Supervised Learning for Relationship Extraction From Textual Documents

João Pedro Lebre Magalhães Pereira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Examination Committee

Chairperson :	Prof. José Carlos Alves Pereira Monteiro
Supervisor:	Prof. Bruno Emanuel da Graça Martins
Co-Supervisor:	Prof. Helena Isabel de Jesus Galhardas
Member of the Committee:	Prof. David Manuel Martins de Matos

November 2013

Abstract

Information Extraction (IE) is the task of automatically extracting structured information from unstructured data, aiming to facilitate the use of said data by other applications. A typical sub-problem is the extraction of relationships from textual documents, which aims at identifying and classifying the relationships expressed between entities mentioned in the texts. In order to extract relationships from a raw text, it is important to pre-process the data, organizing the textual contents into useful data structures, with techniques from Natural Language Processing. Furthermore, since relationships are expressed between entities, it is mandatory to identify the entities using an entity extraction method, which is another sub-problem of IE.

Assigning a relationship type to a pair of entities can be seen as a classification problem. Therefore, supervised machine learning techniques can be applied. In this thesis, we used Support Vector Machines (SVM), which we trained with basis on online methods similar to Pegasos [27]. Two specific modeling choices have been tested. The first one is a simple online solution that trains SVM models considering a single kernel. The second approach is based on the idea of online multiple kernel learning. With existing datasets and common pre-processing tools, we formulated a benchmark, which was then used to evaluate kernel-based methods. We then implemented state-of-the-art kernels, specifically designed for relationship extraction. The results show that a multiple kernel learning solution obtains the best performance, and that multiple kernel learning solutions can perform better than heuristic solutions learning with a linear combinations of the same kernels.

Keywords: Relationship Extraction , Support Vector Machines , Online Learning , Multiple Kernel Learning

Resumo

Extracção de Informação é a tarefa de extraír automáticamente informação estruturada de dados não extruturados, visando facilitar o uso da mesma por parte de outras aplicações. Um típico sub-problema é a extracção de relações de documentos textuais, com o objectivo de identificar e classificar as relações expressas entre as entidades mencionadas no texto. De forma a extrair relações em texto, é importante preprocessar os dados, organizar os conteúdos textuais em estruturas de dados úteis com a utilização de técnicas de Processamento de Lingua Natural. Ademais, como as relações são expressas entre entidades, é necessário identificá-las, usando para isso um método de extracção de entidades, que é outro subproblema da extracção de informação.

Associar um tipo de relação, a um par de entidades, pode ser visto como um problema de classificação. Como tal, podemos aplicar técnicas de aprendizagem automática supervisionada. Nesta tese, usámos máquinas de vectores de suporte (SVMs), que treinámos com base em métodos online, semelhantes ao Pegasos. Testamos dois modelos específicos. O primeiro é uma simples solução online que treina modelos SVM considerando apenas um kernel. O segundo tem por base a ideia de aprendizagem online com múltiplos kernels. Com os bancos de dados existentes e um preprocessamento comum, formulámos uma benchmark a qual usamos para comparar e avaliar métodos baseados em kernels. Posteriormente implementámos os kernels do estado-da-arte, especificamente criados para a extracção de relações. Os resultados experimentais demonstraram depois resultados concordantes com uma melhor performance associada à aprendizagem de múltiplos kernels, em comparação com outras soluções heurísticas que apenas usaram combinações lineares do mesmo conjunto de kernels.

Palavras Chave: Extração de Relações , Máquinas de Vetores de Suporte , Aprendizagem Online , Aprendizagem com Multiplos Kernels

Acknowledgements

I do not think I could ever demand all the credits for this MSc thesis work. Therefore, there are a lot of people I would like to express my gratitude to. Because of their help, I feel this is their work too.

First of all, I would like to thank both my portuguese supervisors, Professor Bruno Martins and Professor Helena Galhardas. We started working, in fact, one year ago. I still remember their availability and helpfulness to discuss the topic when I was choosing the thesis. I thank you for your guidance in busy times, your patient when I was proceeding wrong, and by enlightening me with your experience and knowledge when I lacked fruitful ideas. I hope you are proud of this work too.

My thank you to Professor Erkki Oja from Aalto University, for delegating all the processes in Finland and for accepting to be my supervisor. His constant availability and knowledgeable feedback on this thesis were invaluable.

I would also like to thank my non-official supervisor, the Ph.D. student Gonçalo Simões for all the support and the time spent helping and guiding me in the, at times mysterious, matters of programming, writing and mathematics. It was a great pleasure to work with you, and learn from you. I remember fondly some chats and laughs we had, and I wish you the best luck for the future. I also hope you are proud of this work.

Another thanks goes to all the classmates that crossed my path. They provided me with good times that I will not forget. A special thanks to Fernando Santos, who remained working at TagusPark campus and was always ready to have lunch together or just to grab a snack.

Much obliged to all my group of friends for their understanding when I had to work and could not hang out with them and for the support when lady luck was nowhere to be found. Also a special thanks to Ricardo Ribeiro and Carlos Martins for writing tips and revising

To my parents, António Manuel Pereira and Maria Manuel Pereira for their support and dedication, and love, goes my sincere gratitude. It's undoubtedly due to them that I am here today, and am able to pursue my dreams.

I also would like to acknowledge the financial support that Fundação para a Ciência e Tecnologia (FCT) provided, through a scholarship in project SMARTIES (PTDC/EIA-EIA/115346/2009). Paramount for the

development of this project.

Finally,I would like to extend my acknowledgments to everyone that in a way or another helped in the development of this thesis. Thank you all very much!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Information Extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured textual documents. The structured information that we aim at extracting may correspond to entities, relationships, and/or attributes that describe said entities. Although IE concerns with processing human language texts by means of Natural Language Processing (NLP), being traditionally addressed by rule-based NLP methods, different areas of study have proposed alternatives and improvements. Examples include machine learning, information retrieval, databases, web information systems, and document analysis.

A specific IE sub-task is Relationship Extraction (RE). The objective of RE is to find semantic relationships between entities referenced in the text. Although more than two entities can participate in a relationship, this MSc thesis focus only on relationships between pairs of entities. For instance, from the sentence *Haifa, located 53 miles from Tel Aviv, will host ICML in 2010*, and considering the pair of entities *Haifa* tagged as a location and *ICML* as an organization, a system for RE should be able to find the relationship of *host* between *Haifa* and *ICML*.

The process that leads to relationship extraction starts with data pre-processing. A first step structures the unstructured text based on syntactic analysis. Then, an entity extraction step is needed to identify the candidate pairs of entities. Finally, a relationship extraction step is responsible for finding relationships between the entities identified. Relationship extraction is typically addressed as a classification problem for pairs of entities.

Different methods have been proposed to address RE. In contrast to the more traditional rule-based methods, this MSc thesis explores Machine Learning methods. Machine Learning (ML) methods aim at analyzing patterns in data and then training a model, in this case for the extraction of relationships. ML methods for RE can be feature-based or kernel-based. Feature-based methods make a direct extraction of characteristics (also known as features) from the data, which are then used to train models. Kernel-based methods rely on functions that make use of characteristics from the data to compute a similarity

value between pairs of data instances. Support Vectors Machines (SVMs) are machine learning models that can be trained using both a feature-based or a kernel-based method, and thus they are a natural choice for this work.

RE has been addressed in the scope of IE competitions that aim at evaluating different methods for solving a particular task. These competitions have resulted in a comparison between different methods and have a few issues. Firstly, they focus on a limited group of relationship extraction methods. Secondly, the data sets that were used focus on particular domains and use specific types of relationships. For instance, the reACE[1] corpora focuses on annotated English news, and the MUC[2] dataset contains documents related to aircrashes and missile launch scenarios. Another issue, is that the usability and technical characteristics of the pre-processing tools used are not a considered factor.

This MSc thesis develops a benchmark for relationship extraction, which attempts to overcome some of the issues found in the previous competitions focusing on the RE task. We were inspired by the work of Marrero et al. [18], who produced a benchmark for Named Entity Recognition (NER) systems. They developed their own validation metrics, as well as their own test and training corpora. This MSc thesis aims at producing a similar benchmark for relationship extraction, using publicly available datasets and a common set of pre-processing tools. With this benchmark, our goal is to compare several kernel-based supervised machine learning methods from the literature. Moreover, using the benchmark, we aim at proving the advantages of using an online approach for multiple kernel learning to train SVMs, by combining a set of kernels from state-of-the-art solutions for relationship extraction.

1.1 Hypothesis

The relationship extraction task can be handled with different techniques. Through this work, we evaluated the idea that machine learning techniques offer indeed an efficient approach to address relationship extraction. In particular, we purpose the implementation of techniques which use automatic combinations of multiple kernels from the state-of-the-art. These techniques can be more efficient than using a single kernel or a fixed combination of kernels.

Analogously to the benchmark for NER systems presented by Marrero et al. [18], this MSc thesis addresses the implementation of a benchmark for evaluating relationship extraction systems. The benchmark was used to compare implementations of several kernel-based supervised techniques from the literature on the subject, against distinct datasets from different domains. These datasets have been developed in the context of previous joint evaluation efforts, focusing on the relationship extraction task. Furthermore, a solution based on multiple kernel learning is also evaluated in this benchmark.

[1] http://catalog.ldc.upenn.edu/LDC2011T08
[2] http://catalog.ldc.upenn.edu/LDC2001T02

1.2 Methodology

In order to access the aforementioned hypothesis, we developed a benchmark that can provide us with the ability of fairly comparing different solutions for relationship extraction. This benchmark makes use of datasets currently existing in this area, from different domains. For instance, AImed[1] is a single relationship dataset which describes interactions between proteins. Another dataset is SemEval[2], composed of 9 relationships types and text retrieved from the web. Our benchmark also provides a common collection of NLP pre-processing tools for these datasets. Finally, the benchmark uses well known evaluation measures from the machine learning literature, that provide us a way to compare the performance of different methods.

We used this benchmark to evaluate different methods based on machine learning solutions. We implemented kernel-based methods form the state-of-the-art and we used them within an online procedure to train SVM classifiers. Moreover, we used this benchmark to compare different methods that use online multiple kernel learning.

1.3 Contributions

The main contributions of this MSc thesis are:

- Implementation of an online learning solution for training SVM models to be used to relationship extraction, and the development of an extension which allows the use of a kernel and a bias term.

- Development and implementation of an online multiple kernel learning solution, which can be used to train SVMs models for relationship extraction.

- Development of a benchmark for relationship extraction methods, that includes the use of datasets (AImed and SemEval), a set of measures, and the use of a common set of pre-processing NLP tools. The benchmark also includes the implementation of state-of-the-art kernel-based methods for relationship extraction.

- Application of the developed benchmark for comparing a set of kernel-based methods for relationship extraction. The results from an extensive set of experiments showed that although the best experiments are based on multiple kernel learning, we could not always assert the benefit of a multiple kernel based solution over a single kernel one.

[1] ftp://ftp.cs.utexas.edu/pub/mooney/bio-data/interactions.tar.gz
[2] http://semeval2.fbk.eu/semeval2.php?location=data

1.4 Document Organization

The rest of this dissertation is organized as follows: Chapter 2 presents the fundamental concepts in the area of relationship extraction. In particular, it focused on Machine Learning (ML) techniques. This chapter also summarizes the most important related work about the topic that is addressed in this thesis. Chapter 3 details the main contributions developed in this thesis, namely the online learning solution, the multiple kernel learning method, and a benchmark for the relationship extraction task. Chapter 4 presents the experimental validation procedure, and the results that were obtained. Finally, Chapter 5 draws conclusions about the work that was presented, in terms of results and contributions, and discusses some limitations and directions for future work.

Chapter 2

Background and Related Work

This chapter presents the main concepts and related work in relationship extraction and machine learning. In Section 2.1 we go through the usual pipeline of a relationship extraction system. First we analyze the different pre-processing operations and the resulting data structures, and then we discuss entity extraction methods.

In the relationship extraction module we use the previously processed information within a machine learning solution. Machine Learning (ML) is an area in artificial intelligence which solves problems through the finding of patterns in data, and the fundamental ML concepts are detailed in Section 2.2. In Section 2.2.1 we explain a supervised machine learning algorithm based in weighting relevant examples, called Support Vector Machines (SVMs). SVMs can work with methods which use only a set of features, or with more complex structures through the usage of kernels. Sections 2.3 and 2.4 cover respectively, the related work in feature-based methods and kernel-based methods. Section 2.5 summarizes the chapter and compares the methods in terms of results.

2.1 Overview on Information Extraction

The automatic extraction of information from unstructured and/or semi-structured data sources has been referred to as Information Extraction (IE). IE aims to organize structured information existing in natural language, and storing it in a form that allows further usage to be made by software applications.

A typical IE pipeline for extracting relationships from textual documents is shown in Figure 2.1. Given an excerpt of text as input, the goal is to identify pairs of entities mentions in the text and the relationships between them, if they exist. The typical IE pipeline is composed by three main steps, namely (i)

pre-processing; (ii) entity extraction; and (iii) relationship extraction. Before extracting entities and relationships from unstructured text, we have to pre-process the input text. The pre-processing step aims at annotating the text with linguistic and/or lexico-syntactic information. The output of the pre-processing step include sentences, tokens, part of speech (POS) tags, and other complex structures, such as syntactic parse trees and dependency graphs. After the pre-processing, the entity extraction step identifies the entities mentioned in the text, so that we can extract a relationship between them, if this relationship exists.

The following paragraphs detail each of the steps involved in the IE pipeline shown in Figure 2.1. As a complement to the explanation, Figure 2.2 illustrates the results obtained at each step of the pre-processing and entity extraction phases, for an example sentence in English.

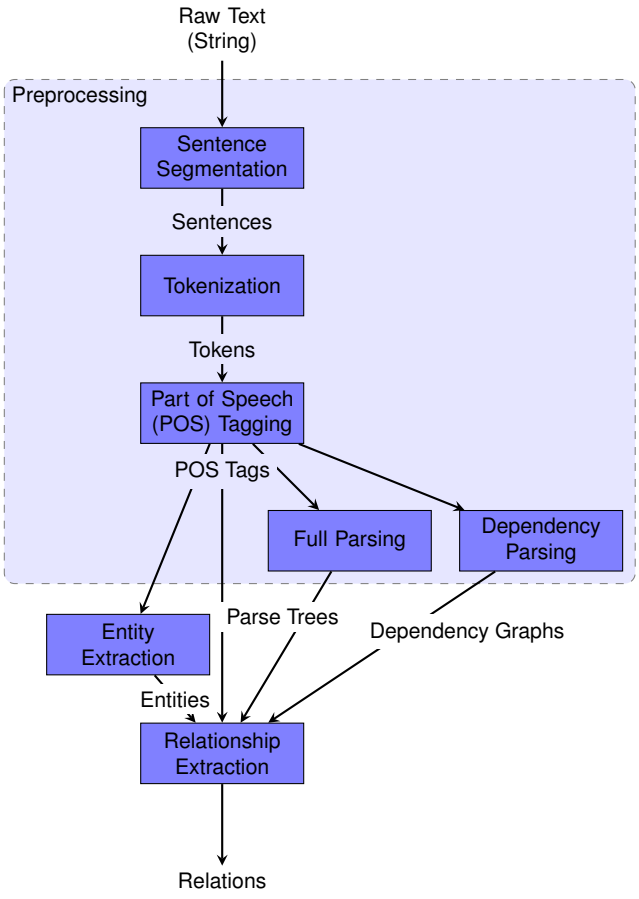


Figure 2.1: An information extraction pipeline for relationship extraction.

Sentence segmentation is the process of separating the sentences present in a given unstructured text (i.e., segmenting a textual string into a list of sentences). This is not an easy problem, because there is not a standard way of separating sentences. For example, a period mark can denote an abbreviation, a decimal point, or an email address, among other possibilities. To disambiguate these cases, there are strategies based on rules, such as the Vanilla approach by Mikheev [19], or approaches based on

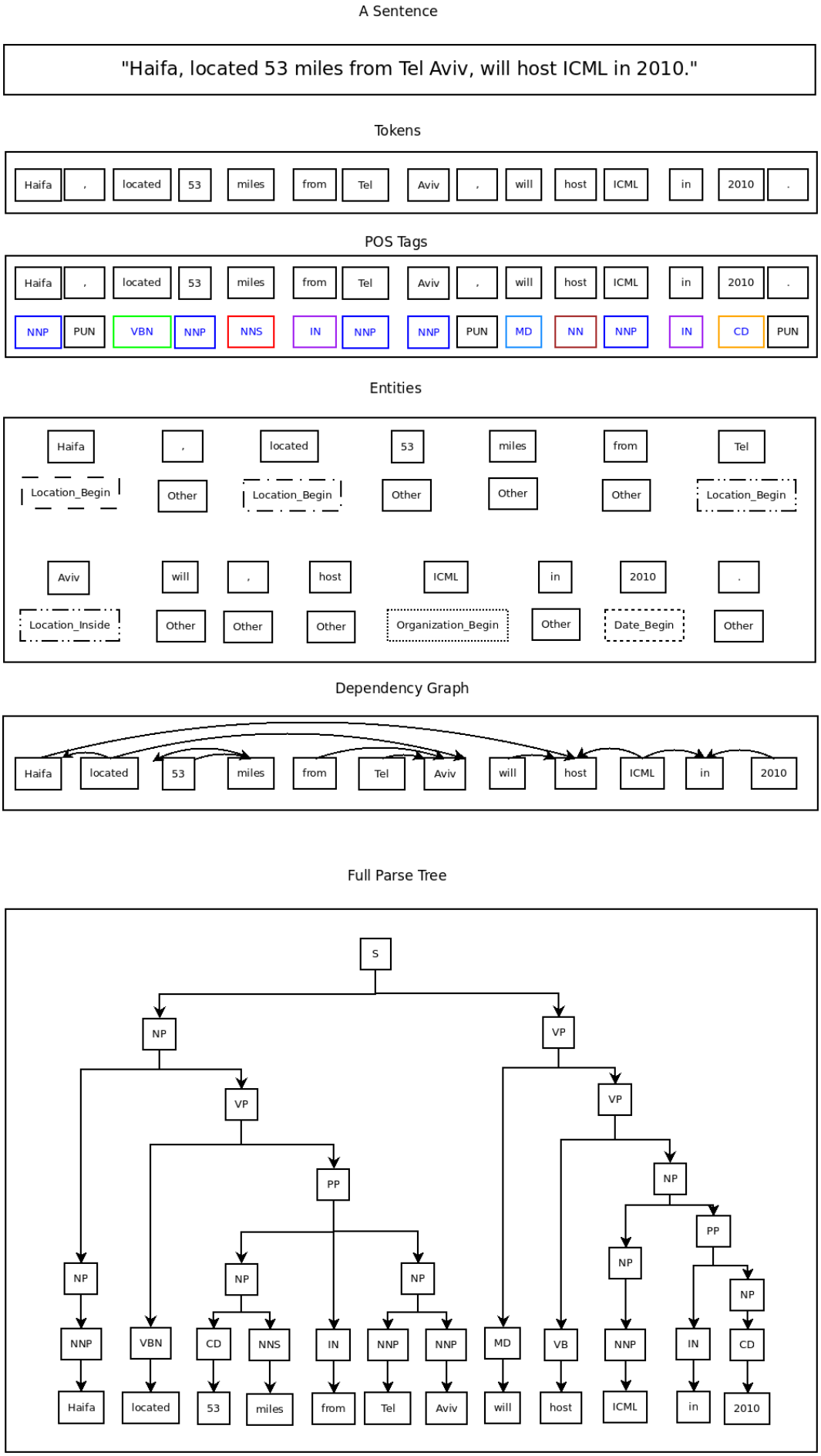


Figure 2.2: An example for the different phases involved in preprocessing and entity extraction.

Machine Learning (ML), such as the maximum entropy model from the SATZ architecture by Palmer [23].

The *tokenization* step consists on identifying the different tokens in each sentence. A token can be either a word, a number, or a punctuation mark. Tokenization can be done by splitting the sentence using delimiters like spaces, commas, and dots. It can be easily performed with a standard lexical analyzer [26] for languages that use the Latin alphabet. For other languages, like Japanese or Chinese, where the words are not separated by spaces, the problem becomes more complex as formulated by Huang et al. [12].

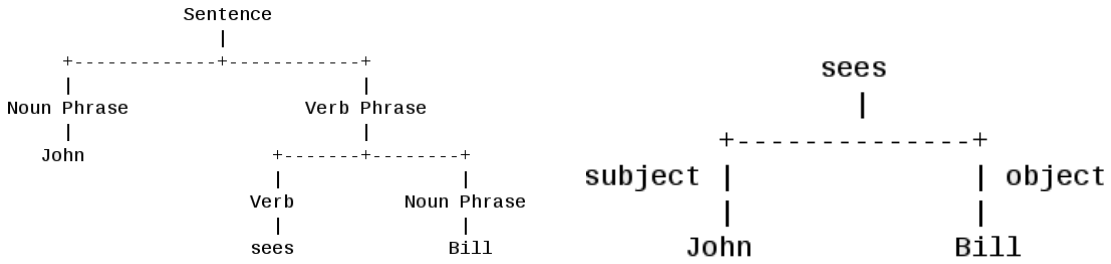
Part of Speech (POS) tagging can be used to associate words with a grammatical category. Through this process one can, for instance, identify verbs, which are valuable elements to determine relationships in text. POS taggers can either be:

Rule-Based taggers: These techniques use a dictionary to annotate words with a tag from a list of alternatives. In order to disambiguate between the possibilities, these systems typically use rules that are manually written by linguists. Rule-based POS-taggers have been described by Jurafsky and Martin [14];

Brill Taggers: These POS taggers are similar to the rule-based ones, but they use statistics instead of rule-based techniques to disambiguate between different possibilities. These taggers associate the most frequent tag to each token. In order to minimize the errors, these taggers try to disambiguate different possibilities by replacing tags through the usage of transformation rules. The first author describing this approach was Brill [3];

Stochastic Taggers: These taggers are based on probabilities and techniques for sequential classification, such as Hidden Markov Models (HMMs) [28]. They obtain statistical information by analyzing a corpora. For instance an algorithm like the one proposed by Viterbi can be used to identify the most probable sequence of tags [8] for a given sentence.

Full parsing produces parse trees which group the words from a given sentence into a syntax tree (e.g, a phrase has a verb, a verb can have complements, etc). This tree is constructed through the relationships among the grammatical classes of each word. For instance, a subject and a determinant are grammatically related through a higher grammar level label, i.e. a noun phrase. The grammar labels directly defining a word token are called terminals, while the others are called non-terminals. Non-terminals are related to more general ones, for instance a noun-phrase and a verb-phrase will be part of a sentence that, in this context, is a non-terminal and head of the tree. The construction of the tree can be performed through a grammar-based method, using either a top-down or a bottom-up approach. These grammars are usually context-free grammars (CFG), in contrast with the context-sensitive grammars. CFGs are defined through rules, so it is expected that their construction will be about following grammatical rules. Parsers are algorithms used to construct parse trees. An example is



(a) Parse Tree (b) Dependency graph

Figure 2.3: A parse tree and a dependency graph for the sentence *John sees Bill*.

the Earley's parser [7], that constructs a CFG parse tree. An example of a parse tree for the sentence *John sees Bill* is represented in Figure 2.3(a).

Dependency parsing produces a dependency graph for each sentence. In a dependency graph, a word that depends on other word will be linked. Nodes in the graph correspond to words, and edges correspond to links between words. The links represent grammatical dependencies between the words. For example, in a phrase, a determinant before the noun is linked to the noun, and the noun can be linked to a verb. In the literature, some authors view a dependency graph as a tree, where the head is usually represented by the main verb node.

Dependency graphs are useful for extracting features to be used in relationship extraction. Dependency graphs are usually less expensive to produce than parse trees, but most of the times they contain enough information to support the extraction of useful features. The Berkeley Parser is an example of a parser that generates dependency graphs [24]. In relationship extraction, sometimes it is useful to create a dependency graph for each pair of entities in the same sentence. MXPOST [25] is a maximum entropy statistical parser, and one can also convert parse trees to dependency graphs. An example of a dependency graph for the sentence *John sees Bill* is shown in Figure 2.3(b).

The *entity extraction* step classifies elements (i.e., particular sequences of tokens) from the text, as for example proper names, dates, values, or other types of entities. Entity extraction methods belong to one of two types: rule-based and statistical.

Rule-based methods use hand made rules (e.g, regular expressions), usually written by an expert. The different approaches vary in the way they solve conflicts, and a typical strategy is to order rules by priority. *Statistical models*, on the other hand, are based on probabilistic approaches and require training corpora. Most Statistical models rely on either token-level methods or segment-level methods [22].

In *token-level* models, the text is split into sorted tokens, and then each token is assigned to a label. Since one or more tokens can be part of the same entity, there is a need to aggregate them. To solve this problem, additional information about the token's location in the entity is assigned to the label, through an encoding procedure. Encoders create new labels based on the original ones, by adding information to the end of the label. BCEO and BIO are examples of encoders, where the first one adds

Begin, *Continue*, *End* or *Other* to the end of each label, while the second one, BIO encoding, adds *Begin*, *Inside* or *Other*, instead.

Segment-level models differ from token-level ones in the type of structure used. Instead of using tokens and encoders, segment-level models make use of segments. The segments are structured in such a way that each segment will be given an entity label. These segments can contain the multiple words that represent a entity [10].

Both token-level models and segment-level models need to use a general Machine Learning (ML) method for performing the actual classification, for example Hidden Markov Models (HMMs). The state-of-art method for assigning labels are Conditional Random Fields (CRFs), which are based on Markov Models and have been described by Lafferty et al. [16].

The *relationship extraction* problem aims at identifying a relationship r among a set of entities. This problem can be multi-way, in the sense that a relationship may exist between three or more entities, and it can be binary, focusing on relations between pairs of entities. Multi-way relationship extraction is a more complex problem and is not the aim of this MSc thesis. This thesis will explore binary relationship extraction, meaning that we have two entities e_1 and e_2, and a set of possible relationships r between them.

Binary relationship extraction can be separated into two main problems. One of the problems is validation, where given a relationship r and the two entities, the system should correctly answer if there is a relationship r between these entities. Another problem deals with, given two entities e_1 and e_2, identifying a relationship r, if it exists. These problems can both be seen as classification tasks, where the first one involves identifying the type of relationship, and the other involves classification as yes or no, depending on whether the relationship exists.

Various methods have been proposed to extract relationships. The first methods were defined as rule-based approaches, where a linguistic expert defines rules manually. These rules require hard work, and although these systems can work well in specific contexts, the complexity rises when more general contexts are to be treated. In order to find better methods, techniques from other areas have been applied to relationship extraction. Machine Learning (ML) is an artificial intelligence area that has been successfully applied to various problems, and also to relationship extraction. ML methods for relationship extraction will be discussed in the next section.

2.2 Machine Learning Algorithms for Relationship Extraction

Machine Learning (ML) refers to a set of algorithms which optimize a performance criterion, using example data or past experiments. A model is defined in terms of some parameters, and learning is the execution of a computer program to optimize the parameters of the model [2]. These programs usually

have two phases. In the first phase, they use input data to estimate the parameters for complex models or functions. If we consider a line in a two dimensional space given by $y = b + mx$, a ML process would for instance corresponds to finding the best parameters b and m that represent some linear data. The second phase uses the model and the parameters estimated in the previous phase to predict further behaviors for new data.

Machine Learning (ML) can also be formalized as the task of predicting a response variable y given the values of a vector of predictor variables x. Let X denote the domain of x and Y the domain of y. If for the first phase we have a data set with pairs (x, y) then this is a *supervised method*, because we know for a given x, the y we are aiming to obtain. The cases where we do not have pairs (x, y), but instead when just variables x are available, are referred to as *unsupervised*.

Supervised methods usually divide the available data into two sets, namely a training set and a test set. The training set is used for the first phase, where the parameters are to be estimated. In the second phase, the objective is to evaluate the behavior of the learned model. The test set is used as input and then the output resulting from the model is compared with the known output from the data, to calculate an error measure.

Classification and *regression* are perhaps the two most important problems in supervised machine learning. If y is a continuous or discrete variable taking real values, the problem is called regression. Otherwise, if the domain of y, Y is a finite set of unordered values, the problem is called classification. If Y contains two unordered values the problem is typically referred to as binary classification, whereas if $|Y| > 2$ the problem is referred to as multi-class classification.

In mathematical terms, the classification and regression problems can be reduced to a task of finding a function $d(x)$ that maps each point in X to a point in Y. The construction of $d(x)$ usually requires the existence of a training sample of n observations $L = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ (i.e., supervised learning is the more typical setting). The criterion for choosing $d(x)$ is usually the expected missclassification cost for the case of classification problems. If Y contains J distinct values, the classification solution (i.e., the classifier), may be written as a partition of X into J disjoint pieces $A_j = \{x : d(x) = j\}$ such that $X = \cup_{j=1}^{J} A_j$.

The relationship extraction task can be seen as a classification problem. Extracting a relationship means to classify a pair of entities. Machine Learning (ML) algorithms can be executed over an annotated corpora to create a model which will be able to predict the existence of relationships or their types. The different characteristics of the input type x allow us to split the ML methods in two groups namely, the ones that use simple features vectors, which we named feature-based and present in Section 2.3, and those that use structured representations (e.g, a graph or a tree), which we named kernel-based and present in Section 2.4. Section 2.2.1 presents Support Vectors Machines (SVMs), a Machine Learning (ML) algorithm that can be trained with both feature-based and kernel-based methods.

2.2.1 Support Vector Machines

Support Vector Machines (SVMs) are a family of supervised learning methods that analyze data and recognize patterns. They offer a straightforward engineering solution for classification tasks. SVMs construct a separating hyperplane in a high-dimensional feature space, aiming at maximizing the separability. SVMs express the hyperplane in the original space using a small set of training vectors, called the *support vectors*. For a given finite set of learning patterns, the optimum separation hyperplane is the linear classifier with maximum margin between the positive and the negative samples. This hyperplane maximizes the margin of separation. Considering two pattern classes that are linearly separable, we can have a set of N training samples $\{x_i, d_i\}_{i=1}^{N}$, where the desired response is $d_i = +1$ for vectors belonging to the first class, and $d_i = -1$ for the training vectors in the second class. After defining a weight vector w and a bias b, the margin of separation corresponds to a hyperplane given by $w^T x_i + b = 0$. The optimal margin of separation is the one where the values for w and b are those that best separate the classes, which means that $w^T x_i + b \geq 0$ for $d_i = +1$ and $w^T x_i + b < 0$ for $d_i = -1$.

The problem of finding a separating hyperplane can be solved using the method of Lagrange multipliers from optimization theory. Constructing the Lagrangian function, and then maximizing it with respect to the Lagrange multipliers α_i, $i = 1, 2, ..., N$, we obtain the resulting objective function given in Equation (2.1). In the end, the optimum decision function is given by $w_o^T + b_0$, where w_o denotes the optimal weight vector calculated from Equation (2.2), and b_o denotes the optimal bias given by Equation (2.3).

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j x_i^T x_j \tag{2.1}$$

$$w_o = \sum_{i=1}^{N} \alpha_{o,i} d_i x_i \tag{2.2}$$

$$b_o = \frac{1}{Nsv} \sum_{i=1}^{Nsv} (w \cdot x_i - d_i), \text{ where } Nsv \text{ represents the number of support vectors} \tag{2.3}$$

The *optimal* decision surface is in general nonlinear, and thus not a hyperplane. The data vectors are also, in general, non-separable, with some inevitable classification error. The solution is to map the data vectors x into a high-dimensional feature space, and to construct an optimal separating hyperplane in that feature space. The *kernel trick* is used to compute the corresponding nonlinear decision surface in the original smaller dimensional input space. The most used kernels are the polynomial function, given by Equation (2.4), the Radial basis function network given by Equation (2.5), and the two-layer perceptron given by Equation (2.6).

$$K(x, x_i) = (x^T x_i + 1)^p, p \text{ is specified by the user.} \tag{2.4}$$

$$K(x, x_i) = exp(-\frac{1}{2\sigma^2}||x - x_i||^2), \sigma^2 \text{ is specified by the user.} \quad (2.5)$$

$$K(x, x_i) = tanh(\beta_0 x^T x_i + \beta_1), \beta_0 \text{ and } \beta_1 \text{ are specified by the user.} \quad (2.6)$$

In Relationship Extraction, the problem of finding the relationship type is usually not modeled as a binary classification task. Thus, before being able to use SVMs, it is necessary to reduce the relationship extraction problem to a binary classification process. There are two solutions for this problem. One is to assign an SVM to each specific class. Each SVM will give the probability of the candidate instance being part of each class. In the end, we use a winner-takes-all strategy, in which the classifier with the highest output function assigns the class. The other solution is to assign a SVM for each pair of classes. Then, classification is performed by a max-wins voting strategy, in which every classifier assigns the candidate instance to one of the two classes, then the vote for the assigned class is increased by one vote, and finally the class with the most votes determines the instance classification.

2.3 Feature-Based Methods

Features are characteristics and attributes derived from data. For example, a simple common feature could be if a key word is presented or not in a sentence. Feature-based methods for relationship extraction use a set of features represented by a vector x as input. For instance, considering a set of words $w = \{Haifa, ICML, miles\}$ each position of a vector x could represent the availability of a key word w_i. Features for relationship extraction are usually extracted from previously created text structures sequences, full parse trees, or dependency graphs. Considering these three text structures as graphs, the nodes are words and edges indicate a path in the structure. For instance, in a sequence structure, each node is linked to the following node in the sentence. An analysis of the graph can result in a feature set that can be used to train machine learning models based on statistics. The approach named Statistics for Information from Text (SIFT) was the first method to follow this approach [21]. It uses its own ML algorithm based on the Bayes theorem. This model uses structural and content features. For instance, the candidate pair (*Haifa*, *ICML*) in the sentence *"Haifa, located 53 miles from Tel Aviv, will host ICML in 2010"* will have the value of zero for the *distance between entities* feature. Other models have also been created, with an example being maximum entropy models that combine lexical, syntactic and semantic features [15]. For example, using the same candidate pair (*Haifa*, *ICML*), the *overlap* feature would return the value 8. The definition of successful features requires some knowledge about natural language, and feature usefulness may change when context changes.

2.3.1 Statistic ML System with Syntactic and Semantic Information

Miller et al. [21] described the first machine learning approach to relationship extraction, named Statistics for Information From Text (SIFT). Associated to SIFT, an entity extraction system was also developed, as well as a pre-processing system that includes its own parser and POS-tagger. This model incorporates syntactic and also semantic information, which is useful to retrieve the features.

The features used in SIFT are presented in Table 2.1, and separated into structural and content features. The features are used to train a ML model to predict relationships in a text. The authors developed their own ML model, through a formalism they named Message-Level Processing. The model is based on the Bayes theorem. Given the measured features for a possible relation, the probability of a relation holding is computed through Equation (2.7), and the probability for it not holding is computed through Equation (2.8), where $p(feats)$ is the probability of a feature $feats$ appearing in a sentence, $p(rel)$ is the probability of happening a relationship and $p(feats|rel)$ is the probability of a feature $feats$ appearing in a sentence that holds a relationship. The symbol ($\sim$) indicates negation. Equation (2.9) is produced by merging the holding and not holding equations, and then by considering feature independence, by taking the product of the contributions for each feature. If the ratio is greater than 1, the model predicts a relation. Moreover, to avoid zero values, the probabilities were smoothed by mixing them with 0.01% of a uniform model.

This system was the only contestant in the MUC-7 competition presented in Section 3.3.1, for the relationship task. The evaluation of this system, using the MUC-7 dataset, achieved a Recall of 64%, a Precision of 81% and an F-Measure of 71.23%.

$$p(rel|feats) = \frac{p(feats|rel)p(rel)}{p(feats)} \tag{2.7}$$

$$p(\sim rel|feats) = \frac{p(feats|\sim rel)p(\sim rel)}{p(feats)} \tag{2.8}$$

$$\frac{p(rel|feats)}{p(\sim rel|feats)} = \frac{p(feats|rel)p(rel)}{p(feats|\sim rel)p(\sim rel)} = \frac{p(rel)\prod_i p(feats_i|rel)}{p(\sim rel)\prod_i p(feats_i|\sim rel)} \tag{2.9}$$

2.3.2 Combining Lexical, Syntactic and Semantic Features

Kambhatla [15] proposed an approach based on a type of statistical ML algorithms, named maximum entropy models, that combines lexical, syntactic and semantic features. The authors built maximum entropy models for predicting the type of relation (if any) between every pair of mentions within each sentence. For each pair of mentions, they compute several features as described in Table 2.2. All the

Table 2.1: Features used in the work of Miller et al. [21].

Feature Name	Description
Structural features	**Properties of the entity position on the text**
Distance between entities	Integer that takes value 0 if the entities are in the same sentence, 1 if they are in neighborhood sentences and 2 if they are more far away than neighborhood sentences
Topic sentence	Boolean that indicates if one of the candidate entities is in the first sentence of the text
Content features	**Properties that a pair of entities in the training data have in common with the candidate entities**
Name similarity	Boolean that indicates if there is a pair of related entities in the training data for which one entity shares at least a name with the candidate entities
Descriptor similarity	Boolean that indicates if there is a pair of related entities in the training data for which one entity shares at least a name and the other shares at least an entity type
Inverse name similarity	Boolean that indicates if there is a pair of related entities in the training data for which one entity shares at least a name and the other does not share any name

syntactic features are derived from a parse tree and a dependency tree.

This approach was trained and tested with the ACE[1] corpus. The reported results for the evaluation of 2003 version was of 55.2% in terms of the F-Measure.

Table 2.2: Features used in the work of Kambhatla [15].

Feature Name	Description
Words	The words of the entities in the relationship
Entity type	The type of both entities
Mention Level	The entity identification as a NAME, a NOMINAL or a PRONOUN
Overlap	The number of words between the entities
Dependency	The words, the part-of-speech tags and the chunk labels of the words on which the entities are dependent
Parse Tree	The path of non-terminals connecting the entities

2.3.3 A Systematic Exploration of the Feature Space

Jiang and Zhai [13] developed an approach that tried to use the most considerable amount features from the literature, and then to select the best features by testing and analyzing the results and also the relative performances. These features are collected from a sentence structure, and also from complex structures, namely dependency graphs and parse trees. For each one of these structures, the method

[1] http://www.nist.gov/speech/tests/ace/

considers a representation of an independent graph, where the nodes represent tokens and edges represent relations between nodes. A node also contains properties of the token, such as its morphological base form, its POS tag, and its semantic class.

A sentence is represented by every token in form of a node in position i connected to the next token in position $i + 1$, representing the next word in the sentence. A dependency graph will simply use these links as edges. The parse tree will use additional nodes to represent syntactic categories, such as an NP or VP characteristics from a parse tree. Thus, these edges will link the token nodes to their parent category, and each category node with their parents, until reaching the head of the parse tree.

By using these graphs it is then possible to collect the features described in Table 2.3. The features are used to train a ML algorithm, the authors tested with a maximum entropy model and a support vector machine. They also defined tests in order to select the best features in terms of both accuracy and computational performance.

This system was tested with the 2004 version of the ACE corpus and, for each combination of features extracted from sequence, parsing tree, and/or dependency graph. The best results obtained were for the combination of all structures, with 68.9% of Precision, 68.6% of Recall, and 68.8 of F-Measure. For instance, the combination of sequence and parsing tree was close to the best combination set, with a Precision of 69.1%, a Recall of 68.6% and a F-Measure of 68.8%.

Table 2.3: Features used in the work of Jiang and Zhai [13]

Feature Name	Description
Entity attributes	A subgraph that contains only the node representing the head word of the argument, labeled with the entity type or entity mention type
Bag-of-Words	A subgraph that contains a single node labeled with the token
Bigrams	A subgraph consisting of two connected nodes from the sequence representation, where each node is labeled with the token
Grammar productions	Sequences of grammar productions, that is, complete subtrees of the syntactic parse tree
Dependency relations	Edges connecting two nodes from the dependency tree
Dependency path	A path in the dependency tree connection two nodes that represent two arguments

2.4 Kernel-Based Methods

A kernel function can be defined as a comparison function between two instances. One of these instances can be seen as an example given from a training set, and the other as the one we want to predict a class for. Basically, a kernel method uses a pair of entities and a relationship, and searches the known data set (input) for similar examples. The samples from the known data set are previously weighted by relevance. In order to predict a sample x, we compute the kernel value of the sample against every support vector (samples from the training data with weight different from zero) and we

multiply these values by the weight associated to each support vector. The operation's result will produce the output we are trying to predict. Kernel functions usually make use of preprocessed structures, like sequences, dependency graphs, and parse trees. These structures can be seen as graphs where nodes are tokens and the links depend on the structure. It is a normal practice that these nodes are enriched with a set of features (i.e. {POS-tag, token lemma, entity type}) . When comparing a node with another instead of looking only to the token, we can also count the number of features. For instance, a token *forces* enriched with {noun, force, ARG-A} and a token *Tikrit* enriched with {noun, Tikrit, ARG-B} share one feature in four (the token can be counted as a feature).

An example of a kernel method for relationship extraction that uses sequences involves comparing the words between and/or in the surrounding context (before-between or between-after) [4]. This method then identifies the common attributes from sequences in the training data with the sequences we want to extract the relationship from. Kernels that use dependency graphs are similar to those that use sequences, but they compare the path of words between the entities, instead of the words between the entities in a sentence.

This section will next present different kernels from the literature. The subsequence kernel in Section 2.4.1 is made of three sub-kernels which cover different parts of a sentence. Section 2.4.2 describes the dependency tree kernel, where the trees that contain the pair of entities are compared in a sparse way. Section 2.4.3 describes the shortest path dependency kernel that compares the shortest paths from an entity to another in each dependency graph of the candidate sentences. A solution which takes into account both sequence and dependency graph structures is presented in Section 2.4.4. Finally, the Bag-Of-N-Grams kernel is a simpler but efficient kernel that uses bags of words and words segments around the entities, and it is explained in Section 2.4.5.

2.4.1 Subsequence Kernel

Bunescu and Mooney [4] presented a kernel-based method based on subsequences. In a sentence, the word defining the relationship can appear not only between the two entities, but also before (e.g., activation of *P1* by *P2*) and/or after (e.g., *P1* and *P2* interact). Therefore, to take care of these cases, this kernel is separated into the three following aspects:

Fore–Between: Words before and between the two entity mentions are simultaneously used to express the relationship;

Between: Only words between the two entities are essential for exposing the relationship;

Between–After: Words between and after the two entity mentions are simultaneously used to express the relationship.

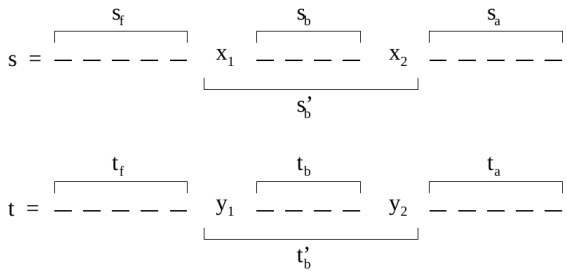


Figure 2.4: Computation of the final sequence kernel.

To explain the cases Fore-Between, Between, and Between-After, we first introduce a kernel from Lodhi et al. [17]. This author describes a kernel to be applied to two text documents, where the idea is to compare them by means of the substrings they contain. The more substrings in common, the more similar the documents are. These substrings do not need to be contiguous, and the degree of contiguity of one such substring in a document determines how much weight it will have in the final kernel computation.

Formally, let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For strings s and t, we denote by $|s|$ and $|t|$ the length of the string $s = s_1...s_{|s|}$ and $t = t_1...t_{|t|}$. The sequence $s[i:j]$ is the contiguous subsequence $s_i...s_j$ of s. Let $\mathbf{i} = (i_1, ..., i_{|\mathbf{i}|})$ be a sequence of $|\mathbf{i}|$ indices in s, in ascending order. We define the length $l(\mathbf{i})$ of the index sequence $\mathbf{i}$ in s as $i_{|\mathbf{i}|} - i_1 + 1$. Similarly, $\mathbf{j}$ is a sequence of $|\mathbf{j}|$ indices in t.

Bunescu and Mooney [4] formalized the kernel shown in Equation 2.10, based on two fixed index sequences i and j, being both of length n. Given two feature vectors $x, y \in \Sigma$, let $c(x, y)$ denote the number of common features between x and y. The parameter λ is used as a decaying factor that penalizes longer subsequences. For sparse subsequences, this means that wider gaps are more penalized, which is exactly the desired behavior.

$$K_n(s, t, \lambda) = \sum_{\mathbf{i}:|\mathbf{i}|=n} \sum_{\mathbf{j}:|\mathbf{j}|=n} \prod_{k=1}^{n} c(s_{\mathbf{i}_k}, t_{\mathbf{j}_k}) \lambda^{l(\mathbf{i})+l(\mathbf{j})} \tag{2.10}$$

Given two strings s and t , let x_1 and x_2 be the entities in s, and let y_1 and y_2 the entities in t. Let us define s_b as the tokens between the entities of s, s_f the tokens before the entities, and s_a the tokens after. The same is applied to string t. This notation is represented graphically in Figure 2.4. The computation of the final kernel $rK(s, t)$ is expressed in Equation (2.15), which will contain the sum of the three aspects, Fore-Between as in Equation (2.12), Between as in Equation (2.13) and Between-After as in Equation (2.14). Both aspects share the computation of the substrings between the entities as in Equation (2.11).

$$bK_i(s, t) = K_i(s_b, t_b, 1) \cdot c(x_1, x_2) \cdot c(y_1, y_2) \cdot \lambda^{l(s_b)+l(t_b)+4} \tag{2.11}$$

$$fbK(s,t) = \sum_{i,j} bK_i(s,t) \cdot K_j(s_f, t_f),\ 1 \leq i, 1 \leq j, i+j \leq fb_{max} \tag{2.12}$$

$$bK(s,t) = \sum_{i} bK_i(s,t),\ 1 \leq i \leq b_{max} \tag{2.13}$$

$$baK(s,t) = \sum_{i,j} bK_i(s,t) \cdot K_j(s_a, t_a),\ 1 \leq i, 1 \leq j, i+j \leq fa_{max} \tag{2.14}$$

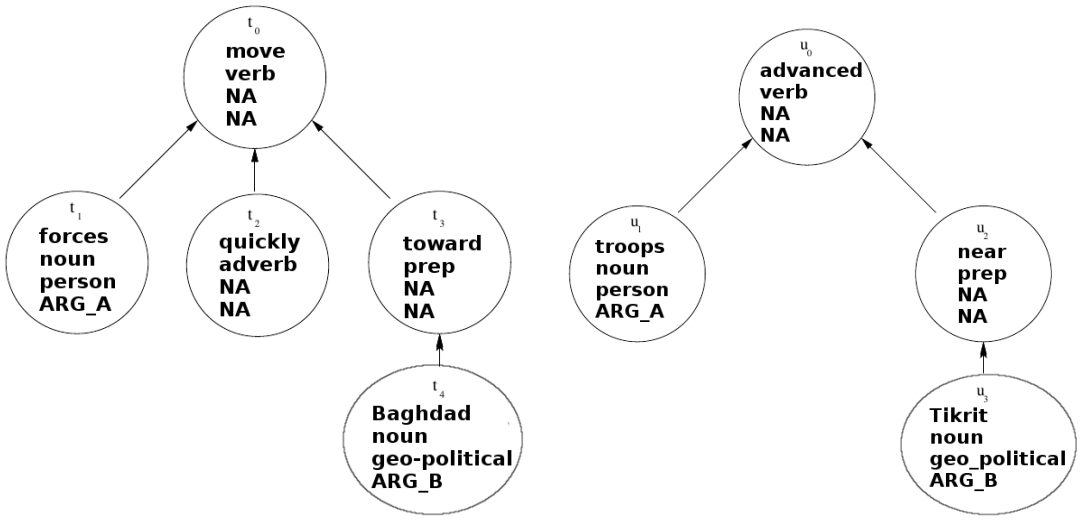
$$rK(s,t) = fbK(s,t) + bK(s,t) + baK(s,t) \tag{2.15}$$

This work was tested using the ACE corpus, the results obtained were 73.9% of Precision, 35.2% of Recall and 47.7% of F-measure.

2.4.2 Dependency Tree Kernel

Culotta and Sorensen [6] presented a kernel method based on dependency trees and a more general version of the kernel described by Zelenko et al. [30]. A dependency tree is a string structure presented previously on Section 2.1. After a process of entity extraction, a dependency tree is generated for each pair of entities. The parsing of these dependency trees is processed in a way that a pair of entities is represented by the smallest dependency tree. An example of the described structure is shown in Figure 2.5. Grammatical dependencies are represented by a node t_i having a set of children $\mathbf{c}$. The jth children of a node t_i is represented by $t_i[j]$, and $t_i[\mathbf{c}]$ represents the set of all children of a node t_i. Additionally to the tree structure, each node t_i, which corresponds to a word, also contains a set of features $\phi(t_i) = \{v_1, ..., v_d\}$. The defined set of features in the work of Culotta and Sorensen [6] includes words, part-of-speech tags, general-pos tags, chunk-tags, entity-types, entity-level tags, Wordnet hypernyms, and relation-arguments.

In order to compare dependency trees, we define two function used to compare the nodes' features. The matching function $m(t_i, t_j) \in \{0, 1\}$ described in Equation (2.16) makes use of a smaller set of features $\phi_m(t_i) \subseteq \phi(t_i)$. This function prunes the search space of matching subtrees, by finding pairs of nodes that are totally distinct. The other function, $s(t_i, t_j) \in [0, \infty]$, is described in Equation (2.17) and gives the similarity score of two nodes t_i and t_j. It uses a different set of features from the matching function $\phi_s(t_i) \subseteq \phi(t_i)$. For each feature value in common for both nodes, the similarity function adds 1 to the score. The function $C(v_q, v_r)$ given in Equation (2.18) is used by the similarity function, and returns 1 if both features have the same value, or 0 otherwise.



(a) Dependency tree for the expression *forces moved quickly towards Baghdad*.

(b) Dependency tree for the expression *troops advanced near Tikrit*.

Figure 2.5: Examples of two dependency structures for a feature vector including the set of features {general-pos, entity-type, relation-argument} for each node.

$$m(t_i, t_j) = \begin{cases} 1 & \text{if } \phi_m(t_i) = \phi_m(t_j) \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

$$s(t_i, t_j) = \sum_{v_q \in \phi_s(t_i)} \sum_{v_r \in \phi_s(t_j)} C(v_q, v_r) \tag{2.17}$$

$$C(v_q, v_r) = \begin{cases} 1 & \text{if } v_q = v_r \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

To compare two trees, we have to define how we calculate the score over the node's children. The authors presented two variations for the kernel. The simplest one is a contiguous kernel, where the child order in the structure is preserved, and the words are continuously selected. The other kernel is a sparse kernel that admits sets of non continuous children. Let us define **a** as a sequence of indices $a_1 < a_2 < \ldots < a_n$, and $d(\mathbf{a}) = a_n – a_1 + 1$ as being the distance between the indexes in the sequence of children $\mathbf{a}$. Let $l(\mathbf{a})$ be the length of the sequence of children. **b** holds the same proprieties of **a**. The value $0 < \lambda < 1$ is a decay factor that penalizes matching sequences, that are spread out within the children sequences. The kernel is then defined by Equation (2.19), calling a function that iterates over children. This function varies if we are using a contiguous kernel (2.21) or a sparse kernel (2.20). The function in Equation (2.22) defines how to evaluate a set of indicies for a node, by computing the sum of the individual scores.

$$K(t_i, t_j) = \begin{cases} 0 & \text{if } m(t_i, t_j) = 0 \\ s(t_i, t_j) + K_c(t_i[\mathbf{c}], t_j[\mathbf{c}]) & \text{otherwise} \end{cases} \tag{2.19}$$

$$K_c(t_i[\mathbf{c}], t_j[\mathbf{c}]) = \sum_{\mathbf{a},\mathbf{b},l(\mathbf{a})=l(\mathbf{b})} \lambda^{d(\mathbf{a})}\lambda^{d(\mathbf{b})} K(t_i[\mathbf{a}], t_j[\mathbf{b}]) \tag{2.20}$$

$$K_c(t_i[\mathbf{c}], t_j[\mathbf{c}]) = \sum_{\mathbf{a},\mathbf{b},l(\mathbf{a})=l(\mathbf{b})} \lambda^{l(\mathbf{a})} K(t_i[\mathbf{a}], t_j[\mathbf{b}]) \tag{2.21}$$

$$K(t_i[\mathbf{a}], t_j[\mathbf{b}]) = \sum_{s=1,2\ldots l(\mathbf{a})} K(t_i[a_s], t_j[b_s]) \tag{2.22}$$

Analyzing Figure 2.5, the contiguous kernel matches the substructure $\{t_0[0], u_0[0]\}$, $\{t_0[2], u_0[1]\}$, $\{t_3[0], u_2[0]\}$, while the sparse kernel matches the additional substructure $\{t_0[0, *, 2], u_0[0, 1]\}$, where the symbol $*$ indicates an arbitrary number of non-matching nodes.

The authors evaluated both sparse and contiguous kernels as well as a bag-of-words kernel, which handles the tree as a vector of features over nodes, disregarding any structural information. The authors also tested the combinations of kernels that are represented in Table 2.4. The kernel K_0 was not tested because the authors claimed that it was burdensome computational time. The chosen dataset was the one from the ACE competition. The results for the remaining kernels are presented in Table 2.5. The authors used the feature vector $\{$general-pos, entity-type, relation-argument$\}$ for the matching function, while the similarity function used a feature vector with the remaining features.

Table 2.4: Kernels tested in the work by Culotta and Sorensen [6].

Kernel	Kernel description
K_0	sparse kernel
K_1	contiguous kernel
K_2	bag-of-words kernel
K_3	$K_0 + K_2$
K_4	$K_1 + K_2$

Table 2.5: Kernel performance comparison in the work by Culotta and Sorensen [6].

Kernel	Average Precision	Average Recall	Average F-Measure
K_1	69.6%	25.3%	36.8%
K_2	47.0%	10.0%	14.2%
K_3	68.9%	24.3%	35.5%
K_4	70.3%	26.3%	38.0%

2.4.3 Shortest Path Dependency Kernel

Bunescu and Mooney [5] presented a dependency kernel with the particularity of using the shortest path between entities. This kernel also tends to be fast when compared to other kernels that use dependencies graphs or trees.

In this kernel, the dependency path is represented as a sequence of tokens linked with arrows that indicate the orientation of each dependency. The tokens are a simple form of information so they can then be enrich with word categories called features. The authors used part-of-speech (POS) tags, general POS tags and the entity type for the entities tokens.

For two dependency paths x and y, the kernel computation is obtained by multiplying every number of common features between two words in the same position. Formally, if $x = x_1, x_2...x_m$ and $y = y_1, y_2, ..., y$ are two relation examples, where x_i denotes the set of word classes corresponding to position i, then the kernel is computed as in Equation (2.23).

$$K(x,y) = \begin{cases} 0, & m \neq n \\ \prod_{i=1}^{n} c(x_i, y_i), & m = n \end{cases} \tag{2.23}$$

where $c(x_i, y_i)$ is the number of common word classes between x_i and y_i.

As an examples, let us consider two phrases:

1. *his actions in Brcko* , and
2. *his arrival in Beijing*.

Their corresponding dependency path are:

1. his $\rightarrow$ actions $\leftarrow$ in $\leftarrow$ Brcko , and
2. his $\rightarrow$ arrival $\leftarrow$ in $\leftarrow$ Beijing.

Their representation as a sequence of set of word classes is given by:

1. $x = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$, where $x_1 =$ { his, PPP, PERSON }, $x_2 = \{\rightarrow\}$, $x_3 =$ { actions, NNS, Noun }, $x_4 = \{\leftarrow\}$, $x_5 = \{$in, IN$\}$, $x_6 = \{\leftarrow\}$, $x_7 =$ { Brcko, NNP, Noun, LOCATION }
2. $x = [y_1, y_2, y_3, y_4, y_5, y_6, y_7]$, where $y_1 =$ { his, PRP, PERSON }, $y_2 = \{\rightarrow\}$, $y_3 =$ { arraival, NN, Noun }, $y_4 = \{\leftarrow\}$, $y_5 = \{$in, IN$\}$, $y_6 = \{\leftarrow\}$, $y_7 =$ { Beijing, NNP, Noun, LOCATION }

The kernel computation of x and y is given from Equation (2.23) as $K(x,y) = 3 \times 1 \times 1 \times 1 \times 2 \times 1 \times 3 = 18$.

The authors tested this kernel using the ACE corpus. The kernel performed 71.1% of Precision, 39.2% of Recall and 50.4% of F-measure.

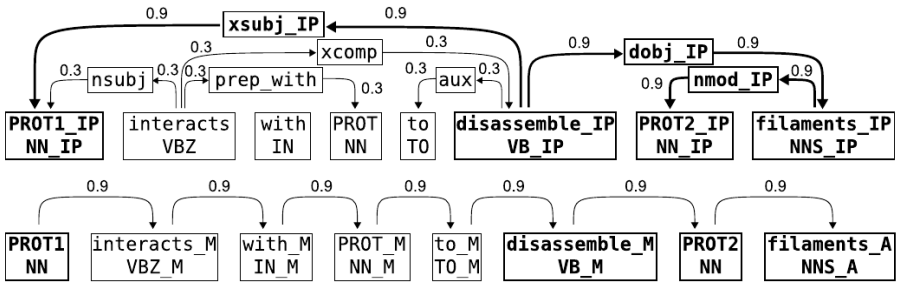


Figure 2.6: Graph representation for the sentence *PROT1 interacts with PROT to disamble PROT2 filaments*.

2.4.4 Kernels Using Multiple Structures

Airola et al. [1] presented a kernel-based method based on a mixture of two structures, namely a linear structure of a sentence, and the sentence dependency tree. These two structures will be converted into a single graph, representing an interaction candidate pair. Each structure is transformed into a non-connected subgraph that will be part of the main graph. Additionally, the authors defined this kernel for Protein-Protein interaction extraction. Therefore, to maximize the generality, the names of proteins are replaced by PROT1 for the first entity, PROT2 for the second, and PROT for any protein name that appears in the text and does not make part of the interaction candidate pair. The final graph is illustrated in Figure 2.6, and the subgraphs are described in the next two paragraphs.

The first subgraph (bottom of Figure 2.6) is built from the linear structure of the sentence. It defines that, for each token, a vertex is created and the labels for the vertices are derived from the textual tokens, POS tags, and named entity tags. In addition to that, an extra label denotes whether the word appears before (B), between (M) or after (A) the protein pair of interest. Each token is connected to its succeeding token through an edge of weight 0.9.

The second subgraph (top of Figure 2.6) is built from the dependency tree structure. One node is created for each token, and for each dependency. Token nodes are labeled with the word's POS-tag, while the dependency nodes are labeled with the type of dependency. Defining a short path between the two entities of the candidate interaction pair, the nodes and edges that make part of this path have an extra label with *IP*. Moreover, edges over this path have a weight 0.9, while the others have a weight of 0.3.

Let V be the set of vertices in the graph and $\pounds$ the set of possible labels a vertice can have. The graph is represented by an adjacency matrix $A \in \Re^{|V|\times|V|}$, where $[A]_{i,j}$ contains the weight of the edge connecting $v_i \in V$ and $v_j \in V$. The non-existing edges are represented by 0. The labels are represented in a label allocation matrix $L \in \Re^{|\pounds|\times|V|}$, so that $L_{i,j} = 1$ if the j-th vertex has the i-th label, and $L_{i,j} = 0$ otherwise.

Now, if we multiply the adjacency matrix by itself, each element $[A^2]_{i,j}$ contains the summed weight paths from vertex v_i to v_j through an arbitrary vertex v_k. The same applies if we want to obtain the summed

path from a vertex v_i to v_j through a fixed number n of vertexes, or the summed path of length n by calculating $[A^n]_{i,j}$. Having this knowledge, we can sum the powers of the adjacency matrix and obtain the summed weights of every possible path $W = \sum_{k=1}^{\infty} A^k$. This can easily be reduced by calculating the Neumann Series as in Equation (2.24), obtaining $W = (I - A)^{-1} - I$. A candidate interaction G is then defined as $G = LWL^T$. Considering two instances G' and G'', the final graph kernel is defined as shown in Equation (2.25).

$$\sum_{k=0}^{\infty} A^k = I + A + A^2 + ... = (I - A)^{-1} \tag{2.24}$$

$$K(G', G'') = \sum_{i=1}^{|\pounds|} \sum_{j=1}^{|\pounds|} G'_{i,j} G''_{i,j} \tag{2.25}$$

This work was tested with a protein-protein interaction dataset called AIMed . The results of the evaluation were 52.9% of Recall, 61.8% of Precision and 56.4% of F-Measure.

2.4.5 Bag-Of-N-Grams Kernel

Giuliano et al. [9] presented a kernel that combines both global and local context. The global context considers the whole sentence, while the local context takes into account the tokens around the entities.

The two main kernels, global and local can be subdivided in three and two kernels, respectively. Each sub-kernel is calculated using the 2-norm given in Equation (2.26), commonly used in other kernels in the literature.

$$K(s, t) = \frac{< \phi(s), \phi(t) >}{||\phi(s)||\ ||\phi(t)||} \tag{2.26}$$

Similar to the subsequence kernel previously presented in Section 2.4.1, the global context kernel is the sum of three kernels. These kernels are similar but they are applied over different set of tokens. The pair of entities defines the borders, If the tokens are between, after and between, or before and between both entities.

The global context kernel is distinct from the subsequences kernel by using a bag-of-words instead of subsequences. A bag-of-words considers only the frequency of a token in a specific sequence or subsequence but it does not take into account its position. In this kernel, we have 3 bags-of-words one for each setup.

Formally, each bag-of-words kernel can be represented in Equation (2.27) and the final global kernel obtained by replacing them in Equation (2.26).

$$\phi_P(R) = (tf(t_1, P), tf(t_2, P), ..., tf(t_l, P)) \in \Re^l \tag{2.27}$$

In the formula, the function $tf(t_i, P)$ returns the frequency with which a token t_i is used in P.

Moreover, the authors found that it was useful to use the type of the interacting entities. Therefore in an attempt to find the type of an entity, the tokens around each entity are used to classify each entity. Two more kernels were developed, one for each entity, called the left local context and the right local context. Each local context uses a window around the entity and can also make use of some features. For instance, the authors use the token, the lemma of the token, part of speech tag of the token and orthographic features.

Given a sample R and a local context $L = t_{-w}, ...t_{-1}, t_0, t_{+1}, ..., t_{+w}$, a local context kernel can be defined as a vector by Equation (2.28).

$$\phi_L(R) = (f_1(L), f_2(L), ..., f_m(L)) \in \{0, 1\}^m \tag{2.28}$$

In the formula, f_i is a feature function that returns 1 if the feature exists or 0 if it does not. Similar to the global context kernel, the local kernel is then computed by using the two vectors of each entity in Equation (2.26).

This work was tested separately in three kernels, Global Context kernel, Local Context kernel and the sum of both Global and Local Context. with the AIMed dataset. Table 2.6 represents the reported results for the experiments with AImed. As expected the Global + Local Context Kernel obtained the best performance.

Table 2.6: Kernel performance comparison in the work by Giuliano et al. [9].

Kernel	Average Precision	Average Recall	Average F-Measure
Global Context kernel	57.7%	60.1%	58.9%
Local Context kernel	37.3%	56.3%	44.9%
Global + Local Context kernel	60.9%	57.2%	59.0%

2.5 Summary and Critical Discussion

In this chapter, we analyzed the typical pipeline, from pre-processing solutions to entity extraction. The pre-processing steps contained sentence segmentation, tokenization, part of speech tagging, tree parsing and dependency parsing. Regarding the relationship extraction problem, we have analyzed machine learning solutions, in particular the support vector machines that weight the data instances, in order to produce a classification model. These solutions can use either a simple set of features, namely feature-based methods, or complex structures (i.e., a tree or a graph), called kernel-based methods.

A short summary description of the ML solutions for relationship extraction, presented in Section 2.3 and in Section 2.4, can be found in Table 2.7. Note that the authors of these methods used different datasets, and thus, the results cannot be used directly to compare different approaches.

One of the main purposes of an analysis over the related work is to identify the best solutions for our problem. Based on this analysis, we selected three kernel-based methods to integrate the Benchmark. The methods are the subsequences kernel in Section 2.4.1, the shortest path dependency kernel in Section 2.4.3 and the bag-of-n-grams kernel in Section 2.4.5. Although these kernels were tested and evaluated, the tests were not performed using the same dataset or using the same pre-processing tools. Additionally some of these kernels are composed by linear combinations of sub-kernels. We believe that better performances can be achieved by automatically combine these sub-kernels with weights. Therefore, in Chapter 3 we cover an implementation of a multiple kernel learning algorithm for SVMs and the corresponding analysis in the Benchmark.

Table 2.7: Overview on the surveyed techniques.

Techniques	Characteristics / Kernels	Features used	Datasets	Best reported F-measure
SIFT [21]	Statistical model based on Bayes theorem	Structural and Context features	MUC-7	71.23%
Kambhatla [15]	Maximum Entropy models	Lexical, Syntactic and Semantic features	ACE 2002 and 2003	55.2%
Jiang and Zhai [13]	Maximum Entropy models and Support Vectors Machine	Sequence, Dependency trees and Parse trees	ACE 2004	68.8%
Subsequence Kernel [4]	Fore-Between, Between and Between-After kernels	Sequence and Lexical features	Almed and ACE 2002	47.7%
Dependency Tree Kernel [6]	Dependency tree	Dependency parse	ACE 2000	38.0%
Shortest Path Dependency Tree Kernel [5]	Dependency Shortest Path	Dependency parse	ACE 2000	50.5%
Airola et al. [1]	Sentence and Dependency	Parse tree and sequence structures	Almed	56.4%
Bag-Of-N-Grams Kernel [9]	Global Context kernel (Fore-Between, Between and Between-After kernels) and Local Context Kernel (left entity and right entity windows kernels)	Ngrams and Lexical	Almed	59.0%

Chapter 3

Relationship Extraction with Multiple Kernel Learning

The relationship extraction problem can be solved using different methods. The focus of this thesis are on supervised machine learning methods. We can extract relationship using a machine learning technique that classifies a candidate pair of entities with a relationship type.

In Section 2.2, we presented a machine learning classifier, named Support Vector Machines, that weights the samples from the training data. In its basic form this classifier only works with binary problems. Therefore, for the cases where multiple relationship types are contained in a corpora, it is necessary to reduce the relationship extraction problem to a binary classification process. Two solutions for this problem were previously presented in Section 2.2. The one we implemented was to assign a SVM classifier for each pair of classes (the set of classes include all relationship types plus a non-relationship class). Then, classification is performed by a max-wins voting strategy. When predicting a relationship, every classifier assigns the candidate sentence to one of the two classes, then the vote for the assigned class is increased by one. Finally, the class with the most votes determines the candidate pair of entities.

A characteristic of SVMs is that they can use kernels which are comparison functions between two candidate instances. In relationship extraction, a kernel allow us to introduce complex structures as input. Several methods for relationship extraction from the current state-of-the-art make use of kernels. Our first contribution in Section 3.1 is the implementation of an online learning approach, and then, developed to use a kernel.

In Section 3.2 we present our second contribution which includes the development and implementation of a multiple kernel learning approach. It is based on the first contribution because both use an online approach and train SVMs, but while the first contribution accepts only one kernel, our multiple kernel

learning approach accepts and trains weights for a set of kernels.

The third contribution is in Section 3.3 where we detail the benchmark used to compare different kernels from the state-of-the-art present in Section 2.4. It also describes the datasets, the NLP pre-processing, and the evaluation procedure.

3.1 Online Learning of Support Vectors Machines

In order to solve the optimization problem associated to the training of SVMs, as presented in Section 2.2.1, different approaches have been considered in the past. Some of them are associated to the training of online learning algorithms, which are iterative algorithms that solve the SVM problem through the successive analysis of new training instances. An advantage of these online approaches is that the model can be updated with new data. One method in this category is based on gradient descent and is known by the name of Pegasos [27]. Pegasos is a fast solver that is known to perform well when compared to other solutions. Therefore, we propose an approach based on Pegasos for solving the optimization problem using kernels. Although originally this method was proposed to solve a SVM problem with features and without considering a bias term, the authors suggested extensions in which a bias and kernels can be added to the Pegasos algorithm. In this section, we will first explain the base algorithm, and then the extensions that have led to the final algorithm used in this work.

The task of learning a support vector machine online is an unconstrained empirical loss minimization problem, with a penalty term for the norm of the classifier that is being learned. Formally, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ of size m, where $\mathbf{x}_i \in \Re^n$ and $y_i \in \{+1, -1\}$, we want to solve the minimization problem, as in Equation (3.29), in order to obtain the weighting vector $\mathbf{w}$ that best describes the data.

$$\min_{\mathbf{w}} \left\{ \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m} \sum_{(\mathbf{x},y)\in S} l(\mathbf{w}; (\mathbf{x}, y)) \right\} \tag{3.29}$$

The parameter λ corresponds to the weight or importance given to the L2-norm $||\mathbf{w}||^2$ in the objective function, against the loss function of every $(\mathbf{x}, y) \in S$, defined as being $l(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$. In the literature, instead of a parameter λ, some support vector solvers use a parameter c associated to weighting the loss function. Note that the process of optimizing $\mathbf{w}$ using a L2-norm is called a L2 regularization, usually applied in machine learning to prevent overfitting.

Pegasos was originally designed to work with a mixture of descendant gradients and projections, using a variable k that corresponds to the number of examples that are used to calculate the sub-gradient. Two extremes of the algorithm correspond to setting k equal to the total number of samples, becoming only a projection based approach, or setting $k = 1$, this way behaving as a stochastic gradient method. In our

implementation, we used $k = 1$. The code of Pegasos simplified for $k = 1$ can be found in Algorithm 1.

input : S, λ, T

initialize : Chose $\mathbf{w}_1$ s.t. $||\mathbf{w}|| \leq 1/\sqrt{\lambda}$;

for $t = 1, 2, ..., T$ **do**

  Choose $A_t \subset S$;

  Set $\eta_t = \frac{1}{\lambda t}$;

  if $\{(\mathbf{x}, y) \in A_t : y\langle \mathbf{w}_t, \mathbf{x}\rangle < 1\}$ **then**

    Set $\mathbf{w}_{t+\frac{1}{2}} = (1 - \eta_t\lambda)\mathbf{w}_t + \eta_t y\mathbf{x}$, $(\mathbf{x}, y) \in A_t$;

  else

    Set $\mathbf{w}_{t+\frac{1}{2}} = (1 - \eta_t\lambda)\mathbf{w}_t$;

  end

  Set $\mathbf{w}_{t+1} = min\left\{1, \frac{1/\sqrt{\lambda}}{||\mathbf{w}_{t+\frac{1}{2}}||}\right\}\mathbf{w}_{t+\frac{1}{2}}$

end

output: $\mathbf{w}_{T+1}$

Algorithm 1: Simplified version of the Pegasos algorithm.

Besides the parameter k, the algorithm allows another parameter to be given as input, namely a parameter T that corresponds to the maximum number of iterations to perform. Since the algorithm is iterative it can iterate over the same samples, the T parameter is also the number of times the samples are iterated.

To understand the algorithm, based on the minimization problem in Equation (3.29), we define the objective function that we want to minimize in Equation (3.30), where A_t represents a subset of samples with size k.

$$f(\mathbf{w}; A_t) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{k}\sum_{(\mathbf{x},y)\in A_t} l(\mathbf{w}; (\mathbf{x}, y)) \tag{3.30}$$

As we only consider $k = 1$, the set A_t will always consist of only one sample. Therefore, the sum can be omitted, as in Equation (3.31):

$$f(\mathbf{w}; (\mathbf{x}, y)) = \frac{\lambda}{2}||\mathbf{w}||^2 + l(\mathbf{w}; (\mathbf{x}, y)) \tag{3.31}$$

The main part of a SVM algorithm is to minimize the objective function resorting to optimization theory methods. Pegasos is based on a stochastic gradient descent approach. A stochastic gradient descendent defines a learning rate η_t depending on the current iteration, in our case given by $\eta_t = 1/(\lambda t)$. Intuitively, the samples are of greater importance at the beginning of the execution. Since we want to reach convergence in the objective function, after each sample, the new information continuously looses importance. Therefore, the stochastic gradient step can be written as $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla_{w_t} f(\mathbf{w}; (\mathbf{x}, y))$

where $\nabla_{w_t} f(\mathbf{w}; (\mathbf{x}, y))$ is the sub-gradient of $f(\mathbf{w}; (\mathbf{x}, y))$ at $\mathbf{w}_t$, as in Equation (3.32).

$$\nabla_{w_t} f(\mathbf{w}; (\mathbf{x}, y)) = \lambda \mathbf{w}_t - y\mathbf{x}, \quad (\mathbf{x}, y) \in A_t \tag{3.32}$$

In other words, the main objective is to find the optimal vector $\mathbf{w}_t$. After each iteration, $\mathbf{w}_t$ is scaled by $(1 - \eta_t \lambda)$. Moreover, if the sample A_t was not in the right margin of the support vectors, then the vector $y\eta_t x$ is added to $\mathbf{w}$, resulting in a new vector $\mathbf{w}_{t+\frac{1}{2}}$. In case A_t was in the right margin there is no correction to be made, and so $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}$.

After the gradient step, the algorithm applies a projection of $\mathbf{w}_{t+\frac{1}{2}}$ in order to agree with a constrain corresponding to $||\mathbf{w}|| \leq 1/\sqrt{\lambda}$ resulting in the final $\mathbf{w}_{t+1}$ of the iteration.

The algorithm will iterate over the samples until all the samples have been evaluated according to the objective function. The procedure repeats itself T times, or until the algorithm converges.

Considering only the simple version of Pegasos, there are two important limitations in what concerns the specific problem considered in this work. The first limitation is not being defined in Pegasos algorithm a kernel approach of the SVMs, which would enable us to use a kernel-based method instead of a features set. The second limitation is the lack of a bias term b that would augment the weight vector $\mathbf{w}$, the bias term is important when training over unbalanced data. In order to solve these limitations, two extensions were applied to Pegasos.

The first extension uses a bias term b. It means when predicting the result for an instance x, we will instead use $\langle \mathbf{w}, \mathbf{x} \rangle + b$. The new lost function is then defined as in Equation (3.33)

$$l((\mathbf{w}, b); (\mathbf{x}, y)) = max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)\} \tag{3.33}$$

To solve the bias limitation, we directly use Equation (3.33), while not penalizing for b. The new minimization problem is defined in Equation (3.34).

$$\min_{\mathbf{w}, b} \frac{\lambda}{2} ||\mathbf{w}||^2 + \sum_{(\mathbf{x}, y) \in S} [1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)] \tag{3.34}$$

The second extension relates to the Pegasos limitation to use kernels. Typical SVM solvers use a dual problem to address the issue of model training when considering kernels, but the Pegasos algorithm minimizes the primal problem. However, it is possible to incorporate kernels within Pegasos. Considering that for all appearances of $\mathbf{w}_t$, it can be written as $\mathbf{w}_t = \sum_{i \in I_t} \alpha_i \mathbf{x}_i$, where I_t is a subset of $\{1, ..., m\}$ and the coefficients α_i are the weights of the corresponding support vectors I_i. The transformation includes changing the inner product operations, using $\langle \mathbf{w}_t, \mathbf{x}_t \rangle = \sum_{i \in I_t} \alpha_i \langle \mathbf{x}_i, \mathbf{x}_t \rangle$, and evaluating the norm of $\mathbf{w}_t$ using $||\mathbf{w}_t||^2 = \sum_{i,j \in I_t} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. The final algorithm, with both extensions, is shown in

the pseudocode for Algorithm 2.

input : S, λ, T
initialize : Chose $\alpha_1 = 0$;
for $t = 1, 2, ..., T$ **do**
  Choose $A_t \subset S$;
  Set $\eta_t = \frac{1}{\lambda t}$;
  for $i \subset t$ **do**
    Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda)\alpha_i$;
  end
  if $\{(x, y) \in A_t : y \sum_{i \in I_t} \alpha_i K(I_i, x) + b < 1\}$ **then**
    Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda)\alpha_i + \eta_t y$, $(x, y)_i \in A_t$;
  end
  for $l \subset t$ **do**
    Set $\alpha_{l+1} = min\{1, \frac{1/\sqrt{\lambda}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)}\}\alpha_{l+\frac{1}{2}}$;
  end
  Set $b_{t+1} = \frac{1/\sqrt{\lambda}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)} b_{t+\frac{1}{2}}$;
end
output: α_{T+1}, b_{T+1}

Algorithm 2: Pegasos algorithm extended to use kernel and bias term.

3.2 Multiple Kernel Learning for Relationship Extraction

We used different kinds of kernels in this work. Some of them actually correspond to a mixture of simpler kernels. We developed and then evaluated an approach based on multiple kernel learning, in which the idea is to automatically learn the weights associated to a linear combination of multiple kernels. Therefore, we learn weights for the kernels, and learn at the same time the weights of the support vectors.

More formally, we used a procedure that finds an optimal set of kernel weights given by β, where a final kernel value is obtained through $K(x_1, x_2) = \beta_0 K_0(x_1, x_2) + \beta_1 K_1(x_1, x_2) + ... + \beta_N K_N(x_1, x_2)$. The scalar N is the total number of kernels. We can now define a new minimization problem as in Equation (3.35).

$$\min_{\alpha,\beta,b} \left\{ \frac{\lambda_\alpha}{2} ||\alpha||_2^2 + \lambda_\beta ||\beta||_1 + \frac{1}{m} \sum_{(x,y) \in S} l(\alpha, \beta, b; (x, y)) \right\} \tag{3.35}$$

Introducing these new ideas into the method explained in Section 3.1, we can formulate a new objective function as shown in Equation (3.36), adding the minimization of the square L1-norm associated to

vector β as being $||\beta||_1$.

$$f(\alpha, \beta, b; (\mathbf{x}, y) \in A_t) = \frac{\lambda_\alpha}{2}||\alpha||_2^2 + \lambda_\beta||\beta||_1 + l(\alpha, \beta, b; (x, y)) \tag{3.36}$$

A new loss function is also formulated as $l(\alpha, \beta, b; (x, y)) = \max\{0, 1 - y[\sum_{i \in I_t} \alpha_i \sum_{n=0}^{N} \beta_N K_n(x, I_i) + b]\}$.

While the α vector uses a L2 regularization procedure, we propose to use a L1 regularization on the vector β, this way encouraging models that use fewer kernels. A L1 regularization is the optimization procedure that uses a L1-norm, it is computed through the sum of the absolute values in a vector, in this case the β vector. A stochastic gradient descent step of β can be added to the algorithm. This step is written as in Equation (3.38), where the sub-gradient of $f(\alpha, \beta, b; (x, y))$ at β_{tn} is in Equation (3.37).

$$\nabla_{\beta_{tn}} f(\alpha, \beta, b; (x, y)) = y \sum_{i \in I_t} \alpha_i K_n(x, I_i) \tag{3.37}$$

$$\beta_{tn} = (1 - \eta_t \lambda_\beta)\beta_{tn} - \eta_t \nabla_{\beta_{tn}} f(\alpha, \beta, b; (x, y)), \quad n \in N \tag{3.38}$$

In a L1 regularization there are two usual problems that we did not have with a L2 regularization. The first one is the difficulty to get a value that is exactly equal to zero, in our case preventing the model from discarding useless kernels. The other problem is that the weights are sensible to the data which means, if a weight at a specific time can get a value different from zero, then it will not be discarded even if in the past it had the value zero for most of the time.

Tsuruoka et al. [29] proposed a method to overcome these two problems in solving a L1 regularization. The authors solved this problem for a L1 regularization of a feature vector, instead we apply it in β vector. The first solution presented solved the problem of the weights hardly getting a zero value. The solution simply demanded a weight to become zero, whenever its value change polarity. To solve the second problem, the authors suggest a cummulative penalty q_{ti}, that takes into account the past weight values. Note that in our solution, the elements of the vector β can not have values that are less than zero.

Defining q_{tn} as being the total L1 penalty that β_n has actually received up to the point $q_{tn} = \sum_{k=1}^{t}(\beta_{k+1n} - \beta_{k+\frac{1}{2}n})$, and u_t as being the absolute value of the total L1 penalty that each weight β could have received up to the point. $u_t = \lambda_\beta \sum_{k=1}^{t} \eta_k$. The final solution for the L1 regularization steps are given by Equation (3.39) and Equation (3.40), which substitute the regularization step defined before in Equation (3.38).

$$\beta_{t+\frac{1}{2}n} = \beta_{tn} + \eta_t \nabla_{\beta_{tn}} f(\alpha, \beta, b; (x, y)), \quad n \in N \tag{3.39}$$

$$\beta_{t+1n} = \max(0, \beta_{t+\frac{1}{2}n} - (u_t + q_{t-1n})), \quad n \in N \tag{3.40}$$

The final pseudocode is presented in Algorithm 3 .

input : S, λ, T

initialize : Choose $\alpha_1 = 0$ and $\beta_1 i = \frac{1}{\sqrt{\lambda_\beta * |\beta|}}$, $i \in t = 1$;

for $t = 1, 2, ..., T$ **do**

  Choose $A_t \subset S$;

  Set $\eta_t = \frac{1}{\lambda_\alpha t}$;

  for $i \subset t$ **do**

    Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda_\alpha)\alpha_i$;

  end

  if $\{(x, y) \in A_t : y \sum_{i \in I_t} \alpha_i K(I_i, x) + b < 1\}$ **then**

    Set $\alpha_{i+\frac{1}{2}} = (1 - \eta_t \lambda_\alpha)\alpha_i + \eta_t y$, $(x, y) \in A_t$;

    for $n \subset N$ **do**

      Set $\beta_{t+\frac{1}{2}n} = \beta_{tn} + \eta_t y \sum_{j \in I_t} \alpha_{tj} K_n(I_j, x)$, $(x, y) \in A_t$;

    end

  end

  for $n \subset N$ **do**

    Set $\beta_{t+1n} = \max(0, \beta_{t+\frac{1}{2}n} - (u_t + q_{t-1n}))$;

    Set $q_{tn} = q_{t-1n} + (\beta_{t+1n} - \beta_{t+\frac{1}{2}n})$;

  end

  for $l \subset t$ **do**

    Set $\alpha_{l+1} = min\left\{1, \frac{1/\sqrt{\lambda}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)}\right\} \alpha_{l+\frac{1}{2}}$;

  end

  Set $b_{t+1} = \frac{1/\sqrt{\lambda_\alpha}}{\sum_{i,j \in I_t} \alpha_i \alpha_j K(I_i, I_j)} b_{t+\frac{1}{2}}$;

  for $n \subset N$ **do**

    Set $\beta_{t+1n} = min\left\{1, \frac{1/\sqrt{\lambda_\beta}}{\sum_m^N \beta_m}\right\} \beta_{t+\frac{1}{2}n}$;

  end

end

output: α_{T+1}, b_{T+1}

Algorithm 3: A Multiple Kernel Online Learner.

Although calculating a norm is a heavy computation, we can calculate a norm in a specific time t using the norm from the previous $t-1$ time. For optimization purposes, the next paragraphs demonstrate how to obtain the norm through this method. The computation of $||\beta||$ is not expensive because the number of kernels is much smaller than the number of support vectors, so we focus only in solving $||w_{t+1}||$ using the previous value $||w_t||$.

$$
\begin{aligned}
||\mathbf{w}_t||^2 &= \sum_{i,j \in I_t} \alpha_{it}\alpha_{jt}K(\mathbf{x}_i, \mathbf{x}_j) \\
&= \sum_{i,j \in I_t} \alpha_i\alpha_j \sum_{n \in N} \beta_{nt}K_n(\mathbf{x}_i, \mathbf{x}_j) \\
&= \sum_{n \in N} \beta_{nt} \sum_{i,j \in I_t} \alpha_i\alpha_j K_n(\mathbf{x}_i, \mathbf{x}_j) \\
&= \sum_{n \in N} \beta_{nt}||\mathbf{w}_{nt}||^2
\end{aligned}
\tag{3.41}
$$

From Equation 3.41 we need to find the $||\mathbf{w}_{nt}||^2$ for each $n \in N$.

$$
\begin{aligned}
||\mathbf{w}_{nt}||^2 &= \sum_{i,j \in I_t} \alpha_i\alpha_j K_n(\mathbf{x}_i, \mathbf{x}_j) \\
&= \sum_{i,j \in I_{t-1}} \alpha_i\alpha_j K_n(\mathbf{x}_i, \mathbf{x}_j) + 2\eta_t y \sum_{i \in I_{t-1}} K_n(I_i, x), (x, y) \in A_t \\
&= ||\mathbf{w}_{nt-1}||^2 + 2\eta_t y \sum_{i \in I_{t-1}} K_n(I_i, x), (x, y) \in A_t
\end{aligned}
\tag{3.42}
$$

Using Equation 3.42 we can observe that simply with an accumulator $\mathbf{w}_{nt}$ per kernel $n \in N$, we can easily maintain and calculate the norm using the previous norms. Moreover, we can store the calculation of $\sum_{i \in I_{t-1}} K_n(I_i, x), (x, y) \in A_t$ because it is already being computed in line 11 of Algorithm 3.

3.3 A Benchmark for Relationship Extraction Methods

Analogously to the benchmark for NER systems presented in Marrero et al. [18], this section describes the process involved in developing a benchmark for relationship extraction methods. This process can be separated of sub-processes as illustrated in Figure 3.7.

The first sub-process converts the datasets into a common format clearly identifying the raw text, the annotated entities and the annotated references to the relationships existing in the text. The chosen datasets are described in Section 3.3.1.

The second sub-process takes care of the raw text contained inside the corpus. The raw text is unstructured and so, a pre-processing step is necessary to structure it. The pre-processing uses state-of-the-art techniques incorporated in existing open-source Natural Language processing tools. These tools and systems are described in Section 3.3.2. The structures available, (POS tagged text, dependency graphs or parse trees) are then used by the different relationship extraction methods.

The third sub-process uses different relationship extraction methods form the state-of-the-art. These

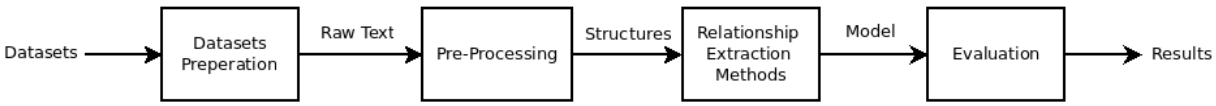


Figure 3.7: Benchmark process decomposition.

methods are machine learning techniques which make use of the pre-processed structures and annotated relationships to train a model. The trained model is then used to predict new instances of relationships given pairs of entities.

The final sub-process is the evaluation of the implemented techniques with metrics described in Section 3.3.4. The models trained by the techniques, first predicts the relationship type of pairs of entities. After, we compare the predicted relationship type with the real relationship type and utilize this information to compute the measures. The dataset using only one relationship (i.e. AImed) is evaluated with the standard measures precision, recall and F1-measure, while the dataset that has multiple relationship types (i.e. SemEval) is evaluated with the micro and macro averages measures.

3.3.1 Corpora Available for Validation Experiments

Since the purpose of this MSc thesis is to evaluate relationship extraction systems, we needed appropriate corpora and evaluation metrics, to evaluate the performance of the relationship extraction methods. This section describes two corpora used that were selected from available datasets from conferences and competitions. These corpora are in the English language. Table 3.8 presents a statistical characterization of the datasets.

AImed: AImed is a protein-protein interaction dataset. It is a single relationship dataset, which means it covers only the problem of predicting whether there is a relationship, from a candidate pair of entities. This dataset is composed by 255 Medline abstracts from which 200 describe interactions between proteins and the other 25 do not refer any interaction. The total number of interacting pairs is 996 and the total number of non-interacting pairs is 4475. An example sentence from this dataset is *The main p47nck region implicated in the association with p66WASP was found to be the carboxy - terminal SH3 domain .* where the entities are *p47nck* and *p66WASP*, these entities form a candidate pair that contains a relationship.

SemEval 2010 task 8: **Sem**antic **Eval**uation [11] is an ongoing series of evaluations of computational semantic analysis systems. This competiton in 2010 elaborated a dataset for what they called the task 8 which consisted of multi-way classification of semantic relations between pairs of nominals. This dataset includes a training set with 8000 sentences and a test set with 2717 sentences. Each sentence is independently identified and represent only one candidate pair. It is a multiple relationship dataset including 9 asymmetric relationships types. The relationship types

are Member-Collection, Cause-Effect, Component-Whole, Instrument-Agency, Entity-Destination, Product-Producer, Message-Topic, Entity-Origin and Content-Container. An example sentence from this dataset is *The most common audits were about waste and recycling*, where a relationship f type Message-Topic is given by the entity pair *audits* and *waste*.

Table 3.8: Statistical characterization of the considered corpora.

Corpora	Sentences	Candidate Pairs of Entities	Entities	Relationships	Relationships Types
Almed	1159	5471	3754	996	1
SemEval	10717	10717	21434	8853	9

3.3.2 Linguistic Pre-processing tools

Before we work on the relationship extraction module, it is important to use the same remaining modules for every solution made part of this Benchmark. Following the modules described before, the tools or techniques applied for each of them are as follows:

Sentence Segmentation: the text was processed with the *Apache OpenNLP*[1] library, a machine learning based toolkit. This library used a module previously trained with English corpora, for performing sentence segmentation. This pre-processing is not need in SemEval because it already includes sentence segmentation.

Tokenization: the sentences were processed with the same library (i.e., openNLP). The difference is that we used a trained module with English corpora for tokenization.

Part of Speech Tagging: Part of Speeches tags were also tagged with *Apache OpenNLP* library. It used a maxent model with tag dictionary, also trained with English corpora

Dependency Parsing: *Stanford CoreNLP*[2] library was used for dependency parsing. It uses a statistical based parser to express dependencies among tokens.

Some kernels form the literature require extra linguistic information. This information is used to enrich the token with word classes. An advantage is instead of comparing only tokens, we can compute a similarity function between tokens by counting the common word classes. When applied we used the same types of word classes into the kernels. The implemented word classes types are:

Part of Speech Tags: computed during the pre-processing, this class represents the grammatical class of the word.

[1] http://opennlp.apache.org/
[2] http://nlp.stanford.edu/software/corenlp.shtml

Generic Part of Speech Tags: are generalization over the Part of Speech Tags, it has less types of classes. It is computed by mapping the POS tags to only $\{$ Adjective Noun, Verb and Other $\}$.

Stem: computed through the Porter Stemming Algorithm. This algorithm uses rules to reduce the work to his basic form.

Capitalization: reduces the words into two patterns. The first pattern reduces each char to a specific symbol if it is capital letter (A), a minuscule (a) or a number(0). The second pattern is similar to the first one but when there is one or more chars of the same type, the sequence is replaced by a symbol following a $+$.

3.3.3 Relationship Extraction Methods

The relationship extraction methods from the current state-of-the-art are based on machine learning approaches. The most indicated machine learning methods are the Support Vector Machines (SVMs), presented in Section 2.2.1. SVMs have the particularity to give good results in classification problems, and can either be used with feature-based methods or kernel-based methods.

Two SVM solvers are contained in our contribution, the simpler online learning from Section 3.1 which is used with single kernels solutions and the multiple kernel online learner Section 3.2 used with a set of kernels.

From the kernel-based methods described in Section 2.4, we selected and implemented the subsequence kernel as in Section 2.4.1, the bag-of-n-grams kernel as in Section 2.4.5 and the shortest path dependency kernel as in Section 2.4.3. These kernels are part of the benchmark and with the online learning solution from Section 3.1 they form the first three evaluated relationship extraction methods.

The next systems are based on these three kernels. As it was exposed in these kernels descriptions, both the subsequence kernel and the bag-of-n-grams kernels are respectively made of three and five sub-kernels. These kernels can be used separately with our multiple kernel learner described in Section 3.2. This way, we aim for a better performance of the system. Therefore, the benchmark includes three more systems that use the multiple kernel SVM solver. The first one, is composed by the three sub-kernels that compose the subsequence kernel, the second one similar to the first, used the five sub-kernels found in the bag-of-n-grams kernel and the last one uses all the sub-kernels from the subsequence kernel and the bag-of-n-grams kernel plus the shortest-path dependency kernel.

3.3.4 Evaluation Metrics for Relationship Extraction

In a benchmark study, it is very important to provide success measures in a way that supports the comparison of methods. This MSc thesis is related to the usage of machine learning methods in a

classification task. Typical measures to evaluate classification systems are *precision*, *recall* and the *f-measure*. Let us define r as a type of relationship. *Precision* is the fraction of correctly retrieved relationships of type r in the text, over the total number of retrieved relationships of type r. *Recall* is the fraction of relationships of type r that are correctly extracted, over the total number of relationships of type r present in the text. An attempt to combine these measures is the *F_1-Measure* , which corresponds to the harmonic mean of both Precision and Recall.

Being N the number of relationships types r, let us define:

•**True Positives (TP_r)** as the number of successfully extracted relationships of type r;

•**False Positives (FP_r)** as the number of extracted relationships that are said to be of type r but are not from type r;

•**True Negatives (TN_r)** as the number of success fully extracted relationships that were not of type r;

•**False Negatives (FN_r)** as the number of extracted relationships that are of type r but are said to be other type;

Precision, Recall and F_1-Measure can accordingly be defined as:

$$P_r = \frac{\text{Number of correctly extracted relationships } r}{\text{Total number of extracted relationships } r} = \frac{TP_r}{TP_r + FP_r} \tag{3.43}$$

$$R_r = \frac{\text{Number of correctly extracted relationships } r}{\text{Total number of relationships type } r \text{ in text}} = \frac{TP_r}{TP_r + FN_r} \tag{3.44}$$

$$F_1 r = \frac{2P_r R_r}{P_r + R_r} \tag{3.45}$$

These measures are naturally defined per class, but it can be useful to extend them in order to compose a final score in the case of multiclass classification problems. A direct extension is obtained by averaging over Precision and Recall, which is what Macro-average and Micro-average methods try to implement. *Macro-averages* given by Equations (3.46) and (3.47) take the average of the precision and recall of the system on different sets. *Micro-averages* given by Equations (3.48) and (3.49) sum up the individual true positives, false positives, and false negatives returned by the system for different classes, and the measures are computed by applying the formula of the single class measures (i.e. Precision or Recall) directly using the sums of the individuals.

$$\text{Macro-average Precision} = \frac{\sum_r^N P_r}{N} \tag{3.46}$$

$$\text{Macro-average Recall} = \frac{\sum_r^N R_r}{N} \tag{3.47}$$

$$\text{Micro-average Precision} = \frac{\sum_r^N TP_r}{\sum_r^N TP_r + \sum_r^N FP_r} \tag{3.48}$$

$$\text{Micro-average Recall} = \frac{\sum_r^N TP_r}{\sum_r^N TP_r + \sum_r^N FN_r} \tag{3.49}$$

3.4 Summary

In this chapter, we described the main contributions of this thesis. The first contribution presented in Section 3.1 is an approach based on the online learning of SVMs, we implemented an already existing algorithm named Pegasos, and then extended this algorithm to fulfill our constrains by containing a bias term and be able to use kernels.

In Section 3.2 we propose our second contribution. We developed and implemented another online learner of SVMs. This solution aims to solve a multiple kernel learning problem. A part from learning the weights for the support vectors, given a set of kernels, we want also to learn the weights associated to each kernel that will give the best performance to the trained model.

The third contribution in Section 3.3 is a benchmark for relationship extraction kernel methods. We defined, metrics, datasets and pre-processing systems to be used on all methods. This way we expect to impartially evaluate the selected methods in Section 4 and analyze their performance.

Chapter 4

Validation

In this chapter, we present the results obtained when applying the benchmark described in Section 3.3 to the relationship extraction methods described in Section 3.3.3. In Section 4.1, we present the experimental setup, in particular the parameter values used and the tested kernel-based methods. Section 4.2 holds the results of the experiments over the datasets AImed and SemEval. Finally, in Section 4.3 we summarize the Chapter.

4.1 Experimental Setup

This Section describes the experiments performed by following the Benchmark described in Section 3.3. First in Section 4.1.1, we enumerate the different kernels, or combinations of kernels, and associate them either to the online learner from Section 3.1 or to the multiple kernel learner from Section 3.2. Then, we present the fixed values for the SVM solver's parameters in Section 4.1.2. Section 4.1.3 describes the hardware and software configurations where we run our experiments.

4.1.1 Kernels Used in the Experiments

The first three experiments are performed with the online learner described in Section 3.1, while the other three are performed with the multiple kernel learner described in Section 3.2. The kernels that constitute part of the experiments and that can accept a word class configuration are using $\{$POS tags, GPOS tags, Lemma, Capitalization pattern 1 and Capitalization pattern 2$\}$. The kernel methods we experiment are:

Experiment 1: Subsequence Kernel, from Section 2.4.1.

Experiment 2: Bag-Of-N-Grams Kernel, from Section 2.4.5.

Experiment 3: Shortest Path kernel, from Section 2.4.3.

Experiment 4: Multiple kernel learning over the following subsequences sub-kernels, subsequences fore-between kernel; subsequences between-after kernel; and the subsequences between kernel from Section 2.4.1.

Experiment 5: Multiple kernel learning over the bag-of-n-grams sub-kernels described in Section 2.4.5. The bag-of-n-grams is composed of two kernels. the global context Kernel and the local context kernel. Moreover, these two kernels can be decomposed in three and two sub-kernels respectively. Then the kernels contained in this set are the global context fore-between kernel; the global context between-after kernel; the global context between kernel; the local context left entity kernel; and the local context right entity kernel.

Experiment 6: Multiple kernel learning over the Shortest Path Kernel form Section 2.4.3 plus the sub-kernels used in the Experiments 4 and 5.

4.1.2 Parameters

The experiments were tested using each available dataset individually AImed and SemEval . Moreover, in the first half of the experiments we solve SVMs by using an online approach with a single kernel, while for the remaining experiments, we use an online multiple kernel approach with a set of kernels. For both datasets and both SVM solvers, a maximum number of iterations corresponding to the parameter T described in Section 3.1 and in Section 3.2 was fixed to the value of 100.

The parameters λ form the online learner in Section 3.1, and the parameters λ_α and λ_β from the multiple kernel learner in Section 3.2 were fixed by finding the best performance depending on the experiment and on the dataset.

For the AImed dataset, experiments were tested using a 10-folds cross configuration. The parameter λ for Experiments 1, 2 and 3 was fixed to $\lambda = \frac{1}{50}$. As for the remaining experiments: Experiment 4 fixes $\lambda_\alpha = \frac{1}{50}$ and $\lambda_\beta = \frac{1}{50}$; Experiment 5 fixes $\lambda_\alpha = \frac{1}{30}$ and $\lambda_\beta = \frac{1}{30}$; and finally Experiment 6 fixes $\lambda_\alpha = \frac{1}{100}$ and $\lambda_\beta = \frac{1}{100}$.

The SemEval competition provided a train set and test sets. Unfortunately, SemEval is a bigger dataset than AImed and we could not perform as many experiences to fix the parameters as we did with AImed. The used parameters values were $\lambda = \frac{1}{50}$ for Experiment 1, 2 and 3. For Experiments 4, 5 and 6, we used $\lambda_\alpha = \frac{1}{50}$ and $\lambda_\beta = \frac{1}{50}$ in the multiple kernel learner.

4.1.3 Hardware and Software Configurations

The experiments were performed in a cluster environment with shared memory and composed by 13 machines, each with 8 Gb of memory RAM and equipped with an Intel Q6600 CPU, 4 cores of 2.4GHz. Our software was adapted in such a way that the training step could be executed by distributing the classifiers on the 13 machines. This configuration reduced the waiting time and we used MPJ Express[1], a Java library, to implement it.

4.2 Experimental Results

The experiments defined in Section 4.1 were conducted over the AImed and SemEval datasets. The results are reported in Tables 4.9 and 4.10 for the AImed and SemEval datasets, respectively. The discussions about the results are in Sections 4.2.1 and 4.2.2 respectively for AImed and SemEval datasets.

4.2.1 AImed Dataset

AImed is a one relationship dataset, it contains interactions between proteins, being the content described in a medicine context. Therefore, the most descriptive measure of the performance is the F1-Measure. Through the observation of the results expressed in Table 4.9, we can conclude that the Experiment 2 is the best in the range of experiments that used the online learner from Section 3.1. Moreover, we conclude that Experiment 1 was in general the best experiment in terms of Precision. The worst performance experiment is Experiment 3, it uses the shortest path dependency kernel which is based on grammatical information between words. Due to AImed being a dataset with a specific domain of protein interactions, its documents are not composed of common structured English sentences. This fact can be the main cause of Experiment 3 bad performance.

Multiple kernel learning solutions are introduced in Experiments 4, 5 and 6. We expected them to score the bests performances. As matter of fact, we observe a tendency for the multiple kernel learner with the sub-kernels perform better than the linear composed solution, which are the cases of Experiment 1 vs Experiment 4 and Experiment 2 vs Experiment 5.

The best performance was obtained by Experiment 5, a multiple kernel learning solution. Note that Experiment 2 and Experiment 5 share the same kernels, the bag-of-n-grams, so, we can conclude it is the best kernel for the AImed dataset.

Experiment 6 which is the experiment with the biggest set of kernels, including the kernels from other experiments, have a lower performance than Experiment 5. It was expected that Experiment 6 would

[1] http://mpj-express.org/

obtain the best performance because Experiment 5 uses a subset of kernels of Experiment 6. This behavior can be explained because Experiment 6 uses the shortest path kernel which is also the kernel used in the worst experiment, Experiment 3. This kernel most probably is adding bad information and making it harder to learn the right SVMs.

Finally, the results of using AImed dataset show us that multiple kernel learning solutions will perform better than online learners using linear combination of the same kernels. The best performance was obtained by Experiment 5, a multiple kernel learning solution.

Table 4.9: Results over the AImed corpus.

Experiment	Average Precision	Average Recall	Average F1-Measure
Experiment 1 (Subsequence kernel)	**69.35%**	36.52%	42.29%
Experiment 2 (Bag-Of-N-Grams kernel)	60.08%	48.83%	52.21%
Experiment 3 (Shortest Path kernel)	56.96%	21.73%	29.20%
Experiment 4 (MKL Subsequence sub-kernels)	58.61%	46.87%	50.27%
Experiment 5 (MKL Bag-Of-N-Grams kernel sub-kernels)	59.08%	**50.92%**	**53.86%**
Experiment 6 (MKL Experiments 4 and 5 sub-kernels plus Shortest Path kernel)	61.89%	46.73%	52.23%

4.2.2 SemEval Dataset

SemEval contains more than one relationship type, and so, the measures used are micro and macro averages of the standard measures (Precision, Recall and F1-measure). The SemEval competition evaluated the systems performance using the Macro measures. Therefore, We give more importance to the Macro F1-Measure in order to analyze the results of the experiments as in Table 4.10

When we observe the Experiments 1, 2 and 3 that use single kernel online learning and we compare these three experiments, we conclude that in term of Macro F1-Measure, Experiment 2 obtains the best score. Moreover, Experiment 2 is better in terms of Micro Recall, Micro F1-Measure, Macro Recall and Macro Precision, and that Experiment 1 gets the best performance in Macro Precision.

We observe that Experiment 2 got the best result in the objective of the competition. Experiment 2 also scores the best performance in terms of Micro Recall, Micro F1-Measure and Macro Recall, while the Experiment 4 perform better for Micro Precision and Experiment 6 for Macro Precision.

From the hypothesis of this thesis, we want to prove that multiple kernel learning is a better solution than single kernel learning. As we observe in this dataset, only Experiment 4 fulfills this constrain. Experiment 4 has a better performance than Experiment 1, which is the multiple kernel solution using the same kernel. Experiment 2 scores the best result which is higher than Experiment 5 and 6 which use common sub-kernels.

From the analysis of the SemEval results, we can not claim for sure that a multiple kernel solution will

always be better than a single kernel solution. Even tough, we believe this problem can be solved with a better parameter setup, as we explained before, for lack of time we could not perform as many experiences as we wanted to fix the parameters for SemEval.

Table 4.10: Results over the SemEval corpus.

Experiment	Micro Precision	Micro Recall	Micro F1-Measure	Macro Precision	Macro Recall	Macro F1-Measure
Experiment 1 (Subsequence kernel)	76.95%	55.46%	64.46%	74.73%	54.42%	62.98%
Experiment 2 (Bag-Of-N-Grams kernel)	76.00	**73.04%**	**74.49%**	75.01%	**72.24%**	**74.60%**
Experiment 3 (Shortest Path kernel)	74.51%	63.68%	68.67%	73.39%	63.04%	67.82%
Experiment 4 (MKL Subsequence sub-kernels)	**77.07%**	67.12%	71.75%	75.16%	66.17%	70.38%
Experiment 5 (MKL Bag-Of-N-Grams kernel sub-kernels)	76.46%	68.45%	72.23%	75.28%	67.58%	71.22%
Experiment 6 (MKL Experiments 4 and 5 sub-kernels plus Shortest Path kernel)	76.91%	69.33%	72.93%	**75.58%**	67.97%	71.57%

4.3 Summary

In this chapter we produced some experiments following the benchmark as in Section 3.3. The setup was previously presented in Section 4.1 and the results described and analyzed in Section 4.2.2. We conclude that for AImed a multiple kernel solution generally have a better performance than a single kernel solution that use the same kernel. As for the SemEval dataset, the results did not let us conclude that a multiple kernel learning solutions always perform better than a single kernel solution using a linear combinations of the same kernels. We expect that with a better parameter setup we could solve this problem.

Chapter 5

Conclusions

This chapter presents the main conclusions of this dissertation. The purpose of this dissertation was to prove that a new solution for the relationship extraction problem, based on SVM clasifiers and online multiple kernel learning, could be developed. To validate this idea we elaborated a benchmark, which included available datasets for relationship extraction and common pre-processing tools. Moreover, methods from the state-of-the-art were implemented. These methods are the subsequence kernel described in Section 2.4.1, the shortest path dependency kernel described in Section 2.4.3 and the bag-of-n-grams kernel described in Section 2.4.5, all three of them being kernel-based methods.

We applied the selected kernels to an online SVM learner, and, by maintaining the specifications of the benchmark, we evaluated the kernel-based methods in common conditions. After, knowing that some of these kernels were composed by linear compositions of sub-kernels, we created sets of kernels and/or sub-kernels. These sets were then used with a multiple kernel SVM learner in the same benchmark. Through a performance analysis, we aimed to prove the advantages of using a solution based on multiple kernels.

Finally, the results showed us that the AImed dataset, we can conclude that multiple kernel learning solution proves to obtain better performances than the single kernel solutions. As for the case of the SemEva dataset, the best experiment is based on single kernel learning, and so, we could not assert the benefit of a multiple kernel based solution, over a single kernel one

In text mining, the act of improving a relationship extraction module can result in significant advances in the area. Note that relationship extraction systems have low performance when compared to other machine learning related problems (F1-measures $< 80\%$). Therefore, advances in relationship extraction can mean an important improvement for diverse systems using structured data, for instance, question answering and search engines.

Section 5.1 summarizes the main contributions in this dissertation and Section 5.2 lists the future work

that can be performed to improve upon this dissertation and the problem of relationship extraction.

5.1 Summary of Contributions

The main contributions of this MSc thesis are as follows:

- Implementation of an online SVMs learner for relationship extraction, called Pegasos.
- The development of two extensions for the online learner which allow us to use a kernel and a bias term.
- Development and implementation of an online learner for relationship extraction with the ability to learn weights for a set of Kernels while learning the SVMs.
- Development of a benchmark for relationship extraction methods. It includes the use of AImed and SemEval datasets. The evaluation using a set of measures for each dataset, Precision, Recall and F1-Measure for AImed, and, Macro and Micro measures for SemEval. The experiments use a common set of pre-processing NLP tools from *Apache OpenNLP*and *Stanford CoreNLP* frameworks. The benchmark also includes the implementation of three state-of-the-art kernel-based methods, the subsequences kernels, the bag-of-n-grams kernel and the shortest path dependency kernel.
- Application of the developed benchmark for comparing a set of kernel-based methods for relationship extraction. The results from an extensive set of experiments showed that although the best experiments are based on multiple kernel learning, we could not always assert the benefit of a multiple kernel based solution over a single kernel one.

5.2 Future Work

It is usually impossible to claim that a work has come to an end. In fact, a set of possible improvements can eventually be explored. We will consider the following six possible improvements as future work:

Datasets: Although we used two datasets with different domains, a more comprehensive benchmark can be developed by adding more datasets, specially with more sentences and from distinct domains. We already implemented a loader for reACE, a dataset that contains news articles and is a re-annotation of ACE 2013. Due to the size of this dataset and the lack of physical machines for experimentation, we were not able to execute this set of experiments before the dissertation's deadline.

Kernel Methods: We can incorporate more kernel-based methods from the literature in the benchmark. These implementations can produce different sets of kernels that can be used with the multiple kernel solution and improve the results.

Tokens enrichment: It was detailed in this dissertation that some kernels can use rich information associated to the token information with word classes, and then, when comparing pairs of tokens, a function would return the number of common word classes. We used a fixed set of word classes, so more experiences can be performed using different sets of word classes, for instance word classes obtained from the wordnet [20] are used in the literature.

Parameters Setup: Better performances can be achieved by adjusting the different parameters. Executing more experiences with different values can also lead to a better performance of the multiple kernel learning based solutions over the remaining ones.

Hierarchical Classification: In multiple classification problems using SVMs, we usually attribute a classifier to each class against the remaining classes or assign a classifier to each possible pair of classes. Furthermore, we can create a hierarchy using classifiers, for instance, when predicting a relationship type, we can first classify a pair of entities by containing, or not, a relationship. If it does, we then classify the type of the relationship. We can go further, for example, when an asymmetric type is predicted. After, we can classify the orientation. Actually, a solution based on hierarchical classification was implemented and tested over a dataset but because of time limitations we could not afford to proceed.

Kernel Weighting: In the process of kernel weighting, the fact that some kernels compute faster than others is not taken into account. This may be a problem, as it can be important to some software applications to design a less computationally expensive model. A solution for this, can pass by re-designing the optimization function in order for it to attribute penalties to expensive kernels.

Bibliography

[1] Antti Airola, Sampo Pyysalo, Jari Björne, Tapio Pahikkala, Filip Ginter, and Tapio Salakoski. A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, BioNLP '08, pages 1–9, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, HLT '91, pages 112–116, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[4] Razvan Bunescu and Raymond J. Mooney. Subsequence kernels for relation extraction. In *Proceedings of the 9th Conference on Natural Language Learning (CoNLL-2005)*, Ann Arbor, MI, July 2006. Available at urlhttp://www.cs.utexas.edu/users/ml/publication/ie.html.

[5] Razvan C. Bunescu and Raymond J. Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 724–731, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[6] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, pages 423 – 430, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[7] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 26(1):57–61, January 1983.

[8] Jr. Forney, G.D. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268 – 278, March 1973.

[9] Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, ACL '06, Trento, Italy, April 2006.

[10] Zhenmei Gu and Nick Cercone. Segment-based hidden markov models for information extraction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 481–488, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[11] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan 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.

[12] Chu-Ren Huang, Petr Šimon, Shu-Kai Hsieh, and Laurent Prévot. Rethinking chinese word segmentation: tokenization, character classification, or wordbreak identification. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 69–72, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[13] Jing Jiang and Chengxiang Zhai. A systematic exploration of the feature space for relation extraction. In *In Proceedings of Human Language Technologies: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT'07*, pages 113–120. Association for Computational Linguistics, 2007.

[14] 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, 2008.

[15] Nanda Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, ACLdemo '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[16] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[17] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444, March 2002.

[18] Mónica Marrero, Sonia Sánchez-Cuadrado, Jorge Morato Lara, and George Andreadakis. Evaluation of named entity extraction systems. *Advances in Computational Linguistics. Research in Computing Science*, 41:47–58, 2009.

[19] Andrei Mikheev. Tagging sentence boundaries. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, NAACL 2000, pages 264–271, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[20] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[21] Scott Miller, Michael Crystal, Heidi Fox, Lance Ramshaw, Richard Schwartz, Rebecca Stone, and Ralph Weischedel. Algorithms that learn to extract information; bbn: Description of the sift system as used for muc-7. In *In Proceedings of MUC-7*, 1998.

[22] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, January 2007.

[23] David D. Palmer. Satz - an adaptive sentence segmentation system. 1994.

[24] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proceedings of HLT-NAACL*, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics.

[25] Adwait Ratnaparkhi. A Maximum Entropy Model for Part-Of-Speech Tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Empirical Methods in Natural Language Processing*, pages 133–142, 1996.

[26] Thomas Reps. “maximal-munch” tokenization in linear time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 20(2):259–273, March 1998.

[27] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, ICML ’07, pages 807–814, New York, NY, USA, 2007. ACM.

[28] Maxim Sidorov. Hidden markov models and steganalysis. In *Proceedings of the 2004 workshop on Multimedia and security*, MM&Sec ’04, pages 63–67, New York, NY, USA, 2004. ACM.

[29] Yoshimasa Tsuruoka, Jun’ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL ’09, pages 477–485, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[30] Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP ’02, pages 71–78, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.