <N id="E2" pos="4" role="obj" base="subject"
        antecedent="1"/>
    </V>
  </C>
</root>
```

```
<root>
    <VBN id="2" pos="0" role="null" phrase="closed">
      <VBZ id="1" pos="2" role="auxpass" phrase="is">
      </VBZ>
    </VBN>
</root>
```

Figure 6.9: Parse trees generated by UIA (top) and TIA (bottom) for the sentence *Subject **is closed*** and ***is closed***, respectively. ***isClosed*** is a JEdit method identifier.

6.4 Concept location

Concept location is an activity where a programmer searches the source code to identify a specific part that implements a given concept ([95]). It involves formulating a query composed of one or more keywords which a programmer thinks are related or refer to the concept to be searched.

After querying the code base with the initially formulated query, the programmer will analyze the returned results. If she is not satisfied with the result she may decide to reformulate the query or to further filter the results (see [89], and [48]). Successive filtering of the query continues until the programmer is satisfied with the result.

To carry out a concept location task, a developer can employ information retrieval (IR) based approaches or regular expression matching. IR-based approaches treat the source code as a document corpus and use methods such as latent semantic indexing (LSI) to index the corpus (see [72], [93], [92], and [33]). In regular expression matching, however, the query formulated by the developers is directly matched against the content of the files in the code base using tools such as *grep*[6]. IR-based approaches return a ranked list of documents indicating the relevance of a document to the query, while grep returns a set of files which contain all the query keywords. In our study, we have used both IR-based and regular expression matching based approaches to locate concepts.

One of the applications in which concept location is widely used is bug fixing. When users of a program encounter a problem, they communicate it to the developers of the

[6] http://www.gnu.org/software/grep/

program by filing a bug report. The bug report contains several data, among which a title, a bug description and (optionally) a set of keywords. It is reasonable to assume that developers will use this information to formulate a query, used to retrieve files which are relevant for the bug to be fixed. In our study, we call such queries *basic queries*.

In the work ([1]), they have proposed to enhance basic queries using concepts taken from the ontology extracted from the corresponding source code. The enhancement of the queries is carried out by expanding the set of keywords used for formulating the basic queries with concept names taken from the ontology. The concept names are selected by first matching each keyword to the concepts in the ontology and taking the neighboring concepts of the matched concept. A match is found when the name of a concept is the same as the keyword. A neighboring concept of a given concept is any concept that is exactly one edge away from the matched concept, where by edge we mean any ontological relation (*isA*, *hasProperty*, etc.). For example, if we take the portion of ontology shown in Figure 6.10, for the concept represented by the keyword *data*, all concepts except *http control socket* and *external ip resolver* will be considered as additional keywords to be used in formulating the query. In the following, we refer to queries formulated in this way as *enhanced queries*.

In our approach, the *enhanced queries* can be formulated from the ontologies built using the parse trees of either UIA, UPA, TPA, or TIA. We call the enhanced queries formulated using concepts taken from the ontology built using parse trees of UIA/UPA/TPA/TIA respectively as *UIA/UPA/TPA/TIA enhanced queries*.

The relation between concepts in the ontology are derived from the natural language dependencies. Since these often represent a semantic relation between the terms they connect, we argue that the concepts connected in the ontology are also closely related. Consequently we conjecture that the expansion of the query with the additional closely related concepts could potentially improve the quality of the query.

6.5 Case studies

To investigate the support programmers can get through ontology extraction, we have conducted a case study. In this case study, we address the following research questions.

- **RQ1. Ontology comparison:** Do the ontologies produced by different analyzers differ between each other?

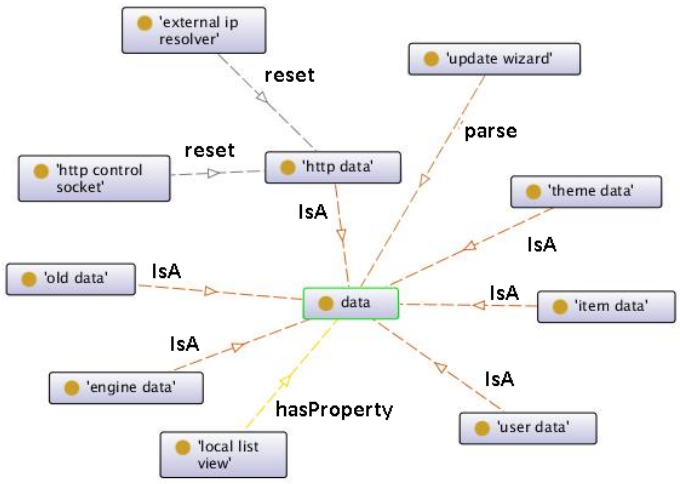

Figure 6.10: An example portion of ontology built using the steps described in Section 6.3. The concepts are presented in the rectangle boxes and the relations connecting the concepts are shown next to the edges connecting the concepts.

- **RQ2. Analyzer impact:** Does the choice of the analyzer impact the effectiveness of concept location?

The concepts used in the enhanced queries are retrieved from different ontologies which are generated using different analyzers. In RQ1, we compare four ontologies, generated using the analyzers UIA, UPA, TPA and TIA. To compare the ontologies, we compute the Jaccard index ($|A \cap B|/|A \cup B|$) between them, to see how similar they are, and the ratio of unique concepts each ontology has to their union ($|A \setminus B|/|A \cup B|$, $|B \setminus A|/|A \cup B|$).

The enhanced queries use concepts taken from different ontologies which are based on different parse trees of identifiers. The different parse trees are produced using UIA, UPA, TPA and TIA. In our last research question, RQ2, we investigate whether the choice of the analyzer impacts the effectiveness of the enhanced queries in concept location and we test if the impact is statistically significant. The investigation is conducted by comparing the effectiveness of the enhanced queries formulated using concepts taken from the ontologies produced using the outputs of the analyzers. To carry out the test, we have formulated the following null/alternative hypotheses:

> $\mathbf{H_{0-RQ2}}$: There is no statistically significant difference between the effectiveness of UIA, UPA, TPA, and TIA enhanced queries formulated by expert or average programmers while using either grep-based or LSI-based approach.
>
> $\mathbf{H_{1-RQ2}}$: There is a statistically significant difference between the effectiveness of UIA, UPA, TPA, and TIA enhanced queries formulated by expert or average programmers while using either grep-based or LSI-based approach.

To test the hypotheses, we have conducted a two-sided, pair-wise Wilcoxon signed-rank test. The pair-wise tests computed to test these hypotheses are summarized in Table 6.1.

Table 6.1: Summary of pairs used to answer RQ1 and RQ2 hypotheses. The analyzer names used in the table correspond to the enhanced queries formulated using the ontology built from the respective analyzer.

Search approach	**LSI-based**		**Grep-based**	
Query	**Best**	**Average**	**Best**	**Average**
H_{0-RQ2}	UIA vs. UPA	UIA vs. UPA	UIA vs. UPA	UIA vs. UPA
	UIA vs. TPA	UIA vs. TPA	UIA vs. TPA	UIA vs. TPA
	UIA vs. TIA	UIA vs. TIA	UIA vs. TIA	UIA vs. TIA
	UPA vs. TPA	UPA vs. TPA	UPA vs. TPA	UPA vs. TPA
	UPA vs. TIA	UPA vs. TIA	UPA vs. TIA	UPA vs. TIA
	TPA vs. TIA	TPA vs. TIA	TPA vs. TIA	TPA vs. TIA

In our study, we have conducted multiple tests on the hypotheses formulated for one research questions (see Table 6.1). To control the false discovery rate and correct for multiple comparison, we have adjusted the p-values using the [8] (BH) correction.

6.5.1 Data sets: Subject programs

In our case study, we considered three medium size open source systems, FileZilla client[7], JEdit[8] and WinMerge[9]. FileZilla client is a GUI based FTP client which is mainly used to upload and download files from an FTP server. WinMerge is a merging and differencing utility for Windows, while JEdit is a cross platform text editor mainly developed for programmers. FileZilla and WinMerge are written in C++ while JEdit is written in Java. All systems have a bug tracking system from which we collected closed bug reports with patch files. From the patches we have collected the names of the classes and files which are actually modified to fix the bugs. These classes and files are used as our reference to compute reciprocal rank, precision, and recall (i.e., these are the correct program entities to be retrieved by means of both basic and enhanced queries).

6.5.2 Procedure

Our case study has three main steps, *identifier parsing*, *ontology extraction* and *concept location*. Below, we describe each step in detail.

[7] http://filezilla-project.org/
[8] http://jedit.org/
[9] http://winmerge.org/

Identifier parsing

Before parsing identifier names using the analyzers, they have to be tokenized. If the identifier names are composed of more than one term, we regard camel casing and underscore as separators and use them to split identifier names into their composing tokens. Before splitting an identifier into its composing tokens, prefixes, such as those associated with the Hungarian notation (e.g., *m_* for data members and *C* for class names) are removed.

Sometimes an identifier's token is not a word. In such cases we consult a predefined list of "known abbreviations and contractions" to identify a possible expansion for the token. If the token is not in the predefined list, we use the longest common sub-string (LCS) technique to find the most similar expanded form for the token. According to this technique, an available dictionary of words is accessed to find the most similar word (i.e., the one with smallest LCS with the given token). For example, the token "remot" is replaced by "remote" after applying the LCS algorithm. The expansion of a token to its respective word can also be carried out using the techniques described in existing works on the topic (see [61, 64, 47, 62]; and [25]). In the following, we assume that the tokenization step has produced a sequence of valid words.

To automate the tokenization step, we have developed a tool which automatically collects and produces a tokenized list of class, attribute and method identifier names, following the procedure described above. In addition to this, the tool produces a *fact file*, which is used for further processing in the next step (see Section 6.5.2). The fact file contains information about all classes in the system and their members. The inputs to our tool are XML representations of the source code files, produced by the *src2srcml* tool ([23]) and a configuration file which contains options and file path information for the files containing the list of "known abbreviations and contractions" and identifier naming conventions (e.g., Hungarian notation in use).

The tokenized list of class, attribute and method identifier names are passed as an input to three types of syntactic analyzers, namely UPA, TPA and TIA (SVMTool/MaltParser). The trained analyzers are obtained following the approach presented in Section 6.2.3. Furthermore the sentences constructed using the same tokenized list is the input of UIA (Minipar) and they are constructed using the steps described in Section 6.2.2. The output of the four analyzers is a set of dependency parse trees which are used as an input of the following step.

Ontology extraction

In this step, we build four types of ontologies using the information captured in the parse trees generated by UIA, UPA, TPA and TIA. To build the ontologies from the parse trees, we have used the natural language dependency to ontological relation mappings described in Section 6.3. For some of the mappings, such as for the *hasProperty* ontological relation, we have to map the identifier containing the ontological relation to the source code where the identifier is defined (e.g., to determine the containing class). To automate this step and make it work with all analyzers, we have modified the tool developed in a previous study ([1]).

To generate ontologies, our tool takes the parse trees produced for all identifiers and the fact file generated in a previous step (see Section 6.5.2) as an input. The fact file is used to identify the containing classes of a given identifier which is reconstructed from its parse tree. The containing classes are required to create some ontological relations, as described above. For each set of parse trees generated by UIA, UPA, TPA and TIA, our tool produces four ontologies, which are named after the corresponding analyzers: *UIA ontology*, *UPA ontology*, *TPA ontology* and *TIA ontology*.

Concept location

To carry out the concept location task, one of the authors has played the role of the programmer and has manually collected keywords from each bug title. Bug titles usually serve as the summary for the problem described in the corresponding bug report and hence are good sources of keywords. To avoid any bias, the collection of keywords was conducted prior to computing any results. These keywords are used in the selecting the concepts to be added when formulating the enhanced queries (see Section 6.4), respectively called *UIA enhanced query*, *UPA enhanced query*, *TPA enhanced query* and *TIA enhanced query*. For example, *UIA enhanced query* is an enhanced query formulated using concepts taken from *UIA ontology* which is built using parse trees generated by UIA.

To query the source code, we have used two different approaches: information retrieval (IR) and regular expression matching. The IR-based approach uses latent semantic indexing (LSI) to index the document corpus (see [72]). To rank the documents in the indexed corpus, similarity between the query and every document in the indexed corpus is computed. If the result of the similarity measure is high, the document is ranked closer to the top. To compute similarity between the query and the documents, we have used

cosine similarity. Cosine similarity is the most widely used measure while dealing with vector-based representation of documents.

For the second approach, we have used the widely used, yet simple method *grep*. *Grep* performs a pattern matching of the query against the content of the files in the code base and returns all the files which contain all keywords in the query. Hereafter we refer to this approach as *grep*-based while we refer to the former, IR-based, approach as *LSI*-based approach.

6.5.3 Results

RQ1. Ontology comparison

To answer the RQ1 research question, we have computed the Jaccard index between each pair of ontologies and the ratio of unique concepts and relations each ontology has to their union (see Tables 6.2 and 6.3). The comparison of the paired ontologies is done by considering the union of all concepts, the union of all relations and the union of all paired concepts.

While the first comparison considers only concepts, the other two focus on pairs of concepts. In the comparisons which focus on pairs of concepts, the union of concept *relations* deals with named relations. On the contrary, the union of *paired concepts* is computed irrespective of the name of the relation connecting the two concepts. Hence, while two relations match if both concepts at the end of each relation and the relation names match, relation names are not taken into consideration while matching paired concepts. In short, concept relations are named while concept pairs are unnamed.

The Jaccard index computed for all union types of the paired ontologies show that there is some degree of similarity between the respective ontologies (see Tables 6.2 and 6.3). From the results, it is also apparent that each type of ontology is characterized by a peculiar set of concepts and relations. Hence, we can say that none of the ontologies subsume any of the other types of ontologies nor they are exactly the same.

The concepts and relations appearing in some but not all ontologies are due to the different parse trees generated by the different analyzers. For example, from the parse trees generated by UPA and TPA for the method name *findMatchingBracket* in class *TextUtilities* (see Figure 6.11), we get different concepts and relations. UPA identifies *matching* as *xcomp* (clausal complement) which is not mapped to any of the ontological relations we defined. It has considered *Bracket* as the direct object of <*verb*> *matching*, which is mapped to the

Table 6.2: **RQ1: Pair wise comparison** between UIA ontology and the remaining three types of ontologies (UPA, TPA and TIA).

		Ontology			
		UIA ⊔ UPA	**Only in**		**Common (Ratio)**
			UIA	**UPA**	
FileZilla	**Concepts**	1940	780(0.402)	524(0.27)	636(0.328)
	Relations	2676	1254(0.469)	804(0.300)	618(0.231)
	Paired cpts	2446			848(0.347)
JEdit	**Concepts**	2911	602(0.207)	562(0.193)	1747(0.600)
	Relations	4163	1081(0.260)	1217(0.292)	1865(0.448)
	Paired cpts	3875			2153(0.556)
WinMerge	**Concepts**	2648	592(0.224)	809(0.306)	1247(0.471)
	Relations	3712	996(0.268)	1305(0.352)	1411(0.380)
	Paired cpts	3508			1615(0.460)
		UIA ⊔ TPA	**Only in**		**Common (Ratio)**
			UIA	**TPA**	
FileZilla	**Concepts**	1957	637(0.325)	541(0.276)	779(0.398)
	Relations	2718	1042(0.383)	846(0.311)	830(0.305)
	Paired cpts	2470			1078(0.436)
JEdit	**Concepts**	2966	564(0.190)	617(0.208)	1785(0.602)
	Relations	4267	1081(0.253)	1321(0.310)	1865(0.437)
	Paired cpts	3980			2152(0.541)
WinMerge	**Concepts**	2591	594(0.229)	752(0.290)	1245(0.481)
	Relations	3583	1061(0.296)	1176(0.328)	1346(0.376)
	Paired cpts	3381			1548(0.458)
		UIA ⊔ TIA	**Only in**		**Common (Ratio)**
			UIA	**TIA**	
FileZilla	**Concepts**	1618	610(0.377)	202(0.125)	806(0.498)
	Relations	2672	889(0.333)	800(0.299)	983(0.368)
	Paired cpts	2351			1304(0.555)
JEdit	**Concepts**	2662	970(0.364)	313(0.118)	1379(0.518)
	Relations	4417	1303(0.295)	1471(0.333)	1643(0.372)
	Paired cpts	4080			1980(0.485)
WinMerge	**Concepts**	2253	844(0.375)	414(0.184)	995(0.442)
	Relations	3409	1078(0.316)	1002(0.294)	1329(0.390)
	Paired cpts	3108			1630(0.524)

Table 6.3: **RQ1: Pair wise comparison** between UPA, TPA and TIA ontologies

		Ontology			
		UPA ⊔ TPA	**Only in**		**Common (Ratio)**
			UPA	**TPA**	
FileZilla	**Concepts**	1582	262(0.166)	422(0.267)	898(0.568)
	Relations	2096	420(0.200)	674(0.322)	1002(0.478)
	Paired cpts	1952			1146(0.587)
JEdit	**Concepts**	2583	181(0.070)	274(0.106)	2128(0.824)
	Relations	3558	372(0.105)	476(0.134)	2710(0.762)
	Paired cpts	3192			3076(0.964)
WinMerge	**Concepts**	2143	146(0.068)	87(0.041)	1910(0.891)
	Relations	2851	329(0.115)	135(0.047)	2387(0.837)
	Paired cpts	2758			2480(0.899)
		UPA ⊔ TIA	**Only in**		**Common (Ratio)**
			UPA	**TIA**	
FileZilla	**Concepts**	1499	491(0.328)	339(0.226)	669(0.446)
	Relations	2365	582(0.246)	943(0.399)	840(0.355)
	Paired cpts	2126			1079(0.508)
JEdit	**Concepts**	2468	776(0.314)	159(0.064)	1533(0.621)
	Relations	4121	1007(0.244)	1039(0.252)	2075(0.504)
	Paired cpts	3473			2723(0.784)
WinMerge	**Concepts**	2279	870(0.382)	223(0.098)	1186(0.520)
	Relations	3608	1277(0.354)	892(0.247)	1439(0.399)
	Paired cpts	3395			1652(0.487)
		TPA ⊔ TIA	**Only in**		**Common (Ratio)**
			TPA	**TIA**	
FileZilla	**Concepts**	1557	549(0.353)	237(0.152)	771(0.495)
	Relations	2424	641(0.264)	748(0.309)	1035(0.427)
	Paired cpts	2169			1290(0.595)
JEdit	**Concepts**	2581	889(0.344)	179(0.069)	1513(0.586)
	Relations	4291	1177(0.274)	1105(0.258)	2009(0.468)
	Paired cpts	3680			2620(0.712)
WinMerge	**Concepts**	2218	809(0.365)	221(0.100)	1188(0.536)
	Relations	3437	1106(0.322)	915(0.266)	1416(0.412)
	Paired cpts	3213			1640(0.510)

ontological relation *matching(TextUtilities, Bracket)*. TPA, on the other hand, has identified the *NN* (noun-noun specifier) natural language dependency between *Matching* and *Bracket*, and considered *MatchingBracket* as the direct object of *find*. As compared to the ontological relations produced by UPA, this results in two different ontological relations: *isA(MatchingBracket, Bracket)* and *find(TextUtilities, MatchingBracket)*.

If we consider the concepts produced by the two analyzers UPA and TPA, two of them are common, *Bracket* and *TextUtilities*. Concept *MatchingBracket* is extracted only by TPA, which, differently from UPA, correctly identifies the specifier dependency relationship between *matching* and *bracket*. In this case, training is crucial in order for the analyzer to be able to recognize the specifier dependency which is instead missed by the general purpose, untrained analyzer UPA. UPA misses one, potentially relevant concept, as compared to TPA.

Relations between concepts identified by UPA and TPA are completely disjoint. While UPA identifies a *matching* relation between *TextUtilities* and *Bracket*, no such relation is reported by TPA, which, instead identifies two other relations, *isA* and *find*, connecting different pairs of concepts, which also means that the two analyzers do not identify any common paired concepts. When the extracted relations are used to determine the neighboring concepts that are used to enhance a query, the two analyzers may report different concepts, because of the difference in the extracted relations. In turn, this might affect the effectiveness of the enhanced query.

```
<root>
  <VB id="1" pos="0" role="null" phrase="find">
    <VBG id="2" pos="1" role="xcomp" phrase="matching">
      <NN id="3" pos="2" role="dobj" phrase="bracket">
      </NN>
    </VBG>
  </VB>
</root>
```

```
<root>
  <VB id="1" pos="0" role="null" phrase="find">
    <NN id="3" pos="1" role="dobj" phrase="bracket">
      <NN id="2" pos="3" role="nn" phrase="matching">
      <NN/>
    </NN>
  </VB>
</root>
```

Figure 6.11: Parse trees generated by UPA (top) and TPA (bottom) for the JEdit method name ***findMatchingBracket*** in class *TextUtilities*.

RQ2. Analyzer impact

The different types of ontologies used in this study are built using dependency parse trees generated by UIA, UPA, TPA and TIA. From the comparison of the ontologies (RQ1), we have seen that the ontologies are not exactly the same. RQ2 investigates if this difference has impacted the effectiveness of the enhanced queries used in concept location. To answer this research question, we have computed the net improvement achieved by each type of enhanced query over the other for both best and average, LSI and grep-based queries (see Tables 6.4 and 6.6 for the net improvement and Tables 6.5, 6.7 for a detailed comparison).

Values (except for the p-values) indicate the number of cases in which the enhanced query indicated in each column improves the enhanced query indicated in each row. A negative value indicates that it is the query in the row that improves the query in the column.

Table 6.4: ***RQ2: Enhanced vs. enhanced queries; best queries.*** Net improvement of paired enhanced queries and the corresponding p-values as computed using their top ranks and best F-measures with the corresponding precision and recall measures. The p-values computed over values of MRR and F-measure are adjusted for multiple tests.

	System		**FileZilla**			**JEdit**			**WinMerge**		
	Enhanced query		**UPA**	**TPA**	**TIA**	**UPA**	**TPA**	**TIA**	**UPA**	**TPA**	**TIA**
LSI-based	**Top Ranks**	**UIA**	0	0	1	1	1	1	-6	-5	-7
		UPA		2	2		1	0		1	-1
		TPA			2			-1			-2
	P-value	**UIA**	0.99	0.93	1.00	1.00	1.00	1.00	0.11	0.16	0.07
		UPA		0.99	0.99		1.00	NaN		1.00	0.58
		TPA			0.99			1.00			0.35
Grep-based	**Precision**	**UIA**	-4	1	3	0	1	1	0	0	0
		UPA		6	6		1	-1		0	-1
		TPA			0			-2			-1
	Recall	**UIA**	0	-1	0	0	0	1	-2	-2	-2
		UPA		-1	0		0	1		0	0
		TPA			1			1			0
	F-measure	**UIA**	-4	1	3	0	1	1	0	0	0
		UPA		6	6		1	-1		0	-1
		TPA			0			-2			-1
	P-value (F)	**UIA**	0.99	0.99	0.93	1.00	1.00	1.00	0.57	0.57	0.57
		UPA		0.99	0.99		1.00	1.00		NaN	0.94
		TPA			0.99			1.00			0.94

For the LSI-based approach, the net improvement of the top rank of one type of enhanced query over the other is marginal for all systems except WinMerge (see Table 6.4). The highest net improvements for WinMerge are observed when UIA is compared with UPA, TPA, and TIA, with net improvements of 6, 5, and 7, respectively. The pair-wise comparison result of the median ranks is also marginal for most cases while using LSI-based approach (see Table 6.6). The highest net improvement in this case is 4; and it is observed for WinMerge when comparing TIA with TPA. The details of the number of times one type of enhanced query is better, less than or equal to the other in terms of top and median ranks are shown in Tables 6.5 and 6.7.

Table 6.5: **RQ2: Detailed comparison of enhanced vs. enhanced queries; best queries.** Enhanced queries are compared on top ranks and best F-measure.

System	Enhanced query	Top ranks			Best F-measures		
		UIA			UIA		
		Better	Less	Equal	Better	Less	Equal
FileZilla	**UPA**	4	4	20	4	8	16
	TPA	3	3	22	5	4	19
	TIA	2	1	25	4	1	23
JEdit	**UPA**	1	0	10	2	2	8
	TPA	1	0	10	3	2	7
	TIA	1	0	10	2	1	9
WinMerge	**UPA**	0	6	13	3	3	14
	TPA	0	5	14	3	3	14
	TIA	0	7	12	3	3	14

System	Enhanced query	UPA			UPA		
		Better	Less	Equal	Better	Less	Equal
FileZilla	**TPA**	4	2	22	9	3	16
	TIA	5	3	20	10	4	14
JEdit	**TPA**	1	0	10	1	0	11
	TIA	0	0	11	1	2	9
WinMerge	**TPA**	1	0	18	0	0	20
	TIA	1	2	16	1	2	17

System	Enhanced query	TPA			TPA		
		Better	Less	Equal	Better	Less	Equal
FileZilla	**TIA**	4	2	22	4	4	20
JEdit	**TIA**	0	1	10	1	3	8
WinMerge	**TIA**	1	3	15	1	2	17

Table 6.6: ***RQ2: Enhanced vs. enhanced queries; average queries.*** Net improvement of paired enhanced queries and the corresponding p-values as computed using their median ranks and median F-measures with the corresponding median precision and recall measures.The p-values computed over values of MRR and F-measure are adjusted for multiple tests.

	System		**FileZilla**			**JEdit**			**WinMerge**		
	Enhanced query		**UPA**	**TPA**	**TIA**	**UPA**	**TPA**	**TIA**	**UPA**	**TPA**	**TIA**
LSI-based	**Median rank**	**UIA**	3	1	-1	1	1	1	1	-1	0
		UPA		-1	2		2	1		-2	2
		TPA			2			1			4
	P-value	**UIA**	0.99	0.99	0.99	1.00	1.00	1.00	0.94	1.00	0.96
		UPA		0.99	0.99		0.50	1.00		0.57	0.94
		TPA			0.99			1.00			0.75
Grep-based	**Precision**	**UIA**	2	4	5	0	-1	-1	7	6	2
		UPA		4	3		-1	-3		-2	-4
		TPA			2			-2			-2
	Recall	**UIA**	0	-2	0	0	0	0	0	0	-1
		UPA		-2	0		0	0		0	-1
		TPA			2			0			-1
	F-measure	**UIA**	4	6	4	0	-1	-1	6	5	1
		UPA		5	0		-1	-3		-1	-3
		TPA			-1			-2			-2
	P-value (F)	**UIA**	0.78	0.25	0.26	1.00	1.00	1.00	0.20	0.28	0.99
		UPA		0.99	0.99		1.00	1.00		1.00	0.29
		TPA			0.99			1.00			0.57

Table 6.7: **RQ2: Detailed comparison of enhanced vs. enhanced queries; average queries.** Comparison of enhanced queries on median ranks and median F-measure.

System	**Enhanced query**	**Median Rank**			**Median F-measure**		
		UIA			**UIA**		
		Better	**Less**	**Equal**	**Better**	**Less**	**Equal**
FileZilla	**UPA**	11	8	9	10	6	9
	TPA	10	9	9	8	2	15
	TIA	7	8	13	5	1	19
JEdit	**UPA**	5	4	2	2	2	8
	TPA	5	4	2	2	3	7
	TIA	5	4	2	1	2	9
WinMerge	**UPA**	8	7	4	9	3	8
	TPA	7	8	4	8	3	9
	TIA	6	6	7	6	5	9
System	**Enhanced query**	**UPA**			**UPA**		
		Better	**Less**	**Equal**	**Better**	**Less**	**Equal**
FileZilla	**TPA**	8	9	11	8	3	14
	TIA	11	9	8	7	7	11
JEdit	**TPA**	3	1	7	1	2	9
	TIA	4	3	4	1	4	7
WinMerge	**TPA**	0	2	17	0	1	19
	TIA	7	5	7	2	5	13
System	**Enhanced query**	**TPA**			**TPA**		
		Better	**Less**	**Equal**	**Better**	**Less**	**Equal**
FileZilla	**TIA**	10	8	10	4	5	16
JEdit	**TIA**	4	3	4	2	4	6
WinMerge	**TIA**	8	4	7	3	5	12

The net improvements of the best F-measures of one type of enhanced query over the others while using grep-based approach are also marginal in all pairs except for FileZilla (see Table 6.4). In FileZilla, TPA and TIA enhanced queries are found more effective than both UIA and UPA enhanced queries. The highest net improvement, 6, is observed when comparing the highest F-measures of TPA and TIA with UPA. Like the best F-measures, the net improvements of the median F-measures are marginal for all pairs except for some cases of FileZilla and WinMerge (see Table 6.6). The highest net improvement, 6, is observed for FileZilla when comparing TPA with UIA, and for WinMerge when comparing UPA with UIA. The details of the number of times one type of enhanced query is better, less than or equal to the other in terms of effectiveness (F-measure) are shown in Tables 6.5 and 6.7. The results show that in the majority of the cases all pairs have performed almost equally.

To further analyze if the differences observed are statistically significant, we have formulated the hypothesis stated in H_{0-RQ2} and we have conducted a two-sided, pair-wise Wilcoxon signed-rank test (see Table 6.1, H_{0-RQ2}, LSI-based and grep-based approaches). The results are shown in Tables 6.4 and 6.6. The p-values in the tables indicate that the observed differences are not statistically significant at $\alpha = 0.05$ in all the cases.

From the results, we can conclude that the difference in the analyzers used to build the ontologies has little or no impact on the effectiveness of the respective enhanced queries in concept location.

6.6 Conclusion and Future Works

We have presented the use of four types of natural language analyzers to parse identifiers of a system and extract ontologies. Two of the analyzers are adapted to directly work on identifiers through training while the other two are standard English analyzers. The training of the analyzers is conducted automatically using a training set constructed from the documentation of the corresponding system. To evaluate the benefits of using ontologies constructed from parse trees of identifiers, we have carried out a case study on three open source systems. The case study was conducted in the context of the support they can give to concept location while using LSI and grep-based approaches.

The results of the case study, show that using concepts taken from the ontologies extracted from the respective systems have improved the effectiveness of concept location queries which can be formulated by experts, while using both LSI and grep-based approaches. This is achieved irrespective of the type of natural language analyzer used in this study.

The statistical test conducted on the results also confirms the observation in the majority of the cases (i.e., the results of at least 8 out of 12 cases for both LSI-based approach and for grep-based approach are found statistically significant at $\alpha = 0.05$). For average queries, improvement in effectiveness of queries is observed only when using the grep-based approach. The improvements observed in this case are statistically significant at $\alpha = 0.05$ in half of the cases.

The comparison of the ontologies generated using different analyzers shows that they are different, with some concepts and relations in common. However, this did not impact the support they give to concept location. The comparison on the support they give to concept location show that in the majority of the cases, they perform equally well and the observed small differences are not statistically significant.

Chapter 7

Italian Parsing performance: Dependecy vs Costituency

7.1 Introduction

The aim of this Chapter is to contribute to the debate on the issues raised by Morphologically Rich Languages, and more precisely to investigate, in a cross-paradigm perspective, the influence of the constituent order on the data-driven parsing of one of such languages (i.e. Italian). The experiments are performed by using state-of-the-art data-driven parsers (i.e. MaltParser and Berkeley parser) and are based on an Italian treebank, i.e. the Turin University Treebank (TUT), available in formats that vary according to two dimensions, i.e. the paradigm of representation (dependency vs. constituency) and the level of detail of linguistic information.

In the following Sections we summarize the main recent experiences in Italian parsing both for dependency and constituency giving also a short description of dependency and constituency syntactic analyses. Afterwards we describe the data used in our experiments and in particular to the description of the annotation formats of TUT and we present the experiments and a discussion of the results. Finally, we draw some conclusion and plans for future work.

7.2 Background and Motivations

In the last years, results for Italian parsing have been reported for both dependency and constituency paradigms mainly in the context of evaluation campaigns.

As far as dependency parsing is concerned, Italian was one of the languages on which parsers were tested during the multilingual track of the CoNLL Shared Task in 2007 [83]. The data set was taken from the Italian Syntactic-Semantic Treebank (ISST) [76], which has been semi-automatically converted to the CoNLL format using information from its two annotated levels, i.e. the constituency and the functional structure. Notwithstanding the relatively small size of the data set (71K tokens), the accuracy for Italian was among the highest (Labeled Accuracy Score 84.40) together with those for Catalan, Chinese and English (Labeled Accuracy Scores between 84.40 and 89.61). More recently, dependency parsers have been tested within the Evalita evaluation campaigns for Italian NLP tools, in 2007, 2009 and 2011 [15, 16, 14]. The data sets were taken from the available releases of TUT, whose size progressively increased from 2,400 to more than 3,450 sentences (102,150 tokens in the current version). A version of this treebank in CoNLL format was created for the Evalita evaluation campaigns. The results reported in the Evalita contests improved constantly over the years. In 2007, the best reported Labeled Accuracy Score (LAS) was 86.94 by TULE [65], a rule-based system developed in parallel with TUT. In 2009, the best LAS was around 88.70 achieved by both TULE [66] and DeSR [4], a statistical parser[1]. Finally in 2011, the best performance has been scored with LAS 91.23 and was achieved by the system described in [43].

As far as constituency parsing is concerned, the only recently published results are those reported with reference to the Evalita campaigns, which have been constantly improved but still far from those for English. In fact, the best performance attested at the 2011 edition of Evalita was F_1 82.96, Bracketing Recall 82.97 and Bracketing Precision 82.94 [13].

The relevance of the comparison between different frameworks is also shown by a recent work [106] on the problem of a fair comparison of performance in different frameworks.

As far as word order (and, more specifically, constituent order) is concerned, it is usually included among the features that can motivate the degradation of results in parsing MRLs, and it has been mainly investigated in the perspective of tasks such as Machine Translation and Language Generation [5]. It is acknowledged that a combination of several factors determines the order of words, e.g. semantic roles, topic, focus, theme/rheme and communicative events. In free word order languages, the order is used to structure the information being conveyed to the hearer, while in fixed word order languages the same role is played by intonation and stress [50]. Nevertheless, a difficulty is related to

[1] This contest included also a pilot task with training and testing data from ISST (best LAS 83.38 by [4]).

the formal description and processing of free word order languages: instead of a complete lack of ordering rules, many subtle language specific restrictions apply to the order variation [41]. Therefore, a large amount of variations can be considered as grammatical in isolated sentences, but, depending on the context, different word orders are either required or more natural than others [104].

On the one hand, approaches to language description based on constituency characterize syntactic structure in predominantly static terms paying minimal attention to the communicative function that mainly motivates the word order variation [111]. They are usually considered not adequate for representing orders like VSO (Verb-Subject-Object) or OVS (Object-Verb-Subject), where the Subject is after the Verb, or where any number of adjuncts can be positioned between complements. For instance, in order to represent such a kind of structures, the Penn format should be increased by new representational tools, like in TUT–Penn (see Section 7.5.1). On the other hand, dependency approaches do not explicitly constrain the word order, at least until structures are continuous and projective, giving a less specified representation of word order. At least in part, this can explain the different performance in constituency versus dependency parsing for MRLs.

7.3 Constituency and Dependency Parsing

Constituency syntactic analysis (parsing) is based on the decomposition of the sentence into smaller segments called *constituents* or *phrases*. According to their internal structure these segments are usually categorized in *noun phrase*, *verbal phrase* or *prepositional phrase*, etc. Figure 7.4 shows a constituency representation of a Italian sentence extracted by TUT, the data set used in our assessment. Furthermore in *dependency syntactic analysis* a sentence is analyzed by linking its words by binary asymmetrical relations, namely *dependency*, which are usually classified according to their functional role into *subject*, *object* etc. In the Figure 7.1 a dependency structure for Italian sentence taken also from TUT. The consequences of such differences in the frameworks are even more evident in data-driven approaches, especially with sparse data. In fact, constituency approaches impose more constraints on word order and therefore consider a larger number of different patterns when a lot of variations are possible. This results in a lower number of occurrences for each pattern, and therefore in a larger impact of data sparseness. All in all, the constituency framework will be more seriously hampered by the changes in word order with respect to the dependency one. In effect, initially methods for constituency parsing were mainly developed through experimentation on English data and especially the Penn Treebank [41]. On the contrary, dependency approaches do not explicitly constrain the

word order, at least until structures are continuous and projective, giving a less specified representation of word order.

7.4 Data Sets

The experiments presented in this Chapter are based on TUT, i.e. the freely available Italian resource developed by the Natural Language Processing group of the University of Turin [12][2]. The data currently consist in 102,150 annotated tokens (among which 84,666 words, 10,056 punctuation marks and 7,428 null elements) in TUT native format, which correspond to around 3,500 sentences[3] extracted from texts varying from newspapers, to legal, to Wikipedia. In the rest of this section, we will describe in detail the formats available for TUT focusing in particular on the distinctive dimensions of variation which characterize the annotation of this treebank, namely the paradigm of representation (dependency vs. constituency) and the level of specification of linguistic information.

7.4.1 The dependency formats

The core of the TUT project is a treebank in an original dependency format, henceforth indicated as *native TUT*, which has been afterwards enriched by the converted versions in constituency (see 7.4.2). Native TUT includes a specific format for representing Italian morphology and syntax.

Moreover, the treebank exploits a rich set of grammatical relations designed to represent linguistic information according to three different perspectives, namely morpho–syntax, functional syntax and semantics. Since the information related to each perspective is annotated in specially designed part of the relation label of TUT, called *component* (i.e. *morpho–syntactic*, *functional–syntactic* or *syntactic–semantic component*), the amount of linguistic knowledge annotated in the treebank can be easily varied by assuming more or less detailed relations, i.e. including from one to three of the above mentioned perspectives (below referred as *1–Comp*, *2–Comp* and *3–Comp*). This means that each relation label can in principle include all the three components, but can be made more or less specialized, including information from only one (i.e. the functional–syntactic) or two of them. For instance, the relation used for the annotation of locative prepositional modifiers, i.e. PREP–RMOD–LOC (which includes all the three components, in Figure

[2] `http://www.di.unito.it/~tutreeb`
[3] Average sentence length 23.90 words per sentence.

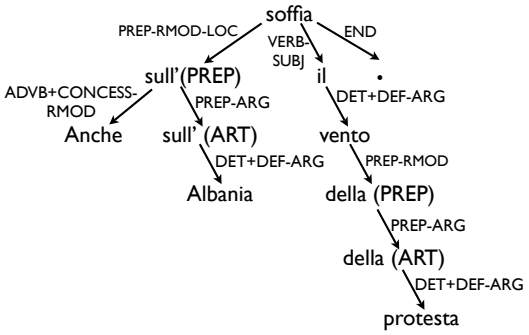

Figure 7.1: Sentence NEWS–549 in 3–Comp setting: "Anche sull'Albania soffia il vento della protesta." (Also on the Albania blows the wind of the revolt.).

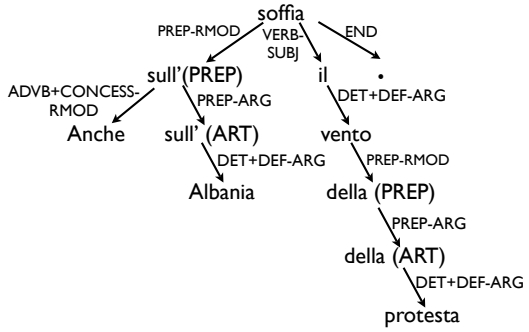

Figure 7.2: Sentence NEWS–549 in 2–Comp setting.

7.1), can be reduced to PREP–RMOD (which includes only the morpho–syntactic and the functional–syntactic component, in Figure 7.2) or to RMOD (which includes only the functional-syntactic component, in Figure 7.3). This works as a means for the annotators

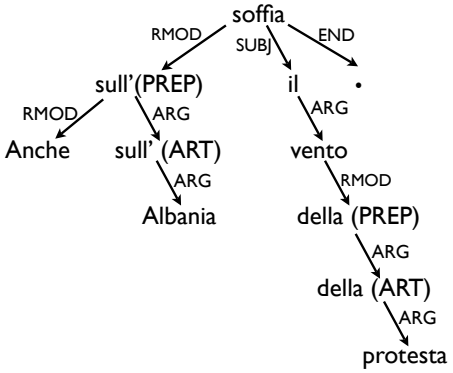

Figure 7.3: Sentence NEWS–549 in 1–Comp setting.

to represent different layers of confidence in the annotation, but can also be applied to increase the comparability of TUT with other existing resources, by exploiting the amount of linguistic information more adequate for the comparison, e.g. in terms of number of relations. For instance, in the Evalita campaigns the 1–Comp setting of the treebank has

been exploited. Since in more coarse-grained settings several relations can be merged into a single one (e.g. PREP–RMOD–TIME, used for temporal modifier, and PREP–RMOD–LOC are merged in RMOD), each setting includes a different number of relations: the setting based on the single functional–syntactic component (1–Comp) includes 72 relations, the one based on morpho–syntactic and functional–syntactic components (2–Comp) 140, and the one based on all the three components (3–Comp) 323.

7.4.2 Constituency formats

By applying conversion scripts to the treebank in native TUT format, the constituency version of TUT has been generated, which includes in particular the TUT–Penn and the Augmented–TUT–Penn (henceforth APE) formats.

TUT–Penn is an application of the English Penn Treebank (PTB) format to Italian, as happened for other languages, like Chinese[4] or Arabic[5], addressing the phenomena typical of these languages by new specific representational means.

For what concerns morphology, the size of the PoS tag set of the TUT–Penn, if compared with that exploited in English PTB, clearly reflects the differences between MRLs and morphologically poorer languages. Nevertheless, even if the representation of morphology is more fine-grained with respect to the one adopted for English in PTB, it is reduced with respect to the PoS tag set used in native TUT in order to avoid serious sparse data problems [24]. As said above, native TUT exploits a tag set including 16 grammatical categories, specialized by 43 types and a large variety of features. By contrast, TUT–Penn adopts a tag set of 68 tags only (versus 36 in the PTB). Beyond the information that the PTB tag set makes explicit[6], TUT–Penn takes into account a richer variety of features for Verbs, Adjectives and Pronouns. For the amalgamated words, as in native TUT, it is assumed an explicit representation of each of their parts as separated morpho-syntactic items, see e.g. the Articled Prepositions “sulla” (on the[fem sing]) and “della” (of the[fem sing]) in figure 7.4.

As far as syntax is concerned, the annotation in TUT–Penn is structurally the same as in PTB, but some difference can be observed with respect to the inventory of functional relations and the use of null elements. In fact, the (very limited set of) functional tags assumed in PTB is used also in TUT–Penn, but it is increased by some relations used

[4] See `http://www.cis.upenn.edu/~chinese/`.

[5] See `http://www.ircs.upenn.edu/arabic/`.

[6] Apart from a few cases of English morphological features which do not exist (e.g. possessive ending) or do not correspond with Italian forms (e.g. comparative Adjective and Adverb).

for representing phenomena related to the flexible Italian word order. For instance, the label EXTPSBJ is used for the annotation of subjects in post-verbal position. Also the standard PTB inventory of null elements is adopted in TUT–Penn, but while for English null elements are mainly traces denoting constituent movements, in TUT–Penn they can play different roles: zero Pronouns, reduction of relative clauses, elliptical Verbs and also the duplication of Subjects which are positioned after Verbs (which occurs around 900 times in the corpus).

```
( (S
    (PP-LOC (ADVB Anche)
            (PREP sull')
            (NP (ART~DE sull') (NOU~PR Albania)))
    (NP-SBJ (-NONE- *-1))
    (VP (VMA~RE soffia)
        (NP-EXTPSBJ-1
            (NP (ART~DE il) (NOU~CS vento))
            (PP (PREP della)
                (NP (ART~DE della) (NOU~CS protesta))))))
    (. .)) )
```

Figure 7.4: Sentence NEWS–549 in TUT–Penn.

To expand the possibility of cross-framework and cross-paradigm comparison and assuming the importance of the representation of the predicate argument structure in constituency-based representations too, we developed also the APE, a format which extends TUT–Penn by inheriting, when possible, the functional-syntactic knowledge encoded in the native dependency TUT. Figure 7.5 shows an example where the tags of this format allow to

```
( (S
    (PP-RMOD-LOC (ADVB Anche)
                 (PREP sull')
                 (NP-ARG (ART~DE sull')
                         (NOU~PR Albania)))
    (NP-SBJ (-NONE- *-1))
    (VP (VMA~RE soffia)
        (NP-EXTPSBJ-1
            (NP (ART~DE il)
                (NOU~CS vento))
            (PP-RMOD (PREP della)
                     (NP-ARG (ART~DE della)
                             (NOU~CS protesta))))))
    (PUNCT-END .)) )
```

Figure 7.5: Sentence NEWS–549 in Augmented–TUT–Penn.

draw distinctions among modifier and argument functions (e.g. PP–RMOD–LOC instead of PP–LOC in TUT–Penn, NP–ARG instead of NP to represent the arguments of Prepositions), and to annotate the function of the final punctuation mark (i.e. END). As in the native TUT, it is therefore possible to graduate the amount of linguistic knowledge annotated also in the constituency formats of TUT.

We conclude this section with some observation on the description of Italian that can be extracted from an analysis of TUT. This description mainly confirms that Italian has to be considered among MRLs (see e.g. [107]) since it shows quantitatively the features known in literature for this kind of languages: rich inflection, amalgamated words, pro–drop and a relatively free order of words.

order	**frequency**
SVO	79.09
SOV	7.20
OSV	6.61
OVS	5.13
VSO	1.08
VOS	0.89

(a)

order	**frequency**
SV	79.10
VS	20.90
OV	18.93
VO	81.07

(b)

Table 7.1: The frequency of permutations of (a) Subject, Verb and Object (SVO) in Italian declarative clauses and of (b) the Subject and Object preceding and following the Verb in Italian declarative clauses.

In particular, for word order, the analysis in Table 7.1(a) according to the Greenberg's six-ways typology [42], shows that all the six possible permutations of the main constituents can be found in declarative clauses[7] in TUT corpus, with the order SVO strongly prevailing on the others. But, since this analysis takes into account only transitive Verbs, with realized Object, and can be influenced by pro–drop, our observation has to be widened to the cases where the Verb precedes or follows Subject and Object [29] (see Table 7.1(b)).

7.5 Experimental Assessment

The aim of the experimental assessment is to compare the robustness of dependency and constituency models with respect to a free constituent order language as Italian and with respect to the amount of annotated linguistic information. The experiments are performed on the different TUT formats (i.e. $1|2|3$–Comp for dependency and APE and Penn for constituency) discussed in Section 7.4 by using two parsers, namely the Berkeley parser [90] for the constituency model, and MaltParser [82, 84] for the dependency one. Indeed, these two parsers have shown state-of-the-art performance during EVALITA 2009 and 2011.

The Berkeley parser is a constituency parser based on a hierarchical coarse-to-fine parsing, where a sequence of grammars is considered, each being the refinement, namely a

[7] The term declarative clause refers to clauses where Verb is in tensed form and not playing the role of relative.

partial splitting, of the preceding one. Its performance is at the state of the art for English and for other languages. An interesting characteristic is that porting the Berkeley parser to a new language requires no additional effort apart from the availability of a treebank. Constituency parser performance is evaluated as usual by labeled precision (LP) and recall (LR) and F_1.

MaltParser is a data-driven dependency parser showing topmost performance in the multilingual track of the CoNLL shared tasks on dependency parsing in 2006 and 2007 and in the EVALITA 2009 dependency parsing task for Italian. Dependency parser performance is evaluated in terms of Labeled Attachment Score (LAS).

Statistical significance has been evaluated by using Dan Bikel's Randomized Parsing Evaluation Comparator[8]. This test checks whether the following null hypothesis can be rejected:

H_0: the difference in performance between the two experiments is not statistically significant.

To do so, the performance scores for the single sentences are shuffled between the two models, and then precision and recall are recomputed. The shuffling is repeated a large number n_t of times (up to $10,000$), and the number n_c of times where shuffling induced a variation in performance larger than the difference between the two models is counted. Eventually the probability p that the null hypothesis is incorrectly rejected is estimated by $p = \frac{n_c+1}{n_t+1}$. In other words, the difference in performance between the two models gets more statistically significant as long as the value of p gets smaller.

In order to study performance variations on sentences with different constituent order, the data set has been split in two parts: the former (SVO) includes all the sentences where the SVO constituent order is represented at least once; the latter (noSVO) includes all the

Data set	pattern	size
training set	SVO	646
	noSVO	2,379
	all	3,025
test set	SVO	110
	noSVO	390
	all	500

Table 7.2: Data set dimensions

other sentences, where all the other constituent orders are represented, but not the SVO. As shown in Table 7.2, the split of data between the two patterns is strongly unbalanced

[8] The tool is freely available from `http://www.cis.upenn.edu/~dbikel/software.html#comparator`

in favor of the noSVO, corresponding to nearly four times the number of sentences of the other pattern, both in the training and in the test sets. To overcome the difficulty of such unbalance, we therefore decided to randomly subsample the noSVO data sets to obtain a training and a test set with exactly the same dimensions of the SVO case. To avoid the risk of biased results, we repeated each experiment 20 times and averaged the corresponding outputs, obtaining the performance reported in Tables 7.3 and 7.4. A statistical significance test is applied to each iteration. The TUT data set is not divided in training and test set. Therefore assessment is performed by following the 10-fold cross validation protocol.

7.5.1 Constituency Parser

For constituency, parsing performance for Penn and APE formats is depicted in Table 7.3. The five macro columns correspond to the five different models, obtained by training the

	Training Set														
	All			**SVO**			**noSVO**			**sub-noSVO**			**balanced**		
Test Set															
Penn	LR	LP	F_1	LR	LP	F_1	LR	LP	F_1	LR	LP	F_1	LR	LP	F_1
SVO	81.75	81.37	81.56	72.34	71.49	71.91	79.39	78.10	78.74	69.73	67.95	68.83	76.87	76.41	76.64
noSVO	80.03	80.19	80.11	71.04	70.09	70.56	77.90	77.37	77.64	70.46	69.56	70.01	76.50	76.46	76.48
all	80.51	80.53	80.52	71.42	70.50	70.95	78.32	77.58	77.95	70.26	69.10	69.68	76.60	76.45	76.52
APE															
SVO	77.11	76.96	77.04	69.56	70.21	69.88	78.50	78.90	78.70	67.03	65.36	66.18	74.90	74.03	74.46
noSVO	79.26	79.47	79.36	72.02	72.12	72.07	79.18	78.92	79.05	70.12	69.06	69.59	75.71	75.42	75.57
all	78.69	78.88	78.78	71.57	71.47	71.52	79.02	78.92	78.97	69.34	68.12	68.73	75.51	75.08	75.29

Table 7.3: Constituency parser performance: comparisons between all pairs of models are statistically significant as $p \leq 0.05$.

parser on: (i) all the training set (All); (ii) only the SVO and (iii) the noSVO parts of the training data (SVO and noSVO respectively); (iv) by averaging performance on 20 runs made by subsampling the noSVO training set (sub-noSVO); and (v) by considering for training the union of the SVO training set and each of the sets in sub noSVO, and again averaging performance (balanced). For all the models, performance in terms of LP, LR and F_1 is reported. In all the cases, the null hypothesis can be rejected with values of p lower than 0.05 and then the comparisons between performance of all pairs of models result to be statistically significant. Also the standard deviation has been computed for all averaged cases (sub-noSVO and balanced) and its values are always lower than 3. The values have not been reported for providing more compact and readable tables.

First of all, note that the first column (All) represents a sort of baseline, where all available data are exploited. We can see how the addition of more detailed information, in APE with respect to Penn format, does not help parsing (except when the training set is composed by noSVO parse trees and in the two test sets there are noSVO and All examples),

probably because of the increased data sparsity. In fact, we would need a bigger treebank to accurately train the more precise APE labels. Furthermore, when comparing parsing performance on the SVO and noSVO data sets, we note that the Penn format favors the SVO pattern, while the APE favors the noSVO. This property is maintained also when training is performed either on SVO or on noSVO data alone, and this is quite surprising, but it probably still depends on the influence of the annotation and on the inclusion in the SVO data set of some noSVO pattern (SVO data set contains all and only the sentences containing *at least* one SVO pattern). On the other hand, when we consider the two models obtained by subsampling, namely sub-noSVO and balanced-train, the former always performs better on the corresponding noSVO test set. We can therefore conclude that the better performance of the noSVO model is also related to the fact that the training set is much larger than in the SVO case.

In general, we can conclude that the best choice is to include all the data available in the training set: indeed, this is the case with the best performance on both SVO and noSVO test sets. As a second choice, when the training sets are balanced, the best performance is obtained, as could have been expected, by training the parser on sentences as similar as possible to the ones composing the test set.

7.5.2 Dependency Parser

Also for the dependency paradigm, performance deteriorates when the information in the annotation augments, particularly for 3-Comp. Moreover, 3-Comp performance is much less stable than the other two cases, suggesting that we are in a data sparsity condition. We therefore decided to focus our analysis on 1-Comp and 2-Comp.

In general, in the dependency case performance remains more or less the same even when training is performed on sentences with a different constituent order with respect to the test set. In fact, in no comparison the value of p is small enough to guarantee the statistical significance of the differences, with the only exception of the difference between the models trained on noSVO and on SVO and tested on the SVO test set. In this case there is no statistical significance both with and without subsampling for the noSVO training set.

The fact that performance is only slightly sensitive to the different patterns suggests that the dependency paradigm is more robust than the constituency one with respect to variability in the constituent order and therefore more suitable to MRLs with such feature.

	Training Set				
	All	**SVO**	**noSVO**	**sub-noSVO**	**balanced**
Test Set					
1-Comp					
SVO	88.44	86.13	83.63	83.49	87.34
noSVO	87.62	86.87	82.43	83.95	85.57
all	87.86	86.65	83.63	83.81	86.08
2-Comp					
SVO	88.84	86.33	86.25	82.62	86.84
noSVO	86.72	86.45	81.11	82.91	84.77
all	87.34	86.41	82.62	82.82	85.38
3-Comp					
SVO	84.60	81.92	86.53	78.09	82.71
noSVO	83.10	82.55	76.87	78.72	80.83
all	83.54	82.86	81.98	78.53	81.38

Table 7.4: Dependency parser performance: Labeled Accuracy Score.

7.6 Conclusion and Future Work

The comparison between the preliminary results obtained with the constituency and with the dependency approaches suggests that the latter is more effective with respect to the free order of constituents than the former. The results should be considered as preliminary because of the limited size of the data set. Indeed, data sparseness hampers the reliability of results, especially for the most detailed annotation formats. As soon as more annotated data are available, we will be able to carry on new experiments that exploit more accurate annotation schemata, such as APE for constituency and 3-Comp for the dependency paradigm.

While we can expect that more annotated data will result in more reliable performance estimation, we do not think that the difference between constituency and dependency will substantially reduce. In fact, with more data, the number of different patterns is likely to grow more rapidly for the constituency paradigm, where different patterns are produced by different word orders, than for the dependency approach.

Another aspect that we plan to investigate is related to null elements. Usually they are removed before parsing, both for constituency and dependency[9]. Given that in Italian null elements occur quite frequently, it would be interesting to apply to Italian what was done for Korean in [20], for investigating the effects of taking into account null elements in parsing.

[9] See e.g. the standard CoNLL format, where null elements are not allowed.

Eventually, when enough data will be available, we could also consider the effect of variations between the different textual genres. Indeed, the TUT data set even now contains legal texts which are substantially different from, for examples, the kind of texts extracted from Wikipedia.

Chapter 8

Conclusions and Future Works

In this thesis we introduced new Boolean features called *Barrier features* (BFs), defined in Chapter 2 and based on PoS Tagging. These features were characterized by a pair ((trigger, endpoint), δ) where trigger and endpoint were two PoS tags and δ was a set of PoS between trigger and endpoint. Basically BFs were based on the set δ which represented a link between trigger and endpoint as described in Section 2.2.

BFs are Boolean features and a very large number of different BFs occur in an input sentence. The problem of data sparsity was solved using appropriate smoothing strategies as described in Section 2.3.

First of all, to reduce the number of BFs, when applying the classifier, we set the feature at *true* whenever the trigger and the endpoint PoS tags were the same and the PoS tag set was *included* in the one of the corresponding dictionary feature. This dictionary is built in unsupervised manner in the approaches for entity and relation classification (Chapters 3 and 4) as described in Section 2.3.1, using a large collection of English texts automatically labeled with PoS tags. While in the Twitter sentiment polarity classifier we do not use in this thesis the unsupervised construction of dictionary but we plan to use it in our future works.

Afterward, we extracted the features defined in the first step from the training and the test set corresponding to the specific task we were considering and we used these data to respectively build and assess the classifier. Moreover we considered the smoothing strategy based on the introduction of an UNKNOWN label [54] for each type of feature. The UNKNOWN feature has been trained by considering as UNKNOWN all features that have a number of occurrences lower than a given and chosen threshold in training set.

We applied the BFs to different tasks, namely the classification of semantic relations (see Chapter 3), entity classification and relation extraction (see Chapter 4) and Twitter polarity sentiment classification (discussed in Chapter 5). Indeed we showed in the experimental assessments their effectiveness in improving classification performance. The system proposed which employed the BFs overcame the state-of-art system and the same system trained without BFs. For these reason we can clinch that although BFs are based on PoS tagging, purely syntactic information, they are able to characterized also semantic aspects of information, such as the sentiments expressed by tweet or the entities and relations to classify.

To improve the performance of the entities and relation classifiers and to overcome their structural limits, as described in Chapter 4, we also proposed a new system, namely *Jointly Entity and Relation Extraction System (JERES)* in which the logical constraints, extracted by an predefined *Knowledge Base* (KB), are used together with the classifier outputs to build a probabilistic model based on GMs. JERES overcame the performance of both state-of-art system and the other our system, called *Pipeline system (PipeLs)*, based on a pipeline architecture composed by the entity and relation classifiers.

In Chapter 6, we have presented an approach to exploit syntactic information and automatically train a natural language parser for identifiers. The identifier parse trees generated by applying the trained parser are used to extract concepts and build an ontology that supports program comprehension. To investigate the impact of parser training on the quality of the concepts extracted, we conducted a comparative case study on three systems. The trained parser was compared with two off-the-shelf general purpose parsers, using the parser outputs to conduct a program understanding task, namely concept location. The result of the comparison showed that training a parser specifically for identifiers is beneficial for concept location, since the ontological concepts and relations extracted from the parser dependencies allowed us to formulate efficient queries, that reduced the search space during concept location.

In the Chapter 7 also studies the importance of syntactic information by comparing the preliminary results obtained with the constituency and with the dependency approaches and suggesting that the latter is more effective with respect to the free order of constituents than the former to investigate, in a cross-paradigm perspective, the influence of the constituent order on the data-driven parsing of Morphologically Rich Languages, as Italian.

In future we would like to make the porting of BFs in other language, such as Italian. Furthermore we plan to define new pairs of trigger and endpoint, perhaps based on the context right of the trigger too.

We exploit the BFs defined in this thesis in other IE or IR tasks. We perform some preliminary experiments about the introduction of BFs in another Software Engineering Task, namely Software Traceability . The traceability task is the process to retrieve a link (dependency) between two software artifacts (requirement document, test case document and so on), called *traceability link*. In our approach we combine some IR-based models, as Vector Space Model (VSM) and Latent Dirichlet Allocation (LDA), and we represent the artifacts with feature vectors which contain unigrams, BFs and other *syntactic features* based on dependency parsing. Preliminary experiments show that this approach overcomes the state-of-art systems.

Appendix A

Penn Treebank Tag Set

In this appendix we presented the tagset used in PoS tagging step of the BFs and in our approaches, described in this thesis. It is the Penn Treebank tag set [73]. In the following table we summarize the main PoS tags contained in this tag set.

ID	PoS Tag	Lexical Category
1	CC	Coordinating conjunction
2	CD	Cardinal number
3	DT	Determiner
4	EX	Existential there
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, comparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NP	Proper noun, singular
15	NPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PP	Personal pronoun
19	PP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	Particle
24	SYM	Symbol
25	TO	to
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
29	VBG	Verb, gerund or present participle
30	VBN	Verb, past participle
31	VBP	Verb, non-3rd person singular present
32	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34	WP	Wh-pronoun
35	WP$	Possessive wh-pronoun
36	WRB	Wh-adverb

Bibliography

[1] S. L. Abebe and P. Tonella. Natural language parsing of program element names for concept extraction. In *ICPC*, pages 156–159, 2010.

[2] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, LSM '11, pages 30–38, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[3] A. Alicante and A. Corazza. Barrier features for classification of semantic relations. In G. Angelova, K. Bontcheva, R. Mitkov, and N. Nicolov, editors, *RANLP*, pages 509–514. RANLP 2011 Organising Committee, 2011.

[4] G. Attardi, F. Dell'Orletta, M. Simi, and J. Turian. Accurate dependency parsing with a stacked multilayer perceptron. In *Proceedings of Evalita'09*, Reggio Emilia, 2009.

[5] T. Baldwin and H. Tanaka. The effects of word order and segmentation on translation retrieval performance. In *COLING*, pages 35–41, 2000.

[6] L. Barbosa and J. Feng. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 36–44, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[7] B. Beamer, S. Bhat, B. Chee, A. Fister, A. Rozovskaya, and R. Girju. UIUC: A Knowledge-rich Approach to Identifying Semantic Relations between Nominals. In *Proc. of SemEval07: the Fourth International Workshop on Semantic Evaluations*, pages 386–389, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[8] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):pp. 289–300, 1995.

[9] T. Biggerstaff, B. Mitbander, and D. Webster. The concept assignment problem in program understanding. In *Proceedings of the 15th International Conference on Software Engineering (ICSE).*, pages 482 –498, may 1993.

[10] D. Binkley, M. Hearn, and D. Lawrie. Improving identifier informativeness using part of speech information. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR)*, MSR '11, pages 203–206, New York, NY, USA, 2011. ACM.

[11] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[12] C. Bosco, V. Lombardo, L. Lesmo, and D. Vassallo. Building a treebank for Italian: a data-driven annotation schema. In *Proceedings of LREC'00*, Athens, Greece, 2000.

[13] C. Bosco and A. Mazzei. The Evalita 2011 parsing task: the constituency track. In *Evalita 2011 Working Notes*, 2012.

[14] C. Bosco and A. Mazzei. The Evalita 2011 parsing task: the dependency track. In *Evalita 2011 Working Notes*, 2012.

[15] C. Bosco, A. Mazzei, and V. Lombardo. Evalita parsing task: an analysis of the first parsing system contest for Italian. *Intelligenza artificiale*, 2(IV), 2007.

[16] C. Bosco, S. Montemagni, A. Mazzei, V. Lombardo, F. Dell'Orletta, and A. Lenci. Evalita'09 parsing task: comparing dependency parsers and treebanks. In *Proceedings of Evalita'09*, Reggio Emilia, 2009.

[17] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp. Mining Java class naming conventions. In *ICSM*, pages 93–102, 2011.

[18] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[19] Y. Choi, E. Breck, and C. Cardie. Joint extraction of entities and relations for opinion recognition. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 431–439, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[20] T. Chung, M. Post, and D. Gildea. Factors affecting the accuracy of Korean parsing. In *Proceedings of SPMRL 2010*, 2010.

[21] B. Cleary, C. Exton, J. Buckley, and M. English. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, 14:93–130, 2009.

[22] P. Clifford. Markov random fields in statistics. In G. Grimmett and D. Welsh, editors, *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley*, pages 19–32. Oxford University Press, Oxford, 1990.

[23] M. L. Collard, J. I. Maletic, and A. Marcus. Supporting document and data views of source code. In *Proceedings of the ACM symposium on Document engineering (DocEng)*, pages 34–41, New York, NY, USA, 2002. ACM.

[24] M. Collins, J. Hajic, L. Ramshaw, and C. Tillmann. A statistical parser of Czech. In *Proceedings of the ACL'99*, 1999.

[25] A. Corazza, S. D. Martino, and V. Maggio. Linsen: An efficient approach to split identifiers and expand abbreviations. In *Proceedings of the 28th IEEE International Conference of Software Maintenance (ICSM)*, pages 233 –242, 2012.

[26] A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. In *Proc. of ACL04: 42nd Meeting of the Association for Computational Linguistics , Main Volume*, pages 423–429, Barcelona, Spain, July 2004.

[27] D. Davidov and A. Rappoport. Classification of Semantic Relationships between Nominals Using Pattern Clusters. In *Proc. of ACL-08: HLT*, pages 227–235, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[28] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, pages n/a–n/a, 2012.

[29] M. S. Dryer. Word order. In T. Shopen, editor, *Clause Structure, Language Typology and Syntactic Description*, volume vol. 1. Cambridge University Press, 2007.

[30] R. J. Evans. A framework for named entity recognition in the open domain. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, editors, *RANLP*, volume 260 of *Current Issues in Linguistic Theory (CILT)*, pages 267–276. John Benjamins, Amsterdam/Philadelphia, 2003.

[31] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[32] Z. P. Fry, D. Shepherd, E. Hill, L. L. Pollock, and K. Vijay-Shanker. Analysing source code: looking for useful verbdirect object pairs in all the right places. *IET Software*, 2(1):27–36, 2008.

[33] G. Gay, S. Haiduc, A. Marcus, and T. Menzies. On the use of relevance feedback in ir-based concept location. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 351 –360, sept. 2009.

[34] J. Gimenez and L. Marquez. Fast and accurate Part-of-Speech Tagging: the SVM approach revisited, 2003.

[35] J. Giménez and L. Màrquez. SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of 4th International Conference on Language Resources and Evaluation (LREC)*, pages 43–46, 2004.

[36] R. Girju, P. Nakov, V. Nastase, S. Szpakowicz, P. Turney, and D. Yuret. Semeval-2007 task 04: Classification of semantic relations between nominals. In *Proc. of SemEval-2007: Fourth International Workshop on Semantic Evaluations*, pages 13–18, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[37] C. Giuliano, A. Lavelli, D. Pighin, and L. Romano. Fbk-irst: Kernel methods for semantic relation extraction. In *Proc. of SemEval-2007: Fourth International Workshop on Semantic Evaluations*, pages 141–144, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[38] C. Giuliano, A. Lavelli, and L. Romano. Relation extraction and the influence of automatic named-entity recognition. *ACM Trans. Speech Lang. Process.*, 5(1):1–26, 2007.

[39] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. Technical report, Stanford University, 2009.

[40] S. Grant, J. R. Cordy, and D. Skillicorn. Automated concept location using independent component analysis. In *Proceedings of the 15th Working Conference on Reverse Engineering (WCRE)*, pages 138–142, Washington, DC, USA, 2008. IEEE Computer Society.

[41] S. Green and C. D. Manning. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of COLING 2010*, 2010.

[42] J. H. Greenberg. Some universals of grammar with particular reference to the order of meaningful elements. In J. H. Greenberg, editor, *Universals of Language*, pages 73–113. MIT Press, London, 1963.

[43] M. Grella, M. Nicola, and D. Christen. Experiments with a constraint-based dependency parser. In *Evalita 2011 Working Notes*, 2012.

[44] S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus. Automatic query performance assessment during the retrieval of software artifacts. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 90–99, New York, NY, USA, 2012. ACM.

[45] S. Haiduc, G. Bavota, R. Oliveto, A. Marcus, and A. De Lucia. Evaluating the specificity of text retrieval queries to support software engineering tasks. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 1273 –1276, 2012.

[46] I. Hendrickx, S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 33–38, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[47] E. Hill, Z. P. Fry, H. Boyd, G. Sridhara, Y. Novikova, L. L. Pollock, and K. Vijay-Shanker. Amap: automatically mining abbreviation expansions in programs to enhance software maintenance tools. In *Proceedings of 5th International Working Conference on Mining Software Repositories (MSR)*, pages 79–88, 2008.

[48] E. Hill, L. L. Pollock, and K. Vijay-Shanker. Automatically capturing source code context of nl-queries for software maintenance and reuse. In *Proc. of Int. Conf. on Soft. Engineering*, pages 232–242, 2009.

[49] R. Hoehndorf, A.-C. N. Ngomo, S. Pyysalo, T. Ohta, A. Oellrich, and D. Rebholz-Schuhmann. Applying ontology design patterns to the implementation of relations in genia. In N. Collier, U. Hahn, D. Rebholz-Schuhmann, F. Rinaldi, and S. Pyysalo, editors, *Semantic Mining in Biomedicine*, volume 714 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[50] B. Hoffman. Integrating "free" word order syntax and information structure. In *Proceedings of EACL'95*, 1995.

[51] E. Jamison. Using grammar rule clusters for semantic relation classification. In *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*,

RELMS '11, pages 46–53, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[52] T. Joachims. Making large-scale support vector machine learning practical. In *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.

[53] M. I. Jordan. An introduction to probabilistic graphical models. *Computing*, M(10):1–14, 2002.

[54] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 1 edition, 2000. neue Auflage kommt im FrÃijhjahr 2008.

[55] N. Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for information extraction. In *Proc. of ACL04: Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, pages 178–181, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[56] F. Karlsson, A. Voutilainen, J. Heikkila, and A. Anttila, editors. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, 1995.

[57] R. J. Kate and R. J. Mooney. Joint entity and relation extraction using card-pyramid parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 203–212, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[58] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proc. of ACL 03 of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[59] D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *Proc of NIPS03: In Advances in Neural Information Processing Systems*, pages 3–10. MIT Press, 2003.

[60] E. Kouloumpis, T. Wilson, and J. Moore. Twitter Sentiment Analysis: The Good the Bad and the OMG! In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[61] K. Laitinen, J. Taramaa, M. Heikkila, and N. C. Rowe. Enhancing maintainability of source programs through disabbreviation. *Journal of Systems and Software*, 37(2):117–128, 1997.

[62] D. Lawrie and D. Binkley. Expanding identifiers to normalize source code vocabulary. In *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 113 –122, sept. 2011.

[63] D. Lawrie, D. Binkley, and C. Morrell. Normalizing source code vocabulary. In *Proceedings of the 17th Working Conference on Reverse Engineering (WCRE)*, pages 3 –12, oct. 2010.

[64] D. Lawrie, H. Feild, and D. Binkley. Extracting meaning from abbreviated identifiers. In *Proceedings of the 7th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, pages 213–222, 2007.

[65] L. Lesmo. The rule-based parser of the NLP group of the University of Torino. *Intelligenza artificiale*, 2(IV), 2007.

[66] L. Lesmo. The Turin University Parser at Evalita 2009. In *Proceedings of Evalita'09*, Reggio Emilia, 2009.

[67] D. Lin. Dependency-based evaluation of minipar. In *Proc. Workshop on the Evaluation of Parsing Systems*, Granada, 1998.

[68] D. Lin. LaTaT: Language and text analysis tools. In *Proc. of Int. Conf. on Human Language Technology Research*, pages 1–6, 2001.

[69] K.-L. Liu, W.-J. Li, and M. Guo. Emoticon smoothed language models for twitter sentiment analysis. In *AAAI'12*, pages –1–1, 2012.

[70] C. D. Manning and D. Klein. Optimization, maxent models, and conditional estimation without magic. In *HLT-NAACL*, 2003.

[71] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[72] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings of the 11th Working Conference on Reverse Engineering*, pages 214–223, Washington, DC, USA, 2004. IEEE Computer Society.

[73] M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistic*, 19(2):313–330, 1993.

[74] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 188–191, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[75] S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. A novel use of statistical parsing to extract information from text. In *Proc. of ANLP00: In 6th Applied Natural Language Processing Conference*, pages 226–233, 2000.

[76] S. Montemagni, F. Barsotti, M. Battista, N. Calzolari, O. Corazzari, A. Lenci, A. Zampolli, F. Fanciulli, M. Massetani, R. Raffaelli, R. Basili, M. T. Pazienza, D. Saracino, F. Zanzotto, N. Nana, F. Pianesi, and R. Delmonte. Building the Italian syntactic- semantic treebank. In A. Abeillè, editor, *Treebanks: Building and Using Parsed Corpora*, volume Treebanks: Building and Using Parsed Corpora, pages 189–210. Kluwer, 2003.

[77] A. Moschitti. A Study on Convolution Kernels for Shallow Statistic Parsing. In *Proc. of the ACL-04:*, pages 335–342, Barcelona, Spain, July 2004.

[78] A. Moschitti. Making tree kernels practical for natural language learning. In *EACL*, 2006.

[79] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *In Proceedings of Uncertainty in AI*, pages 467–475, 1999.

[80] G. Nenadié, S. Ananiadou, and J. McNaught. Enhancing automatic term recognition through recognition of variation. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 604, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[81] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Felix: Scaling inference for markov logic with an operator-based approach. *CoRR*, abs/1108.0294, 2011.

[82] J. Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160, 2003.

[83] J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. The CoNLL 2007 Shared Task on dependency parsing. In *Proceedings of the CoNLL 2007 Shared Task, EMNLPCoNLL*, pages 915–932, Prague, 2007.

[84] J. Nivre, J. Hall, and J. Nilsson. MaltParser: a data-driven parser-generator for dependency parsing. In *Proceedings of LREC-2006*, 2006.

[85] D. Ó Séaghdha and A. Copestake. Semantic classification with distributional kernels. In *Proc. of COLING-08: 22nd International Conference on Computational Linguistics*, Manchester, UK, 2008.

[86] A. Pak and P. Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In N. C. C. Chair), K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, and D. Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[87] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, Jan. 2008.

[88] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[89] M. Petrenko, V. Rajlich, and R. Vanciu. Partial domain comprehension in software evolution and maintenance. In *Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC)*, pages 13–22, 2008.

[90] S. Petrov and D. Klein. Learning and inference for hierarchically split PCFGs. In *Proceedings of AAAI (Nectar Track)*, 2007.

[91] H. Poon and P. Domingos. Joint inference in information extraction. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, AAAI'07, pages 913–918. AAAI Press, 2007.

[92] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC)*, pages 37 –48, june 2007.

[93] D. Poshyvanyk, M. Petrenko, A. Marcus, X. Xie, and D. Liu. Source code exploration with google. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM)*, pages 334–338, 2006.

[94] V. Rajlich. Intensions are a key to program comprehension. In *IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09.*, pages 1 –9, may 2009.

[95] V. Rajlich and N. Wilde. The role of concepts in program comprehension. In *Proceedings of the 10th International Workshop on Program Comprehension (IWPC)*, pages 271–280, 2002.

[96] D. Ratiu, M. Feilkas, and J. Jürjens. Extracting domain ontologies from domain specific apis. In *Proc. of European Conf. on Soft. Maintenance and Reengineering*, pages 203–212, 2008.

[97] B. Rink and S. Harabagiu. Utd: Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 256–259, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[98] D. Roth and W. Yih. A Linear Programming Formulation for Global Inference in Natural Language Tasks. In *Proc. of CoNLL-2004*, pages 1–8. Boston, MA, USA, 2004.

[99] D. Roth and W. Yih. Global inference for entity and relation identification via a linear programming formulation. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[100] H. Saif, Y. He, and H. Alani. Semantic sentiment analysis of twitter. In *The 11th International Semantic Web Conference (ISWC)*, pages 508–524, Boston, UA, 2012. Springer.

[101] D. A. Shamma, L. Kennedy, and E. F. Churchill. Tweet the debates: understanding community annotation of uncollected sources. In *Proceedings of the first SIGMM workshop on Social media*, WSM '09, pages 3–10, New York, NY, USA, 2009. ACM.

[102] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proc. of Int. Conf. on Aspect-Oriented Soft. Development*, pages 212–224, 2007.

[103] M. Speriosu, N. Sudan, S. Upadhyay, and J. Baldridge. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First Workshop on Unsupervised Learning in NLP*, EMNLP '11, pages 53–63, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[104] R. Steinberger. Treating 'free word order' in Machine Translation. In *Proceedings of COLING '94*, 1994.

[105] S. C. Tatikonda and M. I. Jordan. Loopy belief propagation and gibbs measures. In *In Uncertainty in Artificial Intelligence*, pages 493–500. Morgan Kaufmann, 2002.

[106] R. Tsarfaty, J. Nivre, and E. Andersson. Cross-framework evaluation for statistical parsing. In *Proceedings of the 13th Conference of the European Chapter of the ACL (EACL 2012)*, Avignon, France, April 2012.

[107] R. Tsarfaty, D. Seddah, Y. Goldberg, S. Kübler, Y. Versley, M. Candito, J. Foster, I. Rehbein, and L. Tounsi. Statistical parsing of morphologically rich languages (SPMRL) what, how and whither. In *Proceedings of SPMRL 2010*, 2010.

[108] M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33:58–64, December 2004.

[109] V. Vapnik. *Statistical learning theory*. John Wiley & sons, New York, 1998.

[110] S. Žitnik, L. Šubelj, D. Lavbič, A. Zrnec, and M. Bajec. Collective information extraction using first-order probabilistic models. In *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, pages 279–282, New York, NY, USA, 2012. ACM.

[111] A. Weber and K. Müller. Word order variation in German main clauses: a corpus analysis. In *Proceedings of the 20th International Conference on Computational Linguistics*, 2004.

[112] C. Whissell. The dictionary of affect in language. *Robert Plutchik and Henry Kellerman (Ed.), Emotion: Theory, Research, and Experience*, pages 113–131, 1989.

[113] D. C. Wimalasuriya and D. Dou. Ontology-based information extraction: An introduction and a survey of current approaches. *J. Inf. Sci.*, 36(3):306–323, June 2010.

[114] X. Yu and W. Lam. Jointly identifying entities and extracting relations in encyclopedia text via a graphical model approach. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 1399–1407, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[115] R. Zanoli and E. Pianta. Entitypro: exploiting svm for italian named entity recognition. *Intelligenza Artificiale - numero speciale su Strumenti per l'elaborazione del linguaggio naturale per l'italiano*, 4(2):69–70, (2007).

[116] M. Zhang, J. Su, D. Wang, G. Zhou, and C. L. Tan. Discovering relations between named entities from a large raw corpus using tree similarity-based clustering. In *Proc. of IJCNLP05: International Joint Conference on Natural Language Processing*, pages 378–389, 2005.

Dottorato in Scienze Computazionali e Informatiche
XXV Ciclo
Universitá degli studi di Napoli “Federico II”