Combining Feature-based Kernel with Tree kernel for Extracting Relations

Author:
Henrik Bayer Nielsen

Supervisor:
Henrik Legind Larsen

Abstract

This report proposes a composite kernel method of semantic relation extraction between named entities within natural language documents. Using two kernel methods, the benefits from both are combined to gain an increase in performance compared to earlier approaches. 1) A linear kernel processing linguistic features, such as word-span, order-of-entities and word-type. 2) A tree kernel computing the similarity of a relation type with the shortest path-enclosed tree between a pair of candidate entities. Experiments are done using previous implemented methods, such as context sensitiveness and latent annotations to measure their impact on the performance. Evaluating on a dataset for the relation extraction task at the Conference on Computational Natural Language Learning from 2004, the results obtained are on par with previous state-of-the-art approaches on the same dataset.

June 6, 2012

Preface

This report has been written in connection with the 10th semester project for master of science in computer engineering. The period of the project was from February 1, 2012 to June 6, 2012. This report documents my master thesis in Information Extraction (IE) and Relation Extraction (RE). I present my approach on the relation extraction task and compare it to the current state-of-the-art. Additionally, I suggest an approach for post processing the relation extraction output.

The report also comes with a CD, the contents of the CD are:

- Report in pdf-format
- All referred articles in pdf-format
- My relation extraction system including possibility of post processing

Table of Contents

1 Introduction

Today, enormous amounts of information are available to us on the Internet. Whether the information is comments and opinions from blogs and micro-blogging sites, or facts from articles and news sites, it has great potential for gathering knowledge or opinions about a specific topic. In computer science, the task of gathering this information is known as Information Extraction (IE), which is one of the key tasks in the field of Natural Language Processing (NLP). IE applications attempt to identify relevant information from a large amount of unstructured text documents in the form of natural language, and store them in a structured format.

The research in NLP and IE specifically was first initiated by the Message Understanding Conference (MUC, 1987-1998) [1] and then further promoted by the Conference on Computational Natural Language Learning (CoNLL, 1997-2011) [2] and the National Institute of Standards and Technology (NIST) Automatic Content Extraction (ACE, 2000-2008) program [3, 4]. In the NIST ACE program, two subtasks of IE are defined as:

- Entity Detection and Recognition (EDR), also known as Named Entity Recognition (NER)[1]
- Relation Detection and Recognition (RDR), also known as Relation Extraction (RE)[2]

According to the ACE program, an entity is an object or a set of objects in the world, and a relation is an explicitly or implicitly stated relationship between the entities. The objective of NER is to detect and recognize entities in unstructured text, while the objective of relation extraction is to detect and recognize semantic relationships between predefined types of entities previously recognized by the NER subtask. For Example, given the sentence "Henrik works as a system developer at Computerfriend.dk.", the named entities "Henrik" (person) and "Computerfriend.dk" (company) are expected to be identified and extracted by the NER, while the relation extraction is expected to identify and extract the semantic relation *work_for* in which "Henrik" is the first argument (worker) and "Computerfriend.dk" is the second (employer).

NER and especially relation extraction have potential to be very useful in many NLP applications such as for example question answering and text summarization. Unfortunately though, due to the limited accuracy in the current state-of-the-art syntactic and semantic parsing, not to forget the

[1]Will be referred to as NER from here on
[2]Will be referred to as relation extraction from here on

complexity and variability of semantic relations in natural language, reliably extracting, such semantic relations between named entities, is still a difficult and unresolved problem.

In the field of NER, there are implementations with near-human performance. In 2002, Zhou and Su [5] developed a machine-learning named entity recognizer using the Hidden Markov Model (HMM) which achieved an F-measure of 96.6% on the MUC-6 corpus [6]. F-measure is a function combining the Precision and Recall of a system, written as:

$$F = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1}$$

where

$$\text{Precision} = \frac{\text{number of correct answers found by the system}}{\text{number of answers found by the system}},$$
$$\text{Recall} = \frac{\text{number of correct answers found by the system}}{\text{number of correct answers in the test corpus}}$$

An F-measure of 96.6% is close to human performance, as not even humans are capable of reaching 100% without extensive knowledge of the subject in question, which is improbable seeing the test is done on a corpus containing hundreds of documents of different subjects. Because of this, in the recent years researchers have moved their focus from NER to relation extraction.

Initially, relation extraction was approached as a classification problem using a machine learning algorithm relying on feature-based representations of the input instances [7, 8, 9, 10]. These systems first transform the relation examples into the corresponding numerical vectors of various syntactic and semantic features[3], next they apply a machine learning approach (such as HMM or Support Vector Machines (SVM)[11]) to detect and classify them into predefined types of semantic relations between named entities. However, according to Zhou et. al [8], the feature-based methods usually fail to effectively capture the critical structural information inherent in the parse trees, making it difficult for the feature-based methods to extract new and effective features to further improve performance. Because of this limitation, researchers have turned to kernel-based methods[4], which directly computes the similarity between two discrete objects, like parse trees with rich structural information. Since the first kernel-based solution [12], the use of kernel methods have continued to increase the performance in relation extraction due to its effectiveness in modelling discrete objects [13, 14, 15, 16, 17, 18].

[3]The features used include lexical items, phrase and chunk information, syntactic parse trees, deep semantic information and entity-related information

[4]Kernel methods will be described further in Section 3.1

This report gathers the knowledge I have gained while exploring the methodologies used in previous kernel-based relation extraction approaches. I present my approach on the relation extraction task and suggest an approach to social network post processing. In Section 2, I further analyse the related work in kernel-based relation extraction applications. In Section 3, I analyse the state-of-the-art relation extraction methodologies. In Section 4, my relation extraction approach is described. In Section 5, experiments during the project are documented. I discuss and suggest a possible approach for post processing in Section 6. In Section 7, I discuss results and limitations of my relation extraction approach. I conclude the project in Section 8. References are listed in Section 9.

2 Related Work

Many machine learning methods have been proposed for relation extraction, including unsupervised learning, semi-supervised learning [19] and supervised learning. In a supervised learning setting, representative related work can be classified into feature-based [7, 8, 9, 10] or kernel-based methods [12, 13, 14, 15, 16, 17, 18].

In [7], Kambhatla employs Maximum Entropy (ME) models to combine lexical, syntactic, and semantic features. The features he uses include entity type, mention level, overlap, dependency and parse tree. He achieves an F-measure of 52.8% on the ACE 2003 corpus. In [8], Zhou et al. use SVM to incorporate diverse lexical, syntactic, and semantic knowledge in feature-based relation extraction. In addition to the features used by [7] they also use chunking and other semantic resources and they achieve an F-measure of 55.5% on the ACE 2003 corpus. Roth and Yih take a different approach in [9] where they criticise the "pipeline" approach in NLP and develop a linear programming formulation where they simultaneously learn named entities and their relationships. They develop a general approach to inference over the outcomes of predictors in the presence of general constraints, allowing them to efficiently incorporate domain and task specific constraints at decision time. They evaluate their results on the CoNLL'04 corpus [20] and achieve F-measures between 51.6% and 81.7% on the five relation types (*located_in, work_for, orgBased_in, live_in, kill*). In [10] Jiang and Zhai systematically explore a large space of features for relation extraction and evaluate the effectiveness of different subspaces. They present a general definition of a feature space, based on a graphical representation of relation instances. Their research shows that using only basic unit features is generally sufficient to achieve state-of-the-art performance, while over-inclusion of complex features may hurt the performance. They achieved an F-measure of 70.7% using the ME classifier on the ACE 2004 corpus.

However, in all of the feature-based approaches, they fail to effectively capture the critical structural information inherent in the parse trees, and unfortunately for feature-based methods, almost all the flat features related to lexical, syntactic, and semantic knowledge have been systematically explored, making further improvements of relation extraction through feature engineering difficult. Instead, kernel methods have the potential of better modelling structured objects due to its ability of directly measuring the similarity between two structured objects, without explicitly enumerating their substructures. For example, the kernel method can capture the structural information in a parse tree and consequently avoid the burden of feature engineering by directly computing the similarity between any two trees.

The first kernel-based relation extraction approach was made by Zelenko et al. [12], who proposed a kernel between shallow parse trees, which recursively matched nodes from roots to leaves in a top-down manner. They achieved promising results on two simple relation extraction tasks. In [13], Culotta and Sorensen extended the solution of [12] to estimate the similarity between the dependency trees of sentences. However, their solution required the matched nodes to be at the same layer and in the identical path starting from the roots to the current nodes, which is a very strong constraint. Their solution achieved very high precision, but not surprisingly very low recall and achieved an F-measure of 45.8% on the five relation types of the ACE 2003 corpus. In correlation with the very strong constraint, once a node cannot be matched with any node at the same depth in another tree, all the sub-trees below this node are discarded even if some of them can be matched to their counterparts in another tree due to the nature of the top-down node matching mechanism of this kernel. In [14], Bunescu and Mooney proposes a shortest path dependency tree kernel which achieves an F-measure of 52.5% on the ACE 2003 corpus. Their kernel simply sums up the number of common word classes at each position in the two paths. Based on observations, they argue that the information required to assert a relationship between two named entities could be typically captured by the shortest path between the two entities in the dependency graph. As with [13], they suffer from low recall because the two shortest paths must have the same length to get a non-zero similarity score in the kernel computation. Also, the shortest path may not be able to well preserve the necessary structural dependency tree information, leading to further reduction of the system's recall. Zhang et al. [15] proposes a composite kernel[5] for relation extraction. The kernel consists

[5]A composite kernel is a kernel consisting of two or more individual kernels

of an entity kernel that allows for entity-related features and a Convolution Tree Kernel (CTK) that models syntactic information of relation examples. They claim that the composite kernel can effectively capture both flat and structured features without the need for extensive feature engineering, and that it also easily can scale to include more features. They achieve an F-measure of 70.9% on the ACE 2003 corpus, which is a significant increase in performance compared to the earlier approaches by [13] and [14]. The approach by Giuliano et al. [16] is based solely on shallow linguistic processing, such as tokenization, sentence splitting, part-of-speech tagging, and lemmatization. They use a composite kernel to integrate two different information sources: (i) the whole sentence where the relation appears, and (ii) the local contexts around the interacting entities. From evaluation on the CoNLL'04 corpus [20], they achieve F-measures between 65.8% and 80.5% on the five relation types. While their results are close to the state-of-the-art, their solution could be further enhanced by adding another individual kernel function to contribute with syntactic information. In 2007, Zhou et al. [17] proposed a tree kernel with context-sensitive structured parse tree information for relation extraction. It resolved two critical problems in previous tree kernels for relation extraction. First, it automatically determines a dynamic context-sensitive tree span for relation extraction by extending the widely-used Shortest Path-enclosed Tree (SPT) to include necessary context information outside the SPT. Second, it proposes a context-sensitive CTK, which enumerates both context-free and context-sensitive sub-trees by considering their ancestor node paths as their contexts. They achieve an F-measure of 74.1% and 75.8% on the ACE 2003 and 2004 corpora. While this is a significant improvement of the solution proposed by Zhang et. al [15], the CS-SPT only recovers part of the contextual information and still contains as much noisy information as the ordinary SPT. Finally, in [18], Zhou et al. extend the context-sensitive CTK in [17] with approximate matching and further expand the CS-SPT with unique portion labelling and latent annotations. They achieve an F-measure of 75.7% and 77.6% on the ACE 2003 and 2004 corpora, which currently is the best results obtained by anyone in the field of relation extraction.

3 Methodologies for Extracting Relations

Having analyzed the related work, one can argue that the two most significant topics of current state-of-the-art relation extraction approaches are:

- Parse Tree Structure
- Machine Learning

While the problem of relation extraction initially was approached as a classification problem using a machine learning algorithm relying on feature-based representations of the input instances, due to syntactic limitations of the feature representations researchers have switched their focus onto kernel methods for directly computing the similarity of discrete objects such as syntactic parse tree representations of the semantic relations.

For Support Vector Machines (SVM), using a training data set, the machine learning algorithm generates a model containing support vectors, or in this case example parse trees, which is then used to classify new relations from their parse tree representation.

Additionally, both the syntax and span of the parse trees have a great impact on the performance of the relation extraction system, as usually there's also a lot of unnecessary "noise" within a relation phrase or sentence. These two topics and their underlying theory will be studied further in the next two sub sections.

Considering the related work, it is obvious that the F-measure (1), generally is accepted as the best performance measure for systems within information extraction. The F-measure is equally determined by the precision and recall, but in some domains the precision might be more important or vice versa, in which case the F-measure is not ideal. In Section 3.3 I study a possible solution to this through a generalized F-measure.

3.1 Parse Tree Structure

For kernel-based relation extraction, the relations are encapsulated in parse trees, see Figure 1 for an example. In most cases, the parse trees also contain a lot of unnecessary data for the relation extraction task and thus one could represent the relation instances by only a portion of the parse tree to avoid unnecessary "noise". Before doing this however, it is of great importance to understand which portion of the parse tree that is necessary for the relation extraction task within the tree kernel computation.

In previous work [15, 17, 18], researchers have explored different parse tree structures and studied their impact on the relation extraction task. In [15], Zhang et al. explored five different parse tree structures:

1. Minimum Complete Tree (MCT)
2. Path-enclosed Tree (PT)
3. Context-Sensitive Path Tree (CSPT)
4. Flattened Path-enclosed Tree (FPT)
5. Flattened CSPT (FCSPT)

See Figure 1 for an example on the sentence *Several students have jobs in their spare time, Henrik works as a system developer at Computerfriend.dk in Esbjerg.*

MCT refers to the complete sub-tree rooted by the nearest common ancestor of the two entities under consideration, as can be seen on Figure 1(a), the nearest common ancestor node in this case is a noun phrase (NP).

PT refers to the smallest common sub-tree including the two entities, illustrated on Figure 1(b). Or said in another way, the sub-tree is enclosed by the shortest path linking the two entities in the parse tree (hence why it is also known as Shortest Path-enclosed Tree (SPT)).

CSPT refers to the PT extended with context sensitivity, which in this case means extending the PT with the first left word of entity 1, and the first right word of entity 2. Figure 1(c) illustrates the CSPT representation.

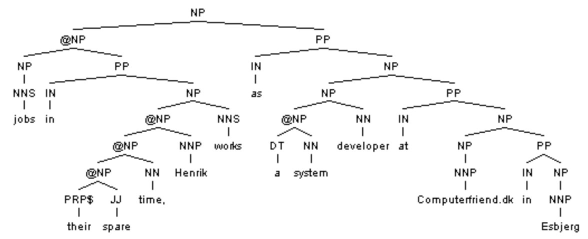
FPT refers to the flattened PT, which means that all single in and out arcs of nonterminal nodes (except POS nodes) are removed, as can be seen on Figure 1(d).

FCSPT is the same as FPT except it is the CSPT that is flattened instead, the FCSPT can be seen on Figure 1(e) [15].

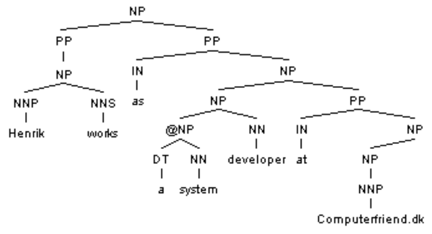
After studying these five parse tree representations, Zhang et al. acknowledged that the best performance was achieved by the Shortest Path-enclosed Tree. They argued that the most significant information was within the SPT and that improper inclusion of additional structural information would only introduce more "noise" to the system [15].

3.1.1 Context-Sensitive Shortest Path-enclosed Tree

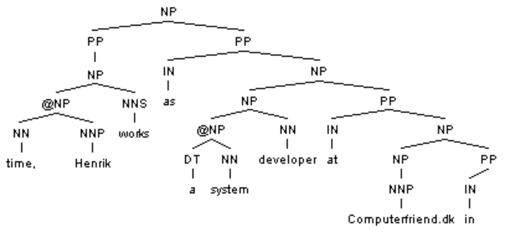
On the contrary, later in [17], Zhou et al. argues that the CSPT should outperform the SPT, as in some cases the SPT will not contain enough information to identify the semantic relation. For example, consider the sentence *Ken and Krystal got divorced.* Here, *got divorced* is critical to determine the relation between *Ken* and *Krystal*, but the SPT would only contain *Ken and Krystal* which obviously is not enough information to identify the relation between them.



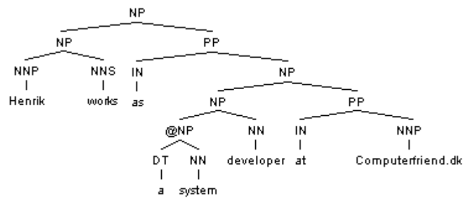
(a) MCT



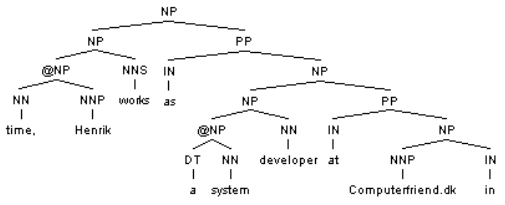
(b) PT



(c) CSPT



(d) FPT



(e) FCSPT

Figure 1: Different parse tree representations of a relation instance in the example sentence *Several students have jobs in their spare time, Henrik works as a system developer at Computerfriend.dk in Esbjerg*, where *Henrik* (PERSON) is the first entity and *Computerfriend.dk* (COMPANY) is the second entity in the semantic relation *work_for*.

One could argue that the problem with the CSPT proposed by Zhang et al. in [15] was that it only considered the availability of the entities' siblings[6], failing to consider the following two important elements:

1. Is the information within the SPT enough to determine the semantic relation between the two entities? In some cases SPT is enough, for example in the embedded case *Ken got divorced with Ken's wife.*, where one entity is embedded in the other i.e. *Ken* and *Ken's wife*. However, in the coordinated case as illustrated in the example sentence *Ken and Krystal got divorced* above, the SPT is not enough.

[6]As explained earlier, the CSPT by [15] extends the SPT with the first left word of entity 1 and first right word of entity 2

2. If the SPT is insufficient for extracting the relation, how can it be extended to include necessary context information?

In [17], Zhou et al. study 100 relation instances from the ACE 2003 training data set, and based on their observations, they implement an algorithm to dynamically determine the necessary tree span for the relation extraction task. The algorithm is designed to dynamically determine the tree span according to the category of the tree and its context. They classify five tree span categories:

1. Embedded, where one entity is embedded into another, for example *Ken* and *Ken's wife*

2. PP-linked, where one entity is linked to another entity via PP attachment, for example *System developer* and *Computerfriend.dk* in *System developer at Computerfriend.dk developed...*

3. Semi-structured, where the sentence consists of a sequence of noun phrases, for example *Henrik* and *Computerfriend.dk* in *Henrik, Computerfriend.dk, Esbjerg*

4. Descriptive, where the relation is described within the sentence, for example *His employer* and *Computerfriend.dk* in *His employer, Computerfriend.dk, launched a new...*

5. Predicate-linked and others, where the predicate information is needed for determining the relationship between two entities, for example *Ken* and *Krystal* in *Ken and Krystal got divorced*

Zhou et al. designed their algorithm so that, given a parse tree containing two entities, it first determines the tree span category and then extends the tree span accordingly [17]. Initially, the parse tree span is based on the SPT, and only when the tree span belongs to the *predicate-linked* category, the tree span is expanded further. For the four other tree span categories, the SPT is sufficient, and as the *predicate-linked* category only occupy around 20% of all the cases, this also explains why Zhang et al. in [15] achieved better results with the SPT than their CSPT.

3.1.2 Enriched CS-SPT

In [18], Zhou et al. further extend their former solution in [17] with unique portion labelling and latent annotations. As will be described later in Section 3.2.2 and 3.2.3, the Convolution Tree Kernel (CTK) counts the number

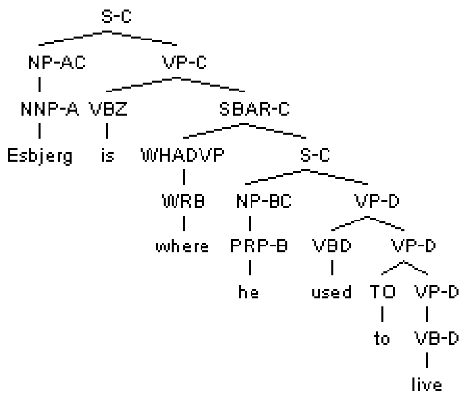


Figure 2: The enriched CS-SPT with unique portion labels on the sentence *Esbjerg is where he used to live*

of common sub-trees to measure the similarity between two parse trees, but Zhou et al. argue that both the SPT and CS-SPT allow wrong matching between different portions of parse trees when computing their similarity with the CTK. To overcome this problem, and avoid wrong matching between different portions of parse trees, they apply unique portion labels to explicitly distinguish different tree portions. They divide the CS-SPT into four portions:

A) The first mention

B) The second mention

C) The shortest path between the two mentions

D) The expanded predicate path for the predicate-linked category

For an example on the sentence *Esbjerg is where he used to live*, see Figure 2. As can be seen, the two mentions *Esbjerg* and *he* are labeled with *A* and *B* as first and second mention, respectively. The nodes included in the shortest path between the two mentions are labeled with *C*, for example *SBAR-C* means that the node *SBAR* is part of the shortest path between the two mentions. The same goes for the nodes marked with *D* and the expanded predicate path for the predicate-linked category.

The purpose of the unique portion labels is then to only match a sub-tree in a particular portion of a parse tree with sub-trees in the same portion of another parse tree. For example, a sub-tree within the second mention portion of a parse tree will only be matched with sub-trees inside the second mention portion of another parse tree [18].

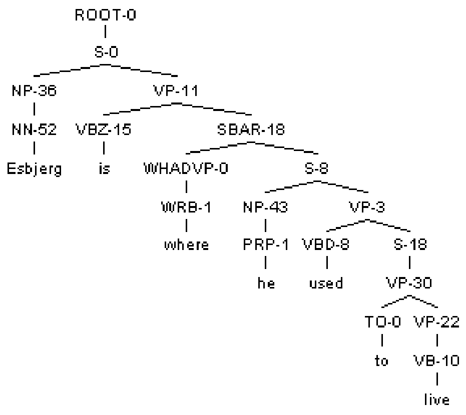


Figure 3: The enriched CS-SPT with latent annotations on the sentence *Esbjerg is where he used to live*

Another extension to the CS-SPT that Zhou et al. proposes, is the latent annotations [18]. They employ latent annotations on the basic non-terminal nodes, such as NP, NNP etc. By using latent annotations, they divide the non-terminals into several finer categories. For example, the treebank[7] non-terminal category NP may be expanded to several finer categories such as object NPs or subject NPs. To employ latent annotations, Zhou et al. uses the Berkeley parser [21][8]. See Figure 3 for an example on the sentence *Esbjerg is where he used to live*

3.2 Machine Learning

As described earlier in Section 3, kernel methods are used to compute the similarity between two discrete objects, e.g. syntactic parse tree representations of semantic relations. The kernel method is employed in a machine learning (ML) algorithm. By definition:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [22]

More specifically, an ML algorithm allows a computer to evolve behaviours based on empirical data[9]. Machine learning is concerned with the develop-

[7]A treebank is a text corpus in which each sentence has been parsed, i.e. annotated with syntactic structure

[8]The Berkeley parser is also used to parse all of the other parse tree examples in this report

[9]Data produced by an experiment or observation, e.g. datasets such as the CoNLL'04 corpus

ment of algorithms allowing the machine to learn via inductive inference[10]. In relation extraction, the task of machine learning is classification, or pattern recognition, in which machines "learn" to:

1. Automatically recognize complex patterns
2. Distinguish between exemplars based on their different patterns
3. Make intelligent predictions on their class

As briefly mentioned in the Related Work in Section 2, there have been several machine learning methods proposed for relation extraction. Based on their desired output, one can organize the ML algorithms into different categories, such as *supervised*, *unsupervised*, *semi-supervised*, and *reinforcement* learning algorithms.

Most relevant for today's relation extraction is supervised learning. Supervised learning generates a function that maps inputs to desired outputs (also called *labels*, because they are often provided by human experts labelling the training data). For example, in a classification problem, the learner approximates a function mapping a parse tree to a relation by looking at previous input-output examples of the function. The most popular supervised learning method in relation extraction is Support Vector Machines (SVM).

3.2.1 Support Vector Machine

SVM is a supervised learning algorithm for binary classification[11], which has been successfully applied in many relation extraction approaches, from early approaches in [12, 13] to today's state-of-the-art systems in [17, 18].

One can define a set of training examples each of which is labelled with either positive or negative class tag $(\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)$. Here, $\mathbf{x}_j \in \mathbf{R}^n$ is a feature vector of the j-th example represented by an n-dimensional vector. $y_j \in \{1, -1\}$ is the label of the j-th example (1 for positive and -1 for negative), and n is the total number of training examples derived from the training set. Specifically, the SVM learns to find a hyperplane that separates positive and negative examples with the highest possible accuracy. This is also known as finding the maximal margin. First, any hyperplane can be written as the set of points $\mathbf{x}$ satisfying:

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \tag{2}$$

[10]The theory of prediction based on observations, e.g. predicting the next symbol based upon a given series of symbols[23]

[11]Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other

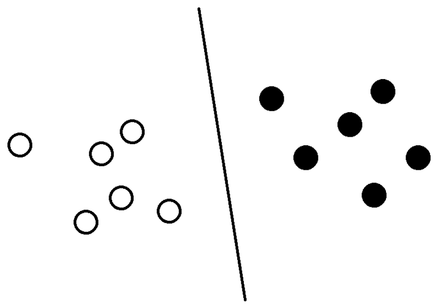


Figure 4: A hyperplane separating positive and negative examples

where $\mathbf{w}$ is the hyperplane's normal vector and b determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$[24]. Suppose the hyperplane separates the training data into positive and negative parts. Initially, several of such hyperplanes exist, see Figure 4 for an example of such a hyperplane. SVM then tries to find the optimal hyperplane that maximizes the margins between the nearest examples to the hyperplane. For each class, these hyperplanes can be described by the equations

$$\mathbf{w} \cdot \mathbf{x}_1 - b = 1$$
$$\mathbf{w} \cdot \mathbf{x}_2 - b = -1$$

where $\mathbf{x}_1$ is the set of points of the first class, and $\mathbf{x}_2$ is the set of points of the second class. The distance between these two hyperplanes, the margin M, can be expressed as:

$$M = \frac{2}{\|w\|} \tag{3}$$

Maximizing M is equivalent to minimizing $\|w\|$, which is equivalent to solving the following optimization problem:

$$\text{Minimize: } \frac{1}{2}\|w\|^2 \tag{4}$$
$$\text{subject to: } y_j(\mathbf{w} \cdot \mathbf{x}_j - b) \geq 1 \tag{5}$$

where $\|w\|$ is substituted by $\frac{1}{2}\|w\|^2$ for mathematical convenience as $\|w\|$ involves a square root making it difficult to solve. The constraint in (5) is to prevent data points from falling into the margin. See Figure 5 for an example of a maximum-margin hyperplane.

Since an ordinary SVM only solves the binary classification problem, it must be extended to a multiclass SVM in order for it to be eligible for semantic relation extraction. Multiclass SVMs aim to design labels to instances,

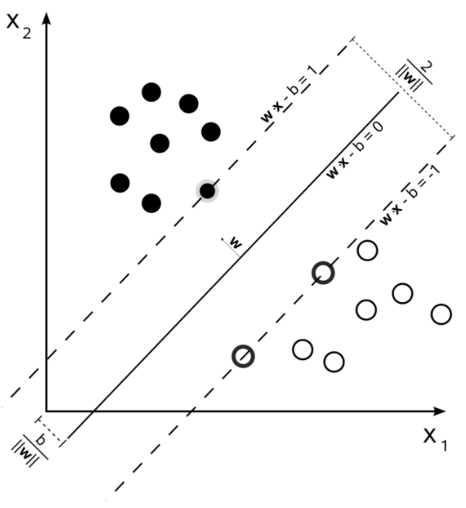


Figure 5: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called support vectors[25]

where the labels are drawn from a finite set of several elements, which means the single multiclass problem is reduced into a multiple binary classification problems. The most common multiclass SVM, is the one-against-all method. For example, in relation extraction, the one-against-all SVM is trained for every semantic relation to be able to distinguish between examples of its current class and the rest.

3.2.2 Kernel Function

In kernel-based relation extraction, the dot product is replaced by a kernel function. As briefly mentioned in the start of Section 3, kernel-based methods are used to directly compute the similarity between discrete objects, such as syntactic parse trees. More precisely, a kernel function K over the object space X is a binary function

$$K : X \times X \to [0, \infty] \tag{6}$$

mapping a pair of objects $x, y \in X$ to their similarity score $K(x, y)$. Additionally, a kernel function is required to be both *symmetric* and *positive-semidefinite.*

A binary function K is *symmetric* over X, if $\forall x, y \in X$:

$$K(x, y) = K(y, x) \tag{7}$$

A binary function K is *positive-semidefinite*, if $\forall x_1, x_2, ..., x_n \in X$, the $n \times n$ matrix $(K(x_i, x_j))_{ij}$ is *positive-semidefinite*[12]. A *positive-semidefinite* matrix is a Hermitian matrix, i.e. $(K(x_i, x_j))_{ij} = \overline{(K(x_i, x_j))}_{ji}$ all of whose eigenvalues are nonnegative.

It can be shown that any kernel function can measure the similarity between two input instances by implicitly computing the dot product of certain features in high-dimensional feature spaces without explicitly enumerating all the features[18]. That is, there exist features $f(\cdot) = (f_1(\cdot), f_2(\cdot), ...)$, $f_i : X \to R$, so that

$$K(x, y) = \langle f(x), f(y)\rangle^{12} \tag{8}$$

As mentioned, in many cases, it may be possible to compute the dot product of certain features without explicitly enumerating all the features. An example of such a case is the subsequence kernel [26]. In the subsequence kernel, the inputs are string of characters, and the kernel function computes the number of common character subsequences between two strings, where each subsequence match is additionally decreased by the factor reflecting how spread out the matched subsequences are in the original sequences. Despite the exponential number of features, or in this case subsequences, it is possible to compute the subsequence kernel in polynomial time. Because of this, the long-range features can be exploited without enumerating the features explicitly [26].

These kinds of kernels are also referred to as convolution kernels [27], which aims to capture structural information in terms of substructures.

3.2.3 Convolution Tree Kernel

As a specialized convolution kernel, the convolution tree kernel $K_{\text{CTK}}(T_1, T_2)$ counts the number of common sub-trees[13] as the syntactic structure similarity between the two parse trees T_1 and T_2 [28]. See Figure 6 for an example[14]. Where in this case, a parse tree T is implicitly represented by a vector of integer counts of each sub-tree type (regardless of its ancestors):

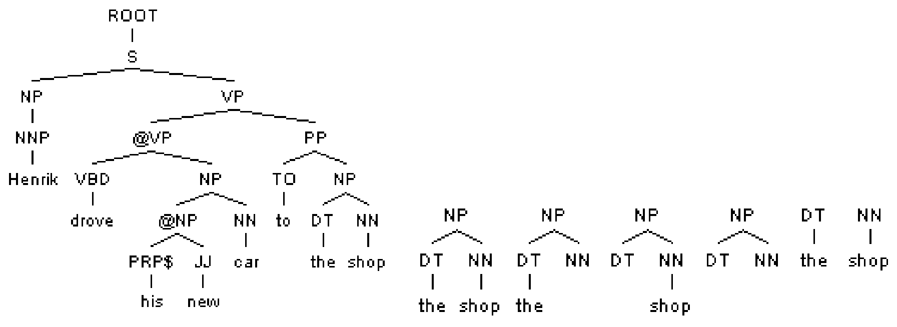
$$\mathbf{h}(T) = (h_1(T), ..., h_i(T), ..., h_n(T)) \tag{9}$$

where $h_i(T)$ is the occurrence number of the i-th subtree type in T [18]. Unfortunately though, since the number of different sub-trees increase exponentially with the parse tree size, it is not computationally feasible to use

[12] $\langle a, b\rangle$ denotes the dot product of vectors a and b

[13] A sub-tree can be defined to be any sub-graph which includes more than one node, restricted by the fact that the entire production rule must be included. For example, the sub-graph (NP (NN shop)) is not a sub-tree and is excluded because it contains only part of the production rule NP $\to$ DT NN.

[14] Part of speech (POS) tags: NP (noun phrase), DT (determiner), NN (noun, singular), PP (prepositional phrase), VP (verb phrase) etc.



(a) (b)

Figure 6: (a) An example tree of the sentence *Henrik drove his new car to the shop*. (b) All of the sub-trees for the NP *the shop* covered by the convolution kernel [28].

the feature vector $\mathbf{h}(T)$ directly. In [28], they attempt to solve this computational issue by proposing a CTK to implicitly compute the dot product between the high-dimensional vectors as follows:

$$\begin{aligned}
&K_{\text{CTK}}(T_1, T_2) = \langle \mathbf{h}(T_1), \mathbf{h}(T_2) \rangle \\
=&\sum_i (h_i(T_1) \cdot h_i(T_2)) \\
=&\sum_i \left(\left(\sum_{n_1 \in N_1} I_i(n_1) \right) \left(\sum_{n_2 \in N_2} I_i(n_2) \right) \right) \\
=&\sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2)
\end{aligned} \tag{10}$$

where N_1 and N_2 are the sets of nodes in trees T_1 and T_2, $I_i(n)$ is an indicator function which is defined to be 1 if there is a sub-tree i rooted at node n and 0 otherwise, and $C(n_1, n_2)$ is the number of the common sub-trees rooted at n_1 and n_2, i.e.,

$$C(n_1, n_2) = \sum_i (I_i(n_1) \cdot I_i(n_2)) \tag{11}$$

Each node n encodes the identity of a sub-tree rooted at n and, if there are more than one node in the tree with the same label, the summation will go over both of them. Therefore, Collins et al. [28] notes that $C(n_1, n_2)$ can be computed in polynomial time, due to the following recursive definition:

1. If the context-free productions at n_1 and n_2 are different, then $C(n_1, n_2) = 0$; otherwise go to 2.

2. If the productions at n_1 and n_2 are the same, and they are both POS tags, then $C(n_1, n_2) = \lambda$; otherwise (i.e., if both n_1 and n_2 are constituent tags, such as NP and S) go to 3.

3. Calculate $C(n_1, n_2)$ recursively as:

$$C(n_1, n_2) = \lambda \prod_{j=1}^{nch(n_1)} (1 + C(ch(n_1, j), ch(n_2, j))) \tag{12}$$

where $nch(n)$ is the number of children of node n, $ch(n, j)$ is the j-th child of node n and $0 < \lambda < 1$ is the decay factor in order to make the kernel value less variable with respect to different sub-tree sizes [28]. An important thing to consider, is that in the above standard CTK, n_1 and n_2 in (12) must have the same number of children, i.e., $nch(n_1) = nch(n_2)$. The addition of the decay factor λ corresponds to a modified kernel:

$$K_{\text{CTK}}(T_1, T_2) = \sum_i \left(\lambda^{size_i} h_i(T_1) h_i(T_2)\right) \tag{13}$$

where $size_i$ is the number of rules in the sub-tree i. In the original convolution kernel ($\lambda = 1$), large matchable sub-trees have much higher influence. According to Collins et al. in [28], if one were to use such a kernel to construct a model which is a linear combination of trees, as is the case with for example SVM, the output would be dominated by the most similar tree. This would cause the model to behave like a nearest neighbour rule. Because of this, the relative importance of the sub-trees are set to scale with their size by introducing the decay factor λ. The decay factor downweights the contribution of sub-trees exponentially with their size.

The convolution kernel by [28] have been successfully applied in many NLP applications [12, 13, 14, 15]. But in [17] and later in [18] Zhou et al. argue that the standard convolution kernel has two shortcomings:

1. All of the sub-trees covered by the convolution kernel are context-free, meaning they don't consider information outside of the specific sub-tree.

2. The convolution kernel may fail to effectively capture the commonality between similar sub-trees because it only allows exact matching of sub-trees.

In [17], Zhou et al. extend the standard CTK with context sensitiveness to overcome 1, and in [18] they further extend the CTK with approximate matching to resolve 2[15].

[15]See [18] for further explanation of the context sensitive and approximate matching CTK

3.3 Performance Measure

As briefly mentioned in the start of Section 3, the F-measure in Equation (1) is generally accepted as the best performance measure for systems within information extraction. The F-measure is just the Harmonic Mean, as the weighted F-Measure just is the Weighted Harmonic Mean, that is, a special case of the (Weighted) Power Means.

One weakness of the F-measure is that it has a degree of andness[16] (as a mean operator) that is fixed to 0.77. Another limitation is that the Weighted F-measure cannot represent recall as more important than the precision or vice versa. In [30], Larsen suggests the Weighted Power Means as a replacement for the F-measure. The Weighted Power Means (WPM) provide a more general and powerful measure, since it allows one to control both the degree of andness and the weighting of the recall and precision.

As a generalized F-measure, the WPM can be written as:

$$F^{\beta}_{\mathrm{WPM}(\alpha)}(R,P) = (\beta R^{\alpha} + (1-\beta)P^{\alpha})^{\frac{1}{\alpha}} \tag{14}$$

where α represents the degree of andness[17] while β and $1-\beta$ are sum-normalized importance weights; i.e., β is the multiplicative importance[29] of the recall (hence, $1-\beta$ is the multiplicative importance of the precision). Assuming that the andness should not be less than 0.5, we have $\alpha \in]-\infty, 1]$.

In domains where the precision or recall is of much greater importance, the generalized F-measure could prove to be of more use than the F-measure we know of today. Also, the unweighted WPM is retained for $\beta = \frac{1}{2}$ by which we at $\alpha = -1$ further retain the F-measure as in Equation (1).

[16]Andness is the degree to which the operator aggregates like the minimum (logic AND) rather than the maximum (logic OR)[29]

[17]$\alpha \to -\infty : 1$, $\alpha = -1 : 0.77$, $\alpha = 0 : 0.66$, $\alpha = 1 : 0.5$, $\alpha \to +\infty : 0$

Table 1: Set of predefined relations and their contraints on the entity types

Relation	Arg_1	Arg_2	Example
Located_In	LOC	LOC	(Toledo, Ohio)
Work_For	PER	ORG	(Steven Bryen, Pentagon)
OrgBased_In	ORG	LOC	(Pentagon, U.S.)
Live_In	PER	LOC	(LeRoy, New York)
Kill	PER	PER	(Jack Ruby, Lee Harvey Oswald)

4 My Relation Extraction Approach

After studying the related work and their state-of-the-art theory, I consider the problem of automatically identifying predefined types of relations between named entities in text documents. Formally, given a sentence S, which consists of a sequence of words and entities $\{E_1, E_2, ..., E_N\}$ indexed according to their order, a binary relation between the entities E_i and E_j is represented as a pair $R_{ij} = (E_i, E_j)$, where E_i and E_j are the first and second argument, respectively. I denote the set of predefined entity and relation types as C_E and C_R, respectively.

Five different types of binary relations are present in the corpus I will be doing my experiments on: *Kill, Located_In, Work_For, OrgBased_In* and *Live_In*, where the named entities are of type *person, location* and *organization.* Table 1 illustrates the set of predefined relations and their entity type constraints[18]. Note that the relations are directed, i.e. $R_{ij} \neq R_{ji}$ since the entities each have their own role. Consider Figure 7 for an example on a sentence from the dataset[19]. There are five named entity mentions, E_1 is of type *person* while E_2, E_5 are of type *location* and E_3, E_4 are of type *organization.* Among the 20 possible directed relations, $R_{12} = (Miller, Wilmington)$ are of type *Live_In* and R_{45} = (*Bringham Young University, Utah*) are of type *OrgBased_In* while the other 18 are of type *no_rel.*

As is the case in most of the related work, I treat relation extraction as a supervised learning problem. For the classification task, a crucial issue concerns how I can generate a dataset suitable to train a classifier from a corpus of annotated sentences as the one shown in Figure 7.

[18]In addition to the relations in Table 1, C_R contains a special element *no_rel* which represents the absence of a relation of interest

[19]The dataset used is the CoNLL'04 which will be further discussed later in this section

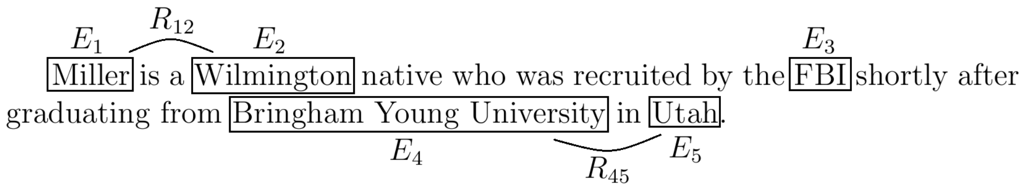

E_1
R_{12}
E_2
E_3
Miller is a Wilmington native who was recruited by the FBI shortly after graduating from Bringham Young University in Utah.
E_4
R_{45}
E_5

Figure 7: A sentence with two relations, R_{12} and R_{45}, between four entities of type *person*: E_1, *location*: E_2, E_5 and *organization*: E_4.

4.1 Dataset

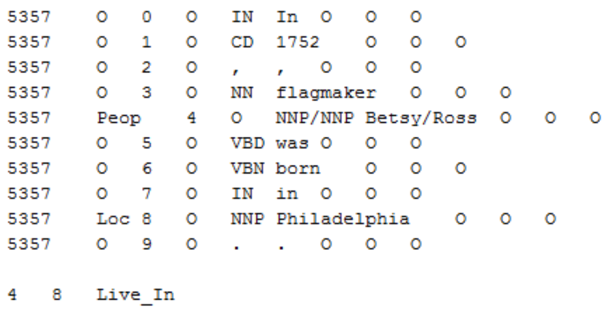
In this section I will describe a generative procedure that allows me to reduce the dataset size and to balance its distribution making the classification task easier[20]

Initially, some assumptions are made about the input data. First, as for most relation extraction datasets, the entities are already recognized and given to us as input. Second, only relations between entities within the same sentence are considered; while there may be relations between entities in different sentences, they are not annotated in the corpus used for evaluation. Especially this assumption significantly limits the size of the dataset. Third, self-relations (i.e. R_{ii}) are not considered as they are not annotated in the corpus. Finally, each relation type is learned independently (i.e. each relation is trained and tested on a distinct dataset).

Given an annotated sentence S, as the one shown in Figure 7, let E be the set of N entities in S. The simplest way to generate examples to train a classifier for a specific relation R is to enumerate all possible ordered pairs of entities $(E_i, E_j \in E, 1 \leq i, j \leq N, i \neq j)$ in S. These pairs will be referred to as *candidate entities* for the relation R. In particular, each example is the copy of the original sentence S represented by the shortest path-enclosed tree between the candidate entities. The candidate entities are assigned $AGENT$ and $TARGET$ attributes according to the roles taken in the relation example, while entities not involved are ignored. If the relation holds between the two candidate entities, then the example is labelled 1, otherwise, it is labelled -1. This approach simplifies the relation extraction into a binary classification task.

As an example consider the sentence in Figure 7. This sentence would generate 20 examples, (i.e., all permutations of five entities taken two at a time). However this strategy would yield a strongly unbalanced dataset (i.e,

[20]Learning with skewed class distributions is a well known problem in machine learning. Studies have shown that an imbalanced class distribution leads to poor performance on the minority class[31]



```
5357    O    0    O    IN   In   O    O    O
5357    O    1    O    CD   1752      O    O    O
5357    O    2    O    ,    ,    O    O    O
5357    O    3    O    NN   flagmaker    O    O    O
5357    Peop      4    O    NNP/NNP Betsy/Ross   O    O    O
5357    O    5    O    VBD was O    O    O
5357    O    6    O    VBN born      O    O    O
5357    O    7    O    IN   in   O    O    O
5357    Loc 8     O    NNP Philadelphia     O    O    O
5357    O    9    O    .    .    O    O    O

4   8   Live_In
```

Figure 8: Example of a sentence block in the CoNLL'04 Relation Recognition Corpus, col-2: Entity class label, col-3: Element order number, col-5: POS tags, col-6: Word

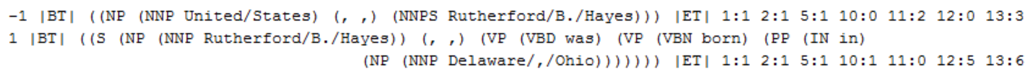
with a low density of positive relation examples). Among the 20 instances, only 2 are positive examples of *Live_In* and *OrgBased_In*, respectively. Furthermore, the resulting dataset would contain pairs of similar examples, differing only in the assigned attributes, but with different classification labels (e.g., (*Miller, Wilmington*)=1 and (*Wilmington, Miller*) = -1).

To overcome these problems, I define a different way of generating the examples, that is, I enumerate the candidate entities without taking into account the order of the entities (e.g., $(E_i, E_j) = (E_j, E_i)$). However, to avoid wrong classifications, each example is assigned an attribute representing the order of the candidate entities according to the relation R (i.e., *WRONG* or *CORRECT* order of entities). Note that this allows me to halve the size of the dataset and prevent the generation of misleading (negative) examples (e.g., in the example (*Wilmington, Miller*) = -1), thereby implicitly undersampling the dataset without losing positive examples. This mitigates the problem of the low density of positive examples.

In an attempt to further reduce the dataset size, I generate only those examples whose candidate entities satisfy the argument type constraints of R, that is, different relation classifiers are trained on different datasets for different relation types. In Section 5, I compare the impact of these reductions.

4.1.1 CoNLL

The dataset used is a corpus generated for the relation extraction task as a part of the Conference on Computational Natural Language Learning in 2004[20]. As mentioned in the previous section, I assume the entities are already given, as they are marked in the corpus. Besides the entities, additional features such as POS tag, and word number is also provided. Consider



```
-1 |BT| ((NP (NNP United/States) (, ,) (NNPS Rutherford/B./Hayes))) |ET| 1:1 2:1 5:1 10:0 11:2 12:0 13:3
1 |BT| ((S (NP (NNP Rutherford/B./Hayes)) (, ,) (VP (VBD was) (VP (VBN born) (PP (IN in)
                                    (NP (NNP Delaware/,/Ohio))))))) |ET| 1:1 2:1 5:1 10:1 11:0 12:5 13:6
```

Figure 9: Example output from the parser on the sentence *"In 1822, the 19th president of the United States, Rutherford B. Hayes, was born in Delaware, Ohio."* for the relation *Live_In*

Figure 8 for an example from the corpus.

4.2 Parsing

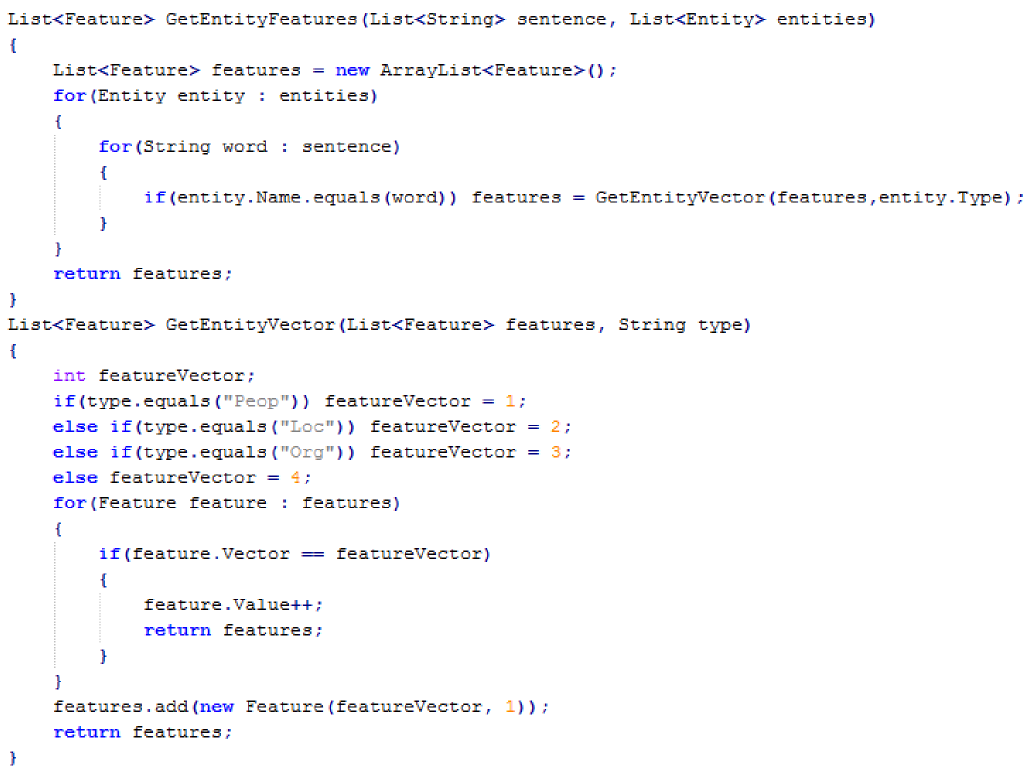
As briefly mentioned in Section 4.1, the parse trees generated for each pair of candidate entities for a relation R are based on the Shortest Path-enclosed Tree (SPT) structure, see Figure 1(b) for an example. While there have been suggested improvements to the SPT, the complexity of these extensions do not match the estimated gain in performance, and because of this they are dismissed in my approach. Instead I combine the SPT with a few simple but efficient features, these features include:

1. Entity types
2. Potential relation types
3. Order of entities
4. Index of first argument in the relation
5. Index of second argument in the relation
6. Distance between the candidate entities

While very simple, these features prove to enhance the performance of the relation extraction task significantly. In Section 5, I do experiments with and without these features.

4.2.1 Berkeley Parser

By rewriting the PCFGLA.BerkeleyParser[21], I adapt the parser to work with the CoNLL'04 corpus. For each pair of candidate entities, the parser checks for a possible relation and generates the dataset as described in Section 4. The parser adopts a hierarchical split-and-merge strategy[32] to enrich basic non-terminal tags with latent annotations as described in Section 3.1.2. In the experiments in Section 5, I compare results with and without latent annotations.



```
List<Feature> GetEntityFeatures(List<String> sentence, List<Entity> entities)
{
    List<Feature> features = new ArrayList<Feature>();
    for(Entity entity : entities)
    {
        for(String word : sentence)
        {
            if(entity.Name.equals(word)) features = GetEntityVector(features,entity.Type);
        }
    }
    return features;
}
List<Feature> GetEntityVector(List<Feature> features, String type)
{
    int featureVector;
    if(type.equals("Peop")) featureVector = 1;
    else if(type.equals("Loc")) featureVector = 2;
    else if(type.equals("Org")) featureVector = 3;
    else featureVector = 4;
    for(Feature feature : features)
    {
        if(feature.Vector == featureVector)
        {
            feature.Value++;
            return features;
        }
    }
    features.add(new Feature(featureVector, 1));
    return features;
}
```

Figure 10: Taking as input the SPT for a pair of candidate entities and it's related entities, the algorithm generates the entity specific feature vectors

Additionally, I integrate a module for generating the feature vectors directly from the CoNLL'04 corpus.

4.2.2 Feature Vectors

The generation of feature vectors is separated into two categories, entity specific and relation specific. Entity specific features are generated by the algorithm in Figure 10. Taking as input the SPT for a pair of candidate entities and it's related entities, the algorithm generates the entity specific feature vectors. While very simple, these features are necessary for the classifier to recognize the entities.

For relation specific features, the candidate entities are essential. Based on the type of the candidate entities, the system is able to determine the order of the entities for the relation (i.e., order of entities is *WRONG* in *"Computerfriend hired Henrik"* for the *Work_For* relation according to the rules in Table 1). Based on the order of the entities, the feature vectors for first and second argument as well as the distance between them can be computed directly.

4.3 Learning

Once the dataset for each of the five relations have been generated by the parser, the dataset is split into a training set and a testing set. In the CoNLL'04[20] corpus, there are 1,441 sentences with 5,349 entities that contain at least one active relation. Of the entities, 1,691 are people, 1,968 are location, 984 are organizations, and 706 are miscellaneous names. Possible pairs of candidate entities (binary relations) are 19,080 specifically, 405 *Located_In*, 401 *Work_For*, 452 *OrgBased_In*, 521 *Live_In*, 268 *Kill*, and 17,033 *no_rel*. Note that the relation *no_rel* significantly outnumbers all the others because there are no active relations at all between most pairs of candidate entities. Consider Table 1 for examples of each relation type and the constraints between a relation and its two entity arguments.

Of the 1,441 sentences, 91% (1,310) are used for training while 9% (131) are used for testing. For classifying the relations, I implement a composite kernel consisting of a linear kernel[21] for the feature space described in Section 4.2, and a convolution tree kernel as the one described in Section 3.2.3.

The kernels are combined using *sequential summation* i.e., the kernels between corresponding pairs of trees and vectors in the input sequence are summed together:

$$K_s(o_1, o_2) = \tau \times K_t(T_1, T_2) + K_v(\mathbf{v}_1, \mathbf{v}_2) \tag{15}$$

where τ rules the contribution of the tree kernel K_t with respect to the feature vector kernel K_v.

4.3.1 SVM-Light-TK

The composite kernel described above is implemented using the open-source software SVM-Light-TK[33]. SVM-Light-TK (SVMLTK) is an extension to the well known open source implementation of SVMs in C, SVM^{light}[34].

As any machine learning algorithm, the SVMLTK is separated into two modules, specifically *svm_learn* and *svm_classify*. Using Equation (15), the *svm_learn* module builds a model composed by support vectors generated from the examples in the training set. Taking the model as input, the *svm_classify* classifies the test set accordingly.

Originally, the *svm_classify* generates no applicable output except the result of the classification (i.e., *precision/recall*). Because of this, the

[21]Linear kernel is described in Section 3.2.2

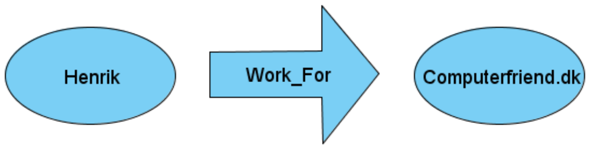


Figure 11: Resource Description Framework triple - Subject (Henrik), Predicate (Work_For), and Object (Computerfriend)

svm_classify is rewritten to generate a more interpretable output. I integrate a module that stores the result of each classification together with the pair of candidate entities using the Resource Description Framework (RDF) data model [35].

4.3.2 Resource Description Framework

The RDF data model is similar to classic conceptual modeling approaches such as entity-relationship or class diagrams, as it is based upon the idea of making statements about resources in the form of subject-predicate-object expressions[22]. For example, one way to represent the *Work_For* relation in the sentence: "Henrik works for Computerfriend.dk" in RDF is as the triple: a subject denoting "Henrik", a predicate denoting "*Work_For*", and an object denoting "Computerfriend.dk". Also consider Figure 11 for a visual illustration of the triple.

RDF has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources. Using RDF in the relation extraction system accommodates possible post processing of the classification. In Section 6, a post processing prototype for gathering information about a specific entity is suggested.

[22]The subject-predicate-object expression is known as *triples* in RDF terminology

Table 2: Performance of tree kernel-based relation extraction

Kill			Live_In			Located_In		
P	R	F1	P	R	F1	P	R	F1
77.8	41.2	53.9	84.6	20.8	33.3	84.6	34.4	48.9

OrgBased_In			Work_For		
P	R	F1	P	R	F1
80.0	38.7	52.2	93.8	37.5	53.6

5 Experiments

In this section, experiments conducted to evaluate different relation extraction methods are described. Additionally, the final system is evaluated and compared to earlier approaches. The objective of these experiments were to investigate the effectiveness of the different relation extraction methods, and optimize my solution to produce the best possible results.

Using the dataset described in Section 4.1, and later in 4.3, the experiments leading up to the final solution were:

1. Relation extraction through the use of a subset tree kernel on the shortest path-enclosed tree of the candidate entities as described in Section 3.2.3.

2. Addition of linear kernel for entity and relation specific features as described in Section 3.2.2.

3. Extending the SPT with context sensitiveness (CSPT) as described in Section 3.1.

4. Extending with latent annotations as described in Section 3.1.2.

5. Further reduction of the dataset as described in Section 4.1.

6. Context sensitive SPT with the composite kernel on the reduced dataset.

5.1 Tree Kernel-based Relation Extraction

Initially, the first thought was to extract relations through the use of a tree kernel exclusively, K_T. Using the theory of the convolution tree kernel as described in Section 3.2.3, and the two open source programs[23], this was

[23]Berkeley Parser[21] and SVM^{light}-TK[33]

Table 3: Performance of composite kernel-based relation extraction

Approach	Kill			Live_In			Located_In		
	P	R	F1	P	R	F1	P	R	F1
K_C	60.9	82.4	70.0	70.0	52.8	60.2	85.7	75.0	80.0
K_{CS}	57.7	88.2	69.8	69.8	56.6	**62.5**	78.8	83.9	**81.3**
K_{LA}	70.6	70.6	70.6	68.6	45.3	54.6	81.5	68.8	74.6
K_{Cr}	72.2	76.5	74.3	68.6	45.3	54.6	85.7	75.0	80.0
K_{CSr}	93.3	82.4	**87.5**	76.7	43.4	55.4	84.0	67.7	75.0

Approach	OrgBased_In			Work_For		
	P	R	F1	P	R	F1
K_C	74.3	83.9	78.8	73.6	97.5	**83.9**
K_{CS}	74.3	83.9	78.8	70.6	94.7	80.9
K_{LA}	80.0	77.4	78.7	72.9	87.5	79.6
K_{Cr}	76.5	83.4	**80.0**	73.6	97.5	**83.9**
K_{CSr}	76.5	83.4	**80.0**	59.7	97.4	74.0

accomplished. Unfortunately, as can be seen in Table 2, the results were very poor. This approach suffered the same issues as early kernel-based relation extraction approaches - high precision, but low recall. This is caused by the nature of the tree kernel as mentioned in Section 2, matching only nodes at the same layer and in the identical path starting from the roots to the current nodes, which is a very strong constraint.

5.2 Composite Kernel-based Relation Extraction

To increase the recall of the system, the tree kernel is further extended to a composite kernel K_C by summing together the results of a linear kernel of the entity and relation specific features generated in Section 4.2.2. As can be seen in Table 3, this significantly improved the performance on especially the *Located_In, OrgBased_In*, and the *Work_For* relations.

Unfortunately, the recall of the *Live_In* and *Kill* relations were still low. The *Live_In* relation often span over a larger amount of words, which causes the strong constraints of the tree kernel to significantly lower the recall. Of the five relation types, the *Kill* relation has a significantly lower amount of actual relations, and as mentioned in Section 4.1, skewed class distribution often leads to poor performance on the minority class (i.e., positive *Kill* relation).

5.3 Context sensitive SPT

In an attempt to improve the performance of the relation extraction, the SPT generated for each pair of candidate entities is extended by the first word on the left and right side of the first and second entity, respectively. The performance of the composite kernel on the context sensitive SPT (K_{CS}) can be seen in Table 3. Argued earlier by Zhang et al. [15], improper inclusion of additional structural information, in most cases, only introduce more "noise" to the system. From the results obtained during this experiment, one can acknowledge that statement, as the inclusion of context sensitiveness produces no visible improvement to the performance of the system. Unfortunately, the dynamic context sensitive SPT proposed by Zhou et al. [17] was not realized in my solution.

5.4 SPT and Latent Annotations

Proposed by Zhou et al. [18], latent annotations are employed on the basic non-terminal nodes, such as NP, NNP etc (K_{LA}). Unfortunately, as can be seen from the results in Table 3, the latent annotations affects the performance negatively, reducing the recall of the system while only slightly improving precision. If one considers the nature of latent annotations, this result is inevitable. While matching of parse trees is already suffering from very strong constraints, adding another constraint is likely to make it even worse, as it also shows in the experiment.

5.5 Reduced Dataset

As described in Section 4.1, the dataset for each relation R is reduced to only contain those pair of candidate entities that satisfy the argument type constraints of R (K_{Cr}) (i.e., only person-person candidate entities are considered for the relation *Kill*). This will greatly reduce the dataset and create a more even class distribution, possibly improving performance.

As can be seen from the results in Table 3, the performance on especially the *Kill* relation increased significantly. As mentioned earlier in Section 5.2, the *Kill* relation has a very skewed class distribution in the large dataset as there are less than half as many *Kill* relations as any other relation type. Unfortunately, the reduction in recall for the *Live_In* relation causes the K_{Cr} to only be equal in performance with the K_C.

5.6 Context Sensitive SPT on Reduced Dataset

Finally, I combine the context sensitive SPT from the experiment in Section 5.3 with the reduced dataset from the experiment in Section 5.5 (K_{CSr}). Evaluating the results in Table 3, the *Kill* relation experienced significantly increase in performance. This is both because of the previous described skewed class distribution which is avoided by reducing the dataset, and the fact that some of the kill relations have critical verbs just before or after the candidate entities. Unfortunately the context sensitive SPT causes especially the *Work_For* classifier to produce too many false positives, significantly reducing the F1.

In the discussion in Section 7, I consider the benefits of the different approaches and talk about the limitations made in regards to the state-of-the-art.

6 Post Processing

In previous relation extraction approaches, researchers have primarily been focusing on improving the performance of the extraction and classification of the relations, but as the state-of-the-art approaches are beginning to produce better and more useful results, I will be looking into the possibility of post processing the output from the relation extraction.

As mentioned in Section 4.3, the classifier is rewritten to produce an output using the RDF data model as described in Section 4.3.2, e.g.:

Prediction: [”Sirhan B. Sirhan”,”Kennedy”, kill]

As it is, this output is still not very consumer friendly, and with the possibility of hundreds of such relations being marked, obtaining an overview of the extracted information will be very difficult. I create a prototype for extracting information about one specific entity, allowing the user to query the system asking for information about one specific entity (e.g., Kennedy as in the example above). For each extracted relation, the algorithm scans the entities in the search for information, i.e.:

```
if $firstMention == entity$ then
    return $\{relation, secondMention\}$
else
    if $secondMention == entity$ then
        return $\{firstMention, relation\}$
    end if
end if
```

Using this algorithm, the system can be queried for an entity and return any extracted information. While simple, this process is very helpful in gathering information about a specific entity, and can greatly reduce the workload on the person analysing the text.

For gathering information about a larger topic, one might want to still gather all the extracted relations. Using the RDF data model, one possibility would be to combine all the relations into a social network graph, representing the different entities and their relations.

Table 4: Comparison to the state-of-the-art approach by Giuliano et. al (K_{SL}) [16]

Approach	Kill			Live_In			Located_In		
	P	R	F1	P	R	F1	P	R	F1
K_C	60.9	82.4	70.0	70.0	52.8	60.2	85.7	75.0	**80.0**
K_{SL}	82.2	81.0	**81.9**	78.0	65.8	**71.4**	79.6	76.0	77.8

Approach	OrgBased_In			Work_For		
	P	R	F1	P	R	F1
K_C	74.3	83.9	**78.8**	73.6	97.5	**83.9**
K_{SL}	74.3	77.2	75.7	76.8	80.0	78.4

7 Discussion

After analysing the theory of the state-of-the-art relation extraction approaches, I did experiments with different kernels and parse tree structures as described in Section 5. While initially expecting the state-of-the-art methodologies to improve performance, on the contrary experiments showed that for example latent annotations[18] would only act as an even stronger constraint on the system, lowering recall significantly in comparison to the slightly, if any, increase in precision. Taking into account the advanced complexity and questionable increase in performance of the extensions Zhou et al. proposed, additional implementation of those extensions were dismissed.

Instead, in an attempt to improve performance, the tree kernel was combined with the simple but significant feature kernel, which in the end proved to perform the best overall performance. As earlier argued by Zhou et al. in [17], a context-sensitive parse tree structure would only introduce more noise without proper administration, which also is what the experiments showed.

In [16], Giuliano et al. argued that reducing the dataset[24] would help to remove skewed class distributions and because of that, improve performance of the system. But as can be seen from the experiments, this was only the case for some of the relation types, again ending with a lower overall performance compared to the default composite kernel[25].

Of course here the performance is measured by the generally accepted performance measure F-measure, if one were to weigh precision or recall higher the outcome might be different using the generalized F-measure as described in Section 3.3.

[24]Reducing the dataset by generating different smaller datasets for each relation type only containing valid pair of candidate entities

[25]Composite kernel, i.e. results of tree kernel and feature kernel summed together

The previous state-of-the-art approach on the dataset used in this report was made by Giuliano et. a. in [16], in Table 4 a comparison can be seen. While the approach in this report achieves better results on the *Located_In*, *OrgBased_In* and *Work_For* relations, Giuliano et al. gets significantly better results on the *Kill* and *Live_In* relations, which leads to a better overall performance.

The approach by Giuliano et al. uses a global context kernel where each sentence is represented using the *bag-of-words* structure instead of subsequences of words, which has been shown to perform better on longer sentences as is the case for the *Kill* and especially *Live_In* relation. This is because the *bag-of-words* doesn't take into account the syntax and structure of the sentence, which gets more difficult to match the longer the sentence is.

After having studied and worked with the task of relation extraction, I have realized that an F-measure of approximately 80% is from my point of view enough to be useful in many domains, with the possibility of adapting a system to produce a much greater precision or recall by refining some of the classification parameters.

Because of this, a more interesting approach for further study in relation extraction would be towards the area of application. Where can the relation extraction be applied? Earlier in the introduction, I mentioned question answering and text summarization for possible applications of the relation extraction. However, using the RDF data model as described in Section 4.3.2, there are many other domains where the application of relation extraction would prove useful.

As briefly mentioned in Section 6, one possible domain is social network analysis (SNA). The application of relation extraction would add another dimension to SNA, instead of simply representing links between entities, the type of links, or in this case relationships, would also be visible.

8 Conclusion and Future Work

In this report, I have presented my approach on the relation extraction task. I proposed a linear combination of a tree kernel and a feature kernel. Applying a CTK on the SPT for a pair of candidate entities, combined with the results of a feature-based kernel, the goal was to gain an increase in performance compared to earlier approaches in the related work. Unfortunately, the final performance of the system only achieved close to same performance as the previous state-of-the-art approach on the CoNLL'04 dataset. However, as my approach produces an output using the RDF data model, it allows for easier application.

For future work, I would like to further study the application of relation extraction, as the performance of the relation extraction task for some domains is reliable enough to be very useful.

9 References

[1] Message Understanding Conference Proceedings (MUC-7), http://www-nlpir.nist.gov/related_projects/muc/proceedings/muc_7_toc.html (24 May 2012), 2001.

[2] Conference on Computational Natural Language Learning (CoNLL), http://www.cnts.ua.ac.be/conll/ (24 May 2012), 2011.

[3] Automatic Content Extraction (ACE), http://www.itl.nist.gov/iad/mig/tests/ace/ (24 May 2012), 2009.

[4] Automatic Content Extraction & Linguistic Data Consortium (ACE), http://projects.ldc.upenn.edu/ace/ (24 May 2012).

[5] G.-D. Zhou and J. Su, "Named entity recognition using an hmm-based chunk tagger," in *Proceedings of of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, pp. 473–480, 2002.

[6] Message Understanding Conference 6 (MUC-6), http://www.cs.nyu.edu/cs/faculty/grishman/muc6.html (24 May 2012), 1996.

[7] N. Kambhatla, "Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations," in *Proceedings of the 42nd Annual Meeting of the Association of Computational Linguistics (ACL'04)*, pp. 178–181, 2004.

[8] G.-D. Zhou, J. Su, J. Zhang, and M. Zhang, "Exploring various knowledge in relation extraction," in *Proceedings of the 43rd Annual Meeting of the Association of Computational Linguistics (ACL'05)*, pp. 427–434, 2005.

[9] D. Roth and W.-t. Yih, "A linear programming formulation for global inference in natural language tasks," in *Proceedings of CoNLL '04*, pp. 1–8, 2004.

[10] J. Jiang and C. Zhai, "A systematic exploration of the feature space for relation extraction," in *Proceedings of Human Language Technologies: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT07*, pp. 113–120, 2007.

[11] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[12] D. Zelenko, C. Aone, and A. Richardella, "Kernel methods for relation extraction," *The Journal of Machine Learning Research*, pp. 1083–1106, 2003.

[13] A. Culotta and J. Sorensen, "Dependency tree kernels for relation extraction," in *Proceedings of the 42nd Annual Meeting of the Association of Computational Linguistics (ACL'04)*, pp. 423–439, 2004.

[14] R. Bunescu and R. J. Mooney, "A shortest path dependency kernel for relation extraction," in *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP'05)*, pp. 724–731, 2005.

[15] M. Zhang, J. Zhang, J. Su, and G.-D. Zhou, "A composite kernel to extract relations between entities with both flat and structured features," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association of Computational Linguistics (ACL'06)*, pp. 825–832, 2006.

[16] C. Giuliano, A. Lavelli, and L. Romano, "Relation extraction and the influence of automatic named-entity recognition," *ACM Transaction on Speech and Language Processing*, vol. 5, pp. 2:1–2:26, 2007.

[17] G.-D. Zhou, M. Zhang, D. Ji, and Q.-M. Zhu, "Tree kernel-based relation extraction with context-sensitive structured parse tree information," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07*, pp. 728–736, 2007.

[18] G.-D. Zhou and Q.-M. Zhu, "Kernel-based semantic relation detection and classification via enriched parse tree structure," *Journal of Computer Science and Technology*, vol. 26, pp. 45–56, 2011.

[19] G.-D. Zhou, L.-H. Qian, and Q.-M. Zhu, "Label propagation via bootstrapped support vectors for semantic relation extraction between named entities," *Computer Speech and Language*, vol. 23, pp. 464–478, 2009.

[20] CoNLL Corpus for Entity and Relation Recognition Experiments, http://cogcomp.cs.illinois.edu/Data/ER/conll04.corp (24 May 2012), 2004.

[21] The Berkeley parser, http://nlp.cs.berkeley.edu/Historical.shtml (24 May 2012), 2009.

[22] T. Mitchell, *Machine Learning.* McGraw Hill, 1997.

[23] Inductive inference, http://en.wikipedia.org/wiki/Inductive_inference (24 May 2012), (2012).

[24] Support Vector Machine, http://en.wikipedia.org/wiki/Support_vector_machine (24 May 2012), (2012).

[25] Maximum-margin hyperplanes, http://en.wikipedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png (24 May 2012), (2012).

[26] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *Journal of Machine Learning Research*, pp. 419–444, 2002.

[27] D. Haussler, "Convolution kernels on discrete structures," tech. rep., University of California, Santa Cruzca, USA, 1999.

[28] M. Collins and N. Duffy, "Convolution kernels for natural language," in *Advances in Neural Information Processing Systems 14*, pp. 625–632, 2001.

[29] H. Legind Larsen, "Importance weighting and andness control in de morgan dual power means and owa operators," *Fuzzy Sets and Systems*, vol. 196, pp. 17–32, 2012.

[30] H. Larsen, "Generalized F-measure for combined recall-precision performance measures", Research note, 2012.

[31] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special issue on learning from imbalanced data sets," *SIGKDD Explorer newsletter*, vol. 6, pp. 1–6, 2004.

[32] S. Petrov, L. Barrett, R. Thibaux, and D. Klein, "Learning accurate, compact, and interpretable tree annotation," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Assocation of Computational Linguistics (ACL'06)*, pp. 433–440, 2006.

[33] SVM-LIGHT-TK 1.2, http://disi.unitn.it/moschitti/Tree-Kernel.htm (24 May 2012).

[34] SVM_light, http://svmlight.joachims.org/ (24 May 2012), 2008.

[35] Resource Description Framework, http://www.w3.org/RDF/ (24 May 2012), (2012).