Technische Universität Berlin, Fakultät I – Geisteswissenschaften
Institut für Sprache und Kommunikation
Fachbereich Anglistische und Allgemeine Linguistik

Peter Palaga

Extracting Relations from Biomedical Texts Using Syntactic Information

Magister Thesis

2009-04-13

Author: Peter Palaga, Matrikelnummer: 228182
Supervisors: Prof. Dr. See-Young Cho, Prof. Dr. Ulf Leser

Table of Contents

Sebständigkeitserklärung

Die selbständige Anfertigung versichere ich an Eides Statt.

Berlin, den

.....

Peter Palaga

Zusammenfassung

Relationsextraktion ist ein Verfahren, mit dem Informationen gezielt aus Fließtext in einer strukturierten Form gewonnen werden. Eine heutzutage sehr häufig untersuchte Anwendung der Relationsexktraktion ist die Extraktion von Protein-Protein Interaktionen (PPI) aus biomedizinischen Texten. Wir haben mehrere automatische kernelbasierte Methoden für die Relationextraktion auf dem PPI-Extraktion Problem evaluiert. Einerseits haben wir Experimente mit in der Fachliteratur beschriebenen Kernelfunkionen durchgeführt und andererseits haben wir einige Kernels selbst entworfen. Wir haben Methoden und Metriken für die Evaluierung benutzt, die einen direkten Vergleich mit den PPI-Extraktionsmethoden erlauben, die “auf dem neusten Stand der Kunst” sind. Ein von uns entworfener Kernel hat höhere Genauigkeit und deutlich kürzere Laufzeiten gezeigt als die Kernels aus der Literatur gezeigt haben. Im Vergleich mit den modernsten Methoden hat jedoch unsere Methode schlechtere Ergebnisse erreicht.

Acknowledgments

My special thanks go to Lenka, my life partner, for her patience and willingness to carry on her own most of our fresh parental duties while I was busy working on this thesis. Also, I would like to thank Prof. Ulf Leser for employing me in Alibaba project at WBI two years ago and so encouraging me to apply my qualification in practical situations and especially for his patient assistance from the beginnings of this thesis to its very end. Moreover, I would like to thank the following people for their special contributions: Many thanks to Prof. See-Young Cho for his assistance. Many thanks to Antti Airola and Sampo Pyysalo from University of Turku, for their methodological hints and for corpus splits used in k-fold cross-validation. Many thanks to Tetsuji Kuboyama from University of Tokyo for providing his spectrum tree kernel implementation. Many thanks to Alessandro Moschitti from University of Trento for sending me the newest version of his tree kernel software. Many thanks to Eugene Charniak, Mark Johnson and David McCloski from Brown University for helping me with their parser.

Der Sachverhalt ist eine Verbindung von Gegenständen (Sachen, Dingen).
Ludwig Wittgenstein (1921)

1 Introduction

Well-formed sentences of natural language usually state facts. Intuitively, such facts can be seen as consisting of two different types of items: firstly, there are some things and secondly there are some associations that are claimed to hold between that things. In the field of natural language processing (NLP), such things are called *entities* and the associations between them are called *relations*. Consider the following sentence as an example:

Alice spends more money on eBay than Bob.

In this sentence, (at least) two entities were mentioned, namely persons Alice and Bob. Furthermore, the sentence asserts that there is a particular relation between them: the former spends more money on eBay then the later.

To state which relations hold between which entities is a typical task of researchers in natural sciences. So, biologists are observing cells to say (among other things) which genes cause which diseases, which genes inhibit other genes or which proteins interact with which other proteins (Protein-Protein Interactions, PPI).

Scientists mostly publish their findings in form of free text, e.g. papers, books, etc. With raising number of known facts, it becomes necessary to systematize the knowledge in a more structured form, e.g. a database. Structured data sources allow richer queries than free text. For example, it is not effectively possible to find out which genes are inhibited by gene SOX9 only using free text queries on a big collection of scientific text.

Indeed, there exist attempts to create databases storing biomedical entities and their relations by hand; e.g Database of Interacting Proteins (DIP, Xenarios et al., 2000), the Biomolecular Interaction Network Database (BIND, Bader et al., 2001) and the Molecular INTeraction database (MINT, Zanzoni et al., 2002). However, such projects are far from being complete (see e.g. Chatr-aryamontri et al., 2006).

It is a task of Information Extraction to develop methods for creating such databases automatically. This task is commonly seen as having three phases: (1) entity recognition (2) coreference resolution and (3) relation extraction (Culotta and Sorensen, 2004). It is precisely

the last of these phases which will be focused in this thesis.

Several approaches have been applied to extract relations from free text. The simplest method is based on the assumption that a mere co-occurrence of two entities implies that there is a relation between them. Trivially, such methods reach 100% recall, but their precision stays low (Pyysalo et al., 2008; see also Section 5.1 for more details on precision and recall).

Pattern and rule based methods try to use context information for finding relations between entities. They usually look for certain words occurring near entity names or use part-of-speech (POS) and/or syntax information. They usually exhibit high precision, but their recall is low, i.e. many of the relations in the text are left undiscovered by them. The patterns used by such approaches may be constructed by hand or learned automatically from an annotated corpus (Hakenberg et al., 2005; Fundel et al., 2007, Blaschke, 1999).

In this thesis, the machine learning approach will be pursued. A statistical classifier will be used to predict the presence or absence of a relation between a given pair of entities in a sample sentence. The decisions of such a classifier rest upon a statistical model which is produced by a training on a text corpus containing positive and negative examples (Donaldson et al., 2003).

Several machine learning techniques have been used in the field of relation extraction, among others nearest neighbor (Fukunaga, 1990), naïve Bayes, sparse regularized least-squares (RLS) (Airola et al., 2008) and support vector machines (SVM) (Cortes and Vapnik, 1995; Culotta and Sorensen, 2004). SVM, introduced in Chapter 2, will be used in this thesis. In Chapter 2 we also introduce the notion of kernel function which plays a central role in our survey.

The chosen method presupposes a resource with annotated entities and relevant relations on which the classifier will be trained. To this end, corpora prepared by Pyysalo et al. (2008) with annotated Protein-Protein Interactions (PPI) will be used. We describe these corpora in Chapter 3.

Further, in Chapter 4, we present several kernel functions known from the literature. We also document our own attempts to design a novel kernel suitable for relation extraction.

We have evaluated the kernel functions introduced Chapter 4 using methods and metrics presented in Chapter 5.

The details of our experimental setup are explained in Chapter 6, while our experimental results are presented in Chapter 7.

The contribution of this thesis to the field of relation extraction and especially to PPI extraction can be seen as threefold: Firstly, we apply the evaluation method proposed recently by Airola et al. (2008) to several kernel functions known from literature. In this way, we provide benchmarks which allow for direct comparability of these kernels with each other on one hand and with state-of-the-art methods on the other hand. Secondly, these kernels – to our best knowledge – have not been evaluated on an relation extraction task yet. Thirdly, we propose a new kernel method for relation extraction and evaluate it so that its performance is directly comparable with both the state-of-the-art methods and other methods evaluated in this thesis.

2 SVM Classifier

Generally, the machine learning techniques have two phases: (1) learning (also called "training") and (2) the actual application of the technique on new data instances. To train a classifier one needs a set of training examples. This set needs to be big enough, so that the classifier is able to learn some sort of general model, which works also for new, previously unseen examples.

Within this thesis, Support Vector Machine (SVM) is used as a classifier (Vapnik, 1995).

Internally in the SVM, training examples are represented as *feature vectors*. Feature vectors are ordered *n*-tuples of numeric values. To illustrate, how an SVM works, let us take the special case of ordered 2-tuples, i.e. pairs. The pairs have the apparent advantage of being interpretable as coordinates of points in a two-dimensional space. In this way the task of classification can be nicely visualized in a diagram.

Example 2.1: Suppose, we have the following set of examples for the learning phase:

+[5, 3], +[7, 4], +[7,6], –[2, 5], –[2, 6], –[2, 8], –[4, 8]

Note, that each of the examples is labeled with "+" or "–", which mean, that the given example is positive (i.e. bearing the relation) or negative (not bearing the relation) respectively. So we have seven feature vectors of length 2, three of them being positive and four of them negative. The diagrammatic representation of these examples can be seen in Figure 2.1.

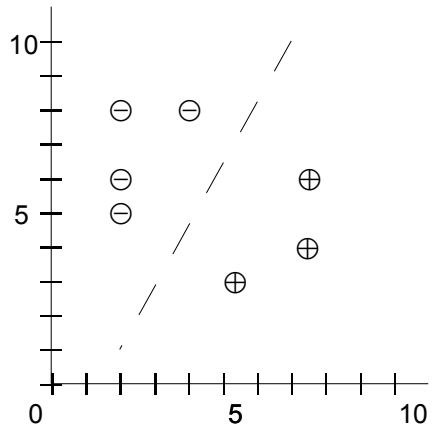

Figure 2.1: Visualization of 7 feature vectors of length 2 in two dimensional space.

In the diagram can be seen that the positive and negative examples build distinguished groups. The positive and negative examples are grouped so clearly, that a line can be drawn, which divides the whole plane in such a way that positive examples are located on one side of the line and the negative examples on the other.

This simplified example shows what an SVM classifier does: in the learning phase, it tries to divide the feature space into into two parts in such a way that the positive and negative examples are separated from each other. Then, in classification phase, a new, previously unseen example is assigned a label depending solely on its location relative to the separating line.

SVM classifiers can do this not only for a two-dimensional feature space, but also for higher-dimensional feature spaces. In case of an n-dimensional feature space, the result of the learning phase would not be a line, but an (n-1)-dimensional hyperplane.

In the situation given in the Example 2.1, there are infinitely many possibilities to draw such a line. Three of them, named a, b and c can be seen in Figure 2.2.

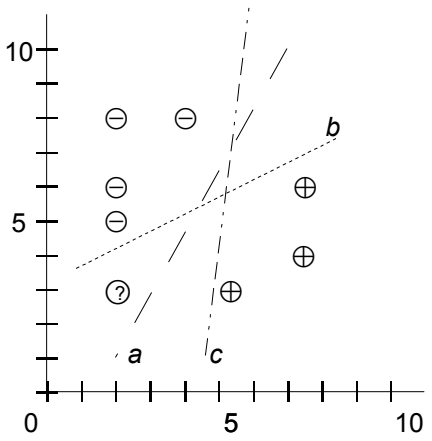

Figure 2.2: There are many lines which separate positive and negative examples.

Intuitively, it is quite important for the classification of new examples, which one of the lines is chosen in the learning phase. To illustrate this, let us consider the new instance labeled with "?" in Figure 2.2: with respect to lines *a* and *c*, it would be classified as positive, but with respect to line *b*, it would be classified as negative.

Clearly, we are interested in a solution, which generalizes – as far as possible – beyond the training data, as the classification of new previously unseen data instances is our ultimate objective. On the contrary, what we obviously do not want is the situation, where a classifier fits the training data very well, but it performs very badly on new examples. Such situation is called *overfitting* in machine learning.

There are two things which have strong impact on the classifier's ability not to overfit the training data.

Firstly, the classifier should by its design draw the boundary in such a way, that it does not prefer any of the divided groups. It is a particular strength of SVM that it selects the line which is located just in the middle between the nearest positive and negative examples. The boundary line is said to have *maximum margin*. In the Figure 2.2, the line *a* has maximum margin. This seems to be the most robust solution, which is best resistant against *overfitting* (Culotta and Sorensen, 2004).

Secondly, the classifier's ability to classify previously unseen examples correctly depends largely on the amount of the training data. Intuitively, the more training examples the more

general the learned model will be. Albeit some experiments which compared the performance of a particular PPI extractor on learning resources of several different sizes were recently performed by Airola et al. (2008), there is no generally accepted minimum of learning examples in the literature.

The task given in Example 2.1 is quite simple, because the positive examples can be clearly separated from the negative ones. However, situations are conceivable in which it is not possible to draw the separating line. In such cases, a so called s*oft margin* is used (Cortes and Vapnik, 1995). This means that the SVM is allowed to draw the boundary in such a way that it allows misclassified examples within some given distance from the separating line. The soft margin is characterized by the constant *c*. Thus, a trade-off between training error and margin is introduced (Joachims, SVM^{light}, see Chapter 10 Software and Corpora).

2.1 SVM for Relation Extraction

In the field of relation extraction we deal with sentences. The reader might ask, how can sentences be transformed to tuples of numbers, so that they can be used by an SVM as training examples? Generally, two solutions to this problem can be found in the literature.

The first solution is quite straightforward: the user of the SVM must invest a substantial effort to provide a set of rules, which transform sentences to feature vectors. This means on one hand to define which positions in the vector characterize which properties of a given sentence and on the other hand, to define how the given property is transformed into a number. The sentence characteristics one could choose may be based e.g. on word *n*-grams, POS *n*-grams, or parse tree substructures like *has an NP-VP subtree* (Culotta and Sorensen, 2004). When e.g. POS bigrams are taken as a base for the feature vectors, one would need to accomplish the following steps:

- List all possible bigrams and assign an index to each member of the list. E.g. for 36 Penn Tree Bank POS tags we would have 36^2 bigrams. We could assign an index to each bigram e.g. according to its position in an alphabetical order.
- In this way, we have obtained 36^2 yes/no sentence properties, which can be transformed to numbers very easily: one obvious way how to do it, is to assign 1 to the given POS bigram feature if the given POS bigram occurs in the given sentence or to assign 0 to it otherwise.

The second approach to solve the problem how natural language sentences can be fed into an SVM has to do with SVM internals. It is called *kernel trick*. It is based on the fact that there are two possibilities how an SVM can be formulated: one of them uses a dot product of explicit feature values (as we assumed so far) and the other one replaces the dot product with so called *kernel function*. The kernel function can be thought of as similarity measure of two given data instances – in our case sentences.

Lets us mention the "bag-of-words" as an example for a kernel function. It simply characterizes the similarity of two sentences through the number of words they have in common.

Convolution kernels represent a special type of kernel functions. They are intended for cases when the structure of instances is important. The main idea is to qualify the similarity of two structures through summing the similarities of their substructures. In this way the similarity of two strings can be characterized through the number of their common substrings. Analogically, the similarity of two trees can be determined as the number of their common subtrees. The matching substructures can be effectively found using dynamic programming (Culotta and Sorensen, 2004).

The main advantage of the kernel approach is the possibility to cover very large (potentially infinite) feature spaces without handling the features explicitly if an efficient procedure to compute the kernel function is available. Depending on the effectiveness of the kernel computation, substantial substantial reduction of time and memory can be gained (Kuboyama et al., 2007).

In this thesis the kernel approach will be pursued using the SVM^{light} software of Thorsten Joachims (Joachims, 1999). Later, in Chapter 6 Experimental Setup, several kernels will be presented and evaluated on the PPI task.

3 Training Data

The chosen method presupposes a resource with annotated entities and relations on which the statistical model will be trained. In this chapter, we introduce the resources we have used to evaluate the kernels presented in Chapter 4.

Pyysalo et al. (2007) note that while there is a "rough consensus" on the annotation of protein names, there are no explicit and widely accepted definitions of Protein-Protein Interaction (PPI) and their annotation. Consequently, it is very hard to compare the performance of PPI-extraction methods that were evaluated on different corpora.

Pyysalo et al. (2007) attempted to examine available PPI-annotated resources and they have described what is their "greatest common factor". For their survey, they gathered resources which:

- are freely available
- have specifically identified named entities
- have manually annotated interactions
- and in which negative examples of PPI are either explicitly marked or can be validly generated under the closed-world assumption.

The following five corpora fulfilled these criteria:

- AIMed (Bunescu at al., 2005b)
- BioInfer (Pyysalo et al., 2007)
- HPRD50 corpus (Fundel et al., 2007)
- IEPA corpus (Ding et el., 2002)
- PPI corpus produced for the LLL challenge (Nédellec, 2005).

Annotation layers of all these corpora carry the information about named entities from the domain of biology. However, only LLL and BioInfer distinguish between several types of entities, such as proteins and genes.

A very important fact for PPI extraction is that the annotation of relevant entities is exhaustive

only in AIMed and BioInfer. The entity annotation of the other corpora is based only on lists of entity names or named entity recognizer output (Pyysalo et al., 2008).

Pyysalo et al. (2008) report that the differences in interaction annotation are even greater than those in entity annotation: in particular, only the BioInfer and IEPA corpora contain information identifying the words stating the interactions, all but HPRD50 specify the direction of interactions, BioInfer alone contains complex or negative interactions and only HPRD50 annotates different interaction certainties. Finally, BioInfer is the only corpus to contain annotation for static entity relations such as protein family membership.

As a result of their survey, Pyysalo et al. (2008) state the following PPI annotation principles as "the greatest common factor" applicable to all five named corpora:

Protein Protein Interactions should be treated as:

- undirected
- untyped
- of non-static types
- bearing no specification of words stating the interaction
- having no complex structure
- containing no information about negation
- containing no information about interaction certainty

The basic characteristics of the five corpora can be seen in Table 3.1.

Corpus	Documents*	Sentences	Positive Pairs**	Negative Pairs**
Aimed	220	1955	1000	4834
BioInfer	833	1100	2534	7132
HPRD50	43	145	163	270
IEPA	200	486	335	482
LLL	45	77	164	166

Table 3.1: Corpora

* Only documents with at least one (positive or negative) pair are counted.

** The example pairs are checked for (orderless) uniqueness and for being non-reflexive

3.1 Learning Format

Pyysalo et al. (2008) have also defined an XML-based format for the annotation of PPI, which they called *learning format*. They provide transformations of the the named resources into this format. The corpora in the learning format are available on the web site of Department of Information Technology of University of Turku (see Section 10 Software and Corpora on page 54).

Four out of these five transformed corpora were used throughout this thesis for evaluation. We were forced to exclude the BioInfer corpus from our evaluations as both the time we had for finishing of this thesis and the computational resources we could use were limited.

The rough structure of a learning format corpus can be seen in Figure 3.1. Corpus contains documents and documents contain sentences. The sentence text can be found in the attribute *text*.

```
<corpus source="AIMed">
  <document id="AIMed.d0" origId="11780382">
    <sentence id="s328" text="Eotaxin-3 competed the binding of
       (125)I-eotaxin to CCR3-expressing L1.2 cells with an IC(50) of 13 nM.">
    <entity charOffset="0-8" id="s328.e0" text="Eotaxin-3" type="protein" />
    <entity charOffset="39-47" id="s328.e1" text="I-eotaxin" type="protein" />
    <entity charOffset="52-55" id="s328.e2" text="CCR3" type="protein" />
    <pair e1="s328.e0" e2="s328.e1" id="s328.p0" interaction="False" />
    <pair e1="s328.e0" e2="s328.e2" id="s328.p1" interaction="True" />
    <pair e1="s328.e1" e2="s328.e2" id="s328.p2" interaction="True" />
    <sentenceanalyses>
      <parses>
        <parse parser="Charniak-Lease" tokenizer="Charniak-Lease">
          <dependency id="clp_1" t1="clt_2" t2="clt_1" type="nsubj" />
          ...
        </parse>
      </parses>
      <tokenizations>
        <tokenization tokenizer="Charniak-Lease">
          <token POS="NN" charOffset="0-8" id="clt_1" text="Eotaxin-3" />
          ...
        </tokenization>
      </tokenizations>
    </sentenceanalyses>
  </sentence>
  </document>
  ...
</corpus>
```

Figure 3.1: Learning format example.

The actual annotation of named entities and relations is encoded through *entity* and *pair* elements. The position of an entity in the sentence text is expressed in the *charOffset* attribute of *entity*. Using offsets instead of nested XML elements is usually called *standoff* annotation in linguistics.

Note, that the presence or absence of a relation is marked on the level of named entity pairs, not on the level of sentences (cf. attribute *interaction* in Figure 3.1). The motivation for this is quite straightforward: In the context of PPI, we are interested in binary relations, i.e. such that hold for pairs of objects. However, there are sentences, as the one in the Figure 3.1, in which there are more than two named entities. In such sentences, it is possible, that a relation holds only for some of entity pairs. Indeed this is also the case for the sentence in the Figure 3.1: there is a relation between entities *e0* and *e2* and between entities *e1* nad *e2*, whereas there is no relation between entities *e0* and *e1*. Thus, in the learning format all possible entity pairs are listed and each of them is explicitly annotated for presence or absence of a relation. As a consequence of this, the learning examples, which are used by a statistical classifier, correspond to pairs rather than sentences.

The learning format also provides means for expressing token boundaries and syntactic parses of sentences. Analogically to named entities, token boundaries are expressed in the *charOffset* attribute.

The format allows for storing several alternative tokenizations and parses for a given sentence. The corpora available from University of Turku contain two alternative parses and tokenizations for each sentence. One of them is the output of Charniak-Lease Parser (Charniak and Lease, 2005) which was used in several experiments of the present work; see also 3.2 Syntax Representations.

3.1.1 Learning Format Pitfalls

The learning format annotation scheme has some important implications for any relation extraction system using it as an input:

1. Named entities may overlap. The string “Arp2/3” (see Figure 3.2) contains two named entities, namely “Arp2” and “Arp 3”.
2. An entity may spread over multiple noncontiguous text ranges. The entity “Arp 3”

from the previous paragraph spreads over two ranges: the first goes from character 0 to character 2 and the second covers the single character 5 of the sentence (see Figure 3.2).

3. Such noncontiguous and overlapping entities may constitute a relation – see Figure 3.2.

4. There is no guarantee of a one-to-one correspondence between named entity boundaries and token boundaries. Virtually any combinations of entailment and overlapping are possible: one entity may spread over several tokens, one entity may correspond to a mere part of a token, there may exist several named entities in one token (see Figure 3.2), etc.

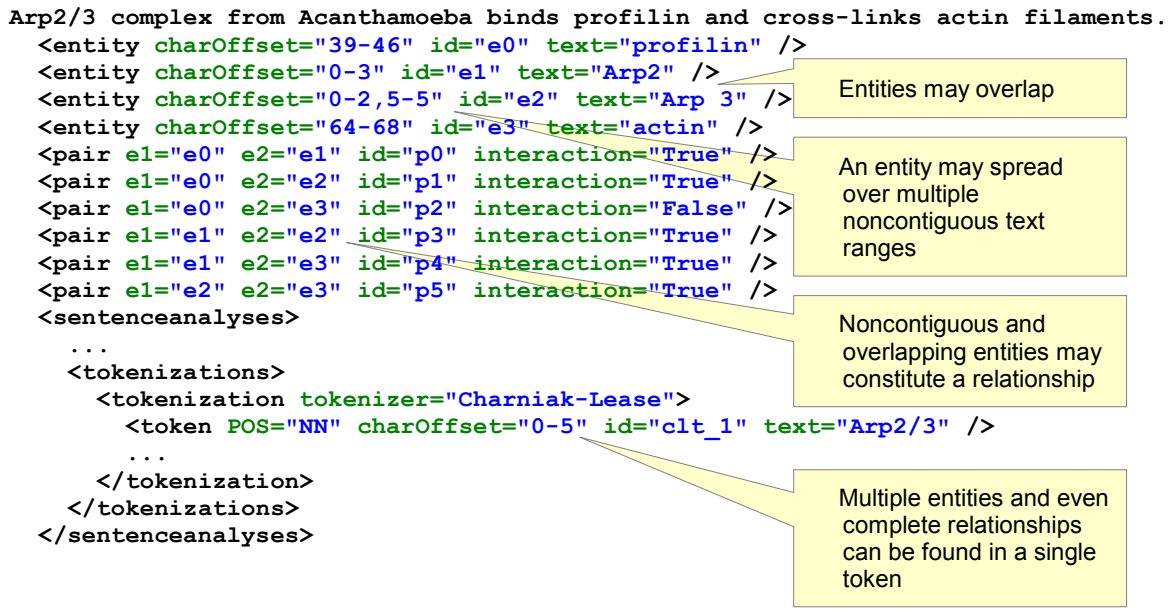

Figure 3.2: Learning format pitfalls (sentence BioInfer.d77.s0).

3.2 Syntax Representations

Relation extraction methods evaluated within this thesis are based on syntactic information. Syntactic analyses of sentences for purposes of relation extraction are produced by special programs, called *parsers*. Parsers are often said to produce *parses*. Hence *parse* and *syntactic*

analysis of a sentence are synonyms.

There are two main syntax formalisms, which currently dominate in the field of relation extraction: (1) constituent trees (see Figure 3.3) and (2) dependency graphs (see Figure 3.4).

Constituent trees (also called phrase structure parses) are *trees* as defined in graph theory. As every graph, a tree consists of nodes (also called vertices) and edges (also called arcs). What makes trees a special kind of graphs is the fact, that they are *acyclic* and *connected*, i.e. any two edges are connected by exactly one path. Constituent trees are furthermore *rooted.* It means that one of the nodes has been designated as a root. Trees can be viewed as implicitly directed, because every edge can be seen as having direction from the root (Kuboyama et al., 2007).

Contrary to constituent trees, dependency graphs do not fulfill the condition of being acyclic (see Figure 3.4). So they are no trees in the sense of graph theory. Furthermore, they are explicitly directed and rootless.

From the linguistic point of view, the nodes in a constituent tree represent phrases and clauses and the root of the tree represents the whole sentence. The tree leaves stand for words (Clegg and Shepherd, 2007). The edges of a constituent tree represent the rules of the underlying grammar.

On the other hand, each node in a dependency graph represents a word of the sentence and each edge represents a grammatical dependency such as that which holds between a verb and its subject (Clegg and Shepherd, 2007).

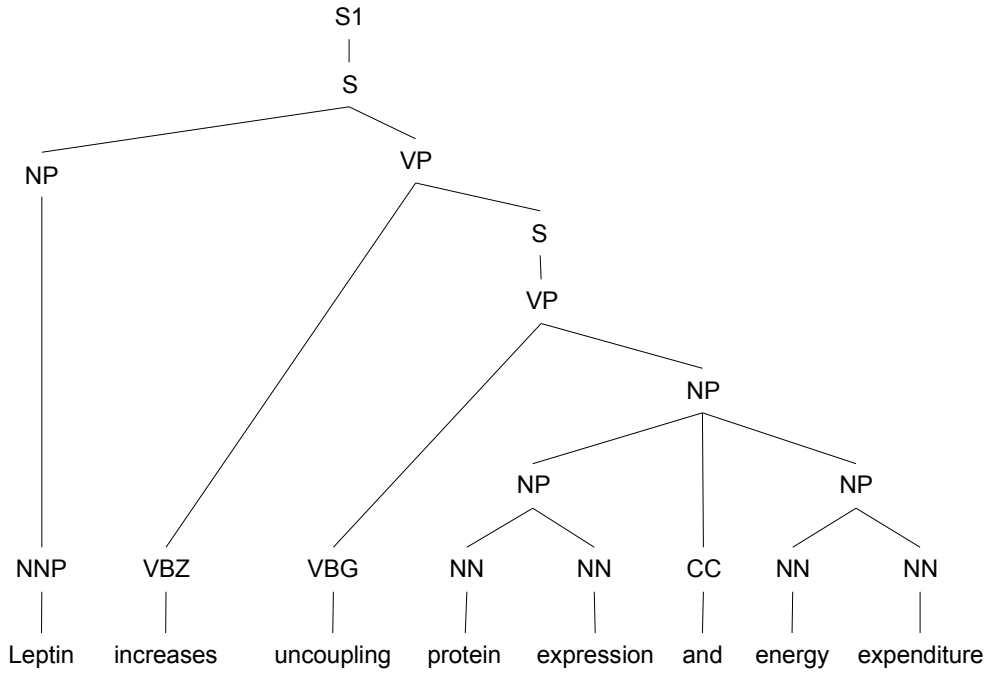

Figure 3.3: Constituent tree parse for the sentence IEPA.d48.s123; an output of Charniak-Johnson-McCloski parser.

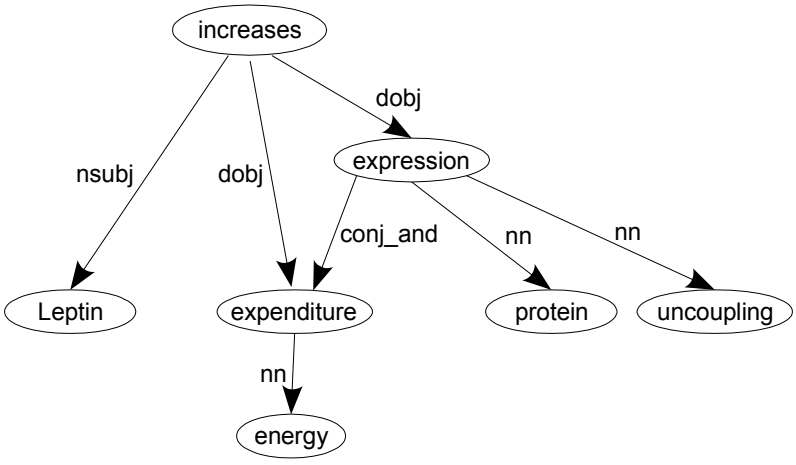

Figure 3.4: Dependency graph parse for the sentence IEPA.d48.s123; an output of Charniak-Lease parser, which was transformed to typed dependencies using Stanford Tools.

Phrase structure representations are better established in the computational linguistic

community than dependency graphs (de Marneffe and Manning, 2008). There is a *de facto* standard for the format of constituent parses which is given by the Penn Treebank (PTB) (Marcus et al., 1994). There are several high-performance parsers available, trained on the PTB. Chaniak-Lease and the Bikel parser are among the best of them (Clegg and Shepherd, 2007).

On the other hand, there is no generally accepted standard for dependency graphs. There is no large standard resource with dependency graph annotations comparable to PTB. The resources are rather small and they use different sets of dependency types and attachment rules (Clegg and Shepherd, 2007). Consequently dependency parsers are not as robust and accurate as phrase-structure parsers trained on very large corpora (de Marneffe et al., 2006).

However, there is an important theoretical link between constituent trees and dependency graphs. The constituent trees can be transformed into dependency graphs through applying some deterministic rules. De Marneffe et al. (2006) provide such a rule set together with an implementation (see Stanford Tools in the Chapter 10 Software and Corpora on page 54). In this way, every phrase structure parser can actually be used to produce dependency graphs.

Several authors have recently advocated the opinion that dependency graphs are more suitable for information extraction purposes than constituent trees. Their main argument is that in dependency graphs, the semantic relationships are closer to the surface, while the same information is not “readily available” from phrase structure parses (Clegg and Shepherd, 2007; de Marneffe et al., 2006). We have performed experiments with both dependency graphs and constituent trees. Our results can be seen in Chapter 7.

4 Kernels for Relations Extraction

In this chapter, two groups of kernel functions (or shortly kernels) for relations extraction will be presented. Firstly, some kernels known from the literature will be explained. Secondly, some attempts to design a novel kernel will be described.

4.1 Kernels of other Authors

The following kernels will be characterized in this section:

- Subtree Kernel (ST) (Vishwanathan and Smola, 2002)
- Subset Tree Kernel (SST) (Collins and Duffy, 2002)
- Partial Tree Kernel (PT) (Moschitti, 2006a)
- Spectrum Tree Kernel (Kuboyama et al., 2007)

All of these kernels can be seen as a closely related family of kernels because they strictly follow the idea of convolution kernel as introduced by Hausler (1999) and because they operate on tree substructures. So, they quantify the similarity of trees by counting their common substructures. “Common” means here that the substructures of two compared sentences must match exactly. The named kernels differ substantially in how they understand the tree substructure.

4.1.1 Subtree Kernel (ST)

For subtree kernel, the counted tree substructure – which is called *subtree* – is defined as any node together with all its descendants down to (and including) the tree leaves (Vishtanam and Smola, 2002); see an example in Figure 4.1. When deciding if a given subtree *s* is common for two trees, the order of child nodes within *s* is important: the order of child nodes must be the same across all its nodes which have children.

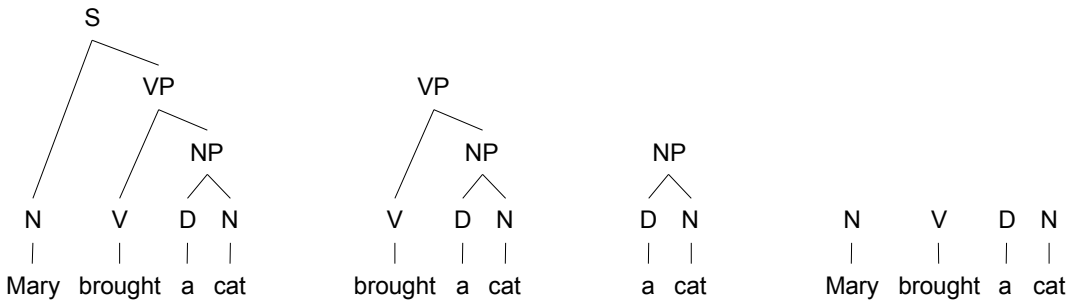

Figure 4.1: A phrase structure parse along with its subtrees (Moschitti, 2006a).

4.1.2 Subset Tree Kernel (SST)

The subset tree kernel relaxes the constraint that all descendants, including leaves, must always be included in the substructures. However, it retains another constraint, which limits the generality of permitted tree substructures: The grammatical rules (mentioned in Section 3.2) may not be broken. This means, that for a given tree node, either none or all of its children must be included into the resulting *subset tree*. An example can be seen in Figure 4.2.

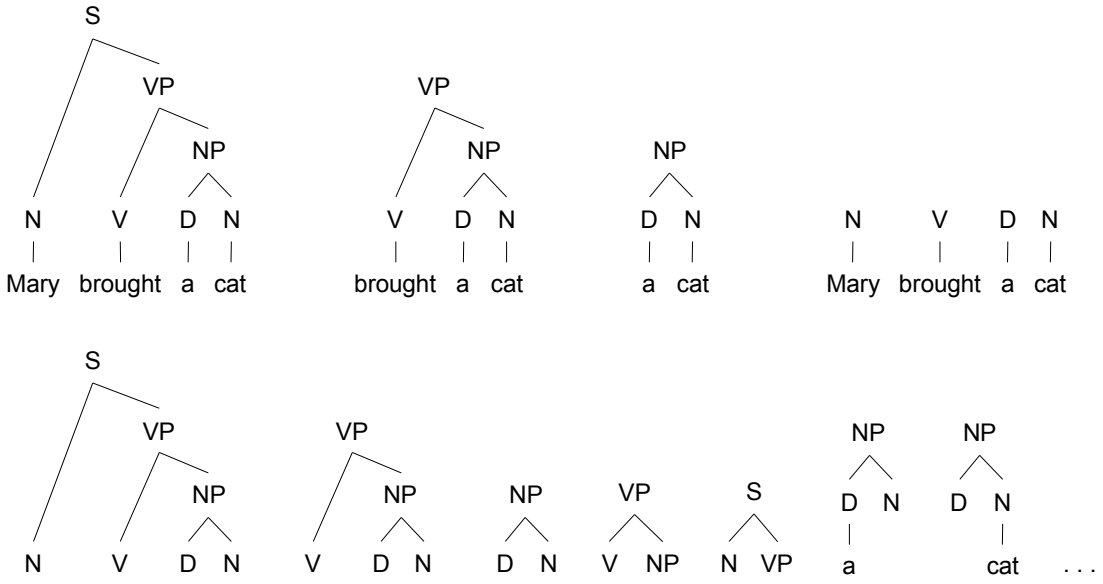

Figure 4.2: A phrase structure parse along with some of its subset trees (Moschitti, 2006a).

As in the case of ST kernel, also in SST kernel the order of child nodes within a given subset tree is important.

4.1.3 Partial Tree Kernel (PT)

The partial tree kernel is the most permissive kernel from those presented here: it allows virtually any tree substructures; no matter if the leaves are included or not and no matter if the grammatical rules are broken or not. An example can be seen in Figure 4.3.

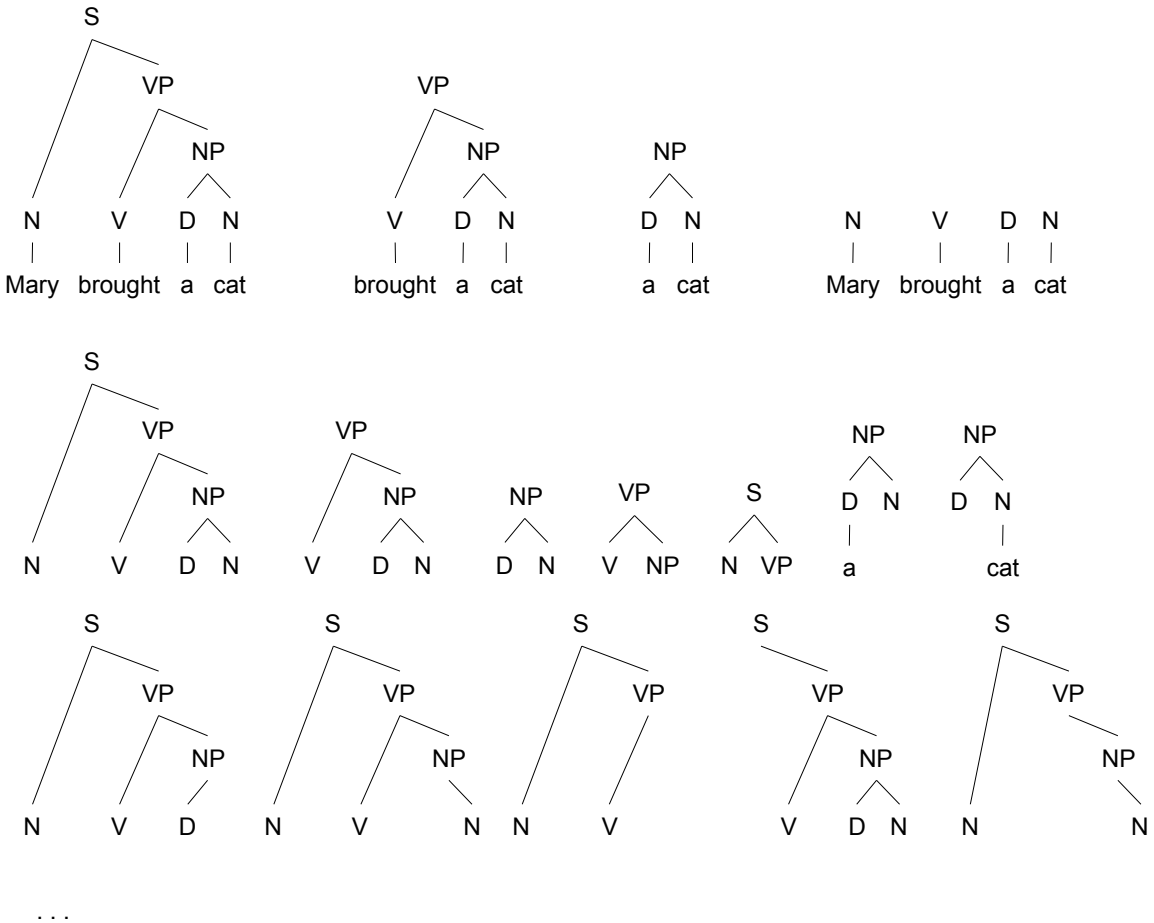

Figure 4.3: A phrase structure parse along with some of its partial trees (Moschitti, 2006a).

Again, is in the case of ST and SST kernels, also in PT kernel the order of child nodes within a given partial tree is important.

4.1.4 Spectrum Tree Kernel (SpT)

Finally, the substructures counted by the spectrum tree kernel are called *tree q-grams*. Tree *q*-grams can be seen as a special case of paths. The notion of path belongs to elements of graph theory. It is defined as is a sequence of graph nodes such that from each of its nodes there is an edge to the next node in the sequence. Tree *q*-grams are a special kind of paths, because of their fixed length *q*. Further, in *q*-grams, the orientation of their edges is important. As noted in Section 3.2, trees can be interpreted as implicitly oriented graphs, the direction being e.g. from root to leaves. So, two 3-grams such as $a \leftarrow b \rightarrow c$ and $a \rightarrow b \rightarrow c$ are not identical, as they differ in the direction of the edge between *a* and *b*. On the other hand, the order of *q*-gram serialization is unimportant: $a \rightarrow b \rightarrow c$ is identical with $c \leftarrow b \leftarrow a$.

An example can be seen in Figure 4.4.

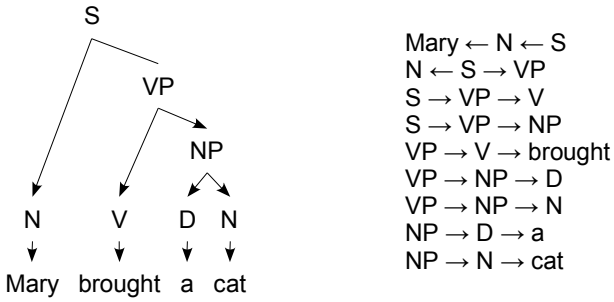

Figure 4.4: 3-grams generated out of a constituent tree.

4.1.5 Comparison

From the provided characterizations and examples can be seen that the amount of possible substructures differs substantially between the presented kernels. This has naturally a strong impact on the computational complexity of these kernels on one hand and on the accuracy of these kernels on the other hand.

For the subtree kernel, a linear complexity algorithm was prosed by Vishwanathan and Smola (2002). Collins and Duffy (2002) have proven that the subset tree kernel complexity is quadratic in the number of tree nodes. Moschitti (2006a) reports that the partial tree kernel also has a quadratic complexity. However, he claims that the running time is linear in the

average case. Kuboyama et al. (2007) have proven that the spectrum tree kernel can be computed in linear time with respect to the number of tree nodes.

We have evaluated all four kernels on the PPI task. The results in terms of running time and accuracy are presented in Chapter 7 Experimental Results.

The application of all these kernels to the relation extraction problem seems to be new, except for the SST kernel which was used for PPI extraction by Miwa et al. (2008). The ST, SST and PT kernels were already used for (or partly indeed designed for) tasks in the domain of natural language processing. Collins and Duffy (2002) use their SST kernel as a parse re-ranker. Moschitti (2006a) applies his PT kernel on semantic role labeling and question classification tasks. On the contrary the spectrum tree kernel of Kuboyama et al. (2007) was so far – to our very best knowledge – evaluated only on glycan molecules classification.

Two state-of-the-art kernels were published recently by Airola et al. (2008) and Miwa et al. (2008). Our results are compared with these kernels later Chapter 7.

The all-paths graph kernel of Airola et al. (2008) is a convolution kernel which counts weighted shared paths of all possible lengths. These paths are generated from a dependency parse on one hand and from the surface sequence of the words on the other hand. The path weights are determined by weights of dependencies which in turn depend on the given dependency's location relative to the shortest path between the candidate entities: The dependencies belonging to the shortest path between candidate entities get higher weight than those outside the shortest path.

The kernel of Miwa et al (2008) combines multiple layers of syntactic information by applying three distinct kernels on outputs of two different syntax parsers. The bag-of-words kernel, the subset tree kernel of Moschitti (2006b) and all paths kernel of Airola et al. (2008) are used together with dependency parses and deep parses. The combination of the kernels is achieved through summing the normalized values of each kernel for each parse.

Further kernel-based methods applied to the PPI extraction task were presented e.g. in Zelenko et al. (2003), Bunescu and Mooney (2005 and 2006), Kim et al. (2008) and Özgür et al. (2008).

4.2 Design of a Novel Kernel

This section documents two attempts to devise a novel kernel which we value as unsuccessful because of the insufficient accuracy. Later a more successful attempt is described.

4.2.1 Preliminary Attempts to Design a New Kernel

The spectrum tree kernel of Kuboyama et al. (2007) was chosen as a starting point for our innovations. We saw two main reasons for this: (1) It has shown a performance superior to ST, SST and PT kernels (see Chapter 7 Experimental Results). (2) Spectrum tree kernel has so far not been evaluated on NLP tasks.

Thus, all kernel proposals which will be presented in Sections 4.2.1, 4.2.3, and 4.2.4 will be based on the idea of counting common q-grams.

As pointed out in Section 3.2, dependency graphs are generally considered as more suitable for information extraction purposes than constituent trees. In accordance with this, our first attempt to design a new kernel was based on the idea to adapt the spectrum tree kernel for dependency graphs.

There are two ways how to achieve this: either to convert the general dependency graphs to trees, or to redefine the spectrum kernel for general dependency graphs.

4.2.1.1 Transforming Dependency Graphs to Trees

The first strategy basically requires to eliminate cycles from the general dependency graphs. Some cycles in dependency graphs – as the one in Figure 3.4 – exist due to the fact that nodes can have more than one governor. Clearly, these can be eliminated through enforcement of a single governor constraint. We believed that some dependency types are more important for relation extraction than others. So, we defined an ordering over the dependency types according to which multiple governing dependencies were removed. The ordering was roughly the following:

conjunctions – prepositions – modifiers – object – subject

So, when a node had more than one governor, we have removed the governing dependency types in the named order until there was only one governor. Sometimes this made the whole graph disconnected. In such situations a new dummy root node was introduced to make the

whole graph connected again.

After enforcing single (or no) governor in each graph node, there may still exist complex cycles in the resulting graphs. As these were relatively rare, we have employed some simple heuristics to remove them:

1. Some node was chosen as the future root node from those nodes from which all other nodes were reachable. If there were more than one such node, we have selected the first one according to the surface order in the sentence. If there was none such, we have created a new dummy root node and connected it appropriately with the rest of the graph.
2. Starting with the selected root node the dependents were traversed recursively while marking all already visited nodes so that they are visited only once. This was actually a *tree* traversal.

The corpora were transformed into trees in this way so that they could be directly fed into the unchanged spectrum tree kernel. However, the accuracy of this solution stood close to random.

4.2.1.2 Redefinition of the Spectrum Tree Kernel for General Graphs

We have interpreted the failure of the previous attempt so that the eliminated dependencies were in fact important for the relation extraction. Thus, we decided to realize the second strategy: To redefine the spectrum kernel to work on general dependency graphs rather than trees. This actually meant no conceptual change to the kernel as q-grams can be defined generally on graphs as well as on trees. Because the edges in dependency graphs are labeled and because these labels bear important information (the type of the dependency), these labels were integrated into q-grams. This was achieved by handling dependency types as standalone nodes.

Note, that despite including the dependency types in q-grams, we coin a convention not to include them in q. q stands for the number of nodes from the original graph. That is why the bigrams in the Figure 4.5 actually have 3 elements. We have also allowed only such q-grams which start and end with a surface token. Therefore there is no such bigram as nsubj ← brought → dobj in Figure 4.5. We had no apparent reason for doing it this way. Permitting the dependency types to occupy not only even but also odd positions in q-grams could perhaps be

proven to perform better in the future.

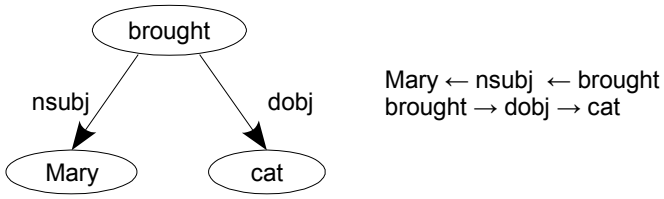

Figure 4.5: Dependency graph example along with its two bigrams.

We have evaluated this kernel for values of q ranging from 2 to 4. For each of these values we could reach only performance close to random.

4.2.2 The Successful Strategy

We analyzed problems and found the failures of both previous strategies as having two causes which we explain in detail in the following two Sections 4.2.2.1 and 4.2.2.2. These two points are used for the construction of two new kernels in Sections 4.2.3 and 4.2.4.

4.2.2.1 Fuzzy Matching

Constituent trees are more pattern-like than dependency graphs. In both constituent trees and dependency graphs, there are two kinds of elements: (i) surface tokens, e.g. “Leptin”, “increases” in Figures 3.3 and 3.4 and (ii) generic labels like “NP”, “VP” in Figure 3.3 or “nsubj”, “dobj” in Figure 3.4. While there are relatively many possible surface tokens, there are comparably few generic labels. Consequently, it is much more probable that generic labels will match than that surface tokens will match. In constituent trees, only leaf nodes can have surface tokens as labels. On the contrary, in dependency graphs, each node is labeled with a surface token and only dependencies are labeled with generic labels. So while in q-grams generated from constituent trees, there will be at most two surface tokens (at the beginning and at the end), in q-grams generated from dependency graphs, there will always be $(q + 1) / 2$ surface tokens which is at *least* 2. So, the probability that two q-grams generated from dependency graphs will match, is generally lower than the probability that two q-grams

generated from constituent trees will match.

To eliminate this disadvantage of dependency graphs, we propose to use some kind of fuzzy matching instead of exact matching. In the next two sections we describe two fuzzy matching techniques which we have implemented in the kernels introduced later in Sections 4.2.3 and 4.2.4.

4.2.2.1.1 Tolerant matching

We have developed this technique specially for dependency tree *q*-paths as described in Section 4.2.1.2. Tolerant matching distinguishes three distinct types of elements in dependency tree *q*-paths:

i. dependency types (shortly Ds)

ii. candidate entities (shortly Es; see also Section 6.1.2 Entity Blinding)

iii. surface tokens other than candidate entities (shortly Ls).

When matching two *q*-grams, firstly some mismatches will be tolerated and secondly, the matches of Ds, Es and Ls will be scored differently. "Tolerate" means, that if elements on a given position of given two *q*-paths do not match, the score for the given position will be simply 0; "not tolerate" means that if elements on a given position of given two *q*-paths do not match, the similarity score of the *whole q*-path pair will be 0. Several examples can be seen in Figure 4.6.

score_tolerant(Mary ← nsubj ← brought, Mary ← nsubj ← brought) = 1 + 6 + 1 = 8
score_tolerant(Mary ← nsubj ← brought, Mary ← nsubj ← sees) = 1 + 6 + 0 = 7
score_tolerant(Mary ← nsubj ← brought, Mary ← dobj ← brought) = 0
score_tolerant(_ENTITY_1_ ← nsubj ← phosphorylates, Mary ← nsubj ← brought) = 0 + 6 + 0 = 7
score_tolerant(_ENTITY_1_ ← nsubj ← phosphorylates, _ENTITY_1_ ← dobj ← phosphorylates) = 0

Figure 4.6: Tolerant matching examples. In these examples, L and E mismatches are tolerated, while D mismatches are not tolerated; scores for matching Ls, Es and Ds are $l = 1$, $e = 3$, $d = 6$

We have experimented with tolerating and not tolerating mismatches of all three groups. Tolerating both L and E mismatches and not tolerating D mismatches has consistently proven

to deliver the best accuracy over all corpora.

As for different scores for L, E and D matches, the scores which performed best, were different for each corpus. See Section 6.2 Parameters and their Optimization.

4.2.2.1.2 Levenshtein distance

Levenshtein distance (Levenshtein, 1966) is a well known measure for quantifying similarity of two strings. It is defined as the number of operations needed to transform the first of the compared string to the second compared string. The allowed operations are insertion, deletion and replacement. Levenshtein distance can be straightforwardly adapted to work with graph *q*-grams instead of strings. However, to fit the general contract of convolution kernel (more similar substructures should get higher score), we have used the complement of Levenshtein distance rather than Levenshtein distance itself:

$$\text{levenshtein\_complement} = q - \textit{levenshtein}(\text{q\_gram}_1, \text{q\_gram}_2)$$

4.2.2.2 Context Selection

In this section we sketch what we see as the second reason for the failures of the kernels documented in Sections 4.2.1.1 and 4.2.1.2. All of the kernels presented so far quantify the similarity of the *whole* sentences. However, the similarity of whole sentences is not exactly what we are looking for. For the purposes of relation extraction, whole sentences do not need to be similar to express the same relation. It is enough if they contain a similar part which actually expresses the relation. Particularly, when working with convolution kernels, it is enough to count the common substructures only in those parts of the sentence which are likely to express a relation. To make this idea usable, it must be specified which part of the sentence expresses the relation or which one is the most likely to express it. We call this the *context selection problem*.

Intuitively, the named entities themselves are the minimum of the sentence which *must* belong to the selected context. However, which parts of the sentence are important except for the named entities themselves?

Airola et al. (2008) report that

> "[i]t is widely acknowledged that the words between the candidate entities or connecting

them in a syntactic representation are particularly likely to carry information regarding their relationship"

but by the same token they argue that regarding only the nodes on the shortest path may exclude relevant words in "many simple cases", e.g.:

P1 is a *P2* binding protein.

Airola et al. (2008) solve this problem through assigning different weights to structures outside of the shortest path.

We propose two alternative solutions to the context selection problem. Both of them use the idea to include graph elements within a fixed range around some "center". We have two proposals what this "center" could be. Firstly, both entity candidates are considered as the "center" for the context selection and secondly, the shortest path itself constitutes the "center".

Based on these two context selection strategies, we have designed two spectrum graph kernels. Their detailed descriptions can be found in the following two sections.

4.2.3 *d*-Entity Context Spectrum Kernel

This kernel adopts the first strategy for the context selection. So, to compare two sentences the following steps need to be performed:

1. In both sentences, select all dependency graph nodes which are within the distance *d* from the candidate entities. The distance between a node *n* and the candidate entity node *e* is equal to the length of the shortest path between *n* and *e*.
2. For the contexts selected in both sentences, all possible *q*-grams are generated. So two sets of *q*-grams are obtained, one for each sentence.
3. For each *q*-gram from the first set, the best match from the second set is found.
4. The match scores of the best matches are summed together. This sum represents the value of the *d*-entity context spectrum kernel function for the two input sentences.

We have tested this kernel with both tolerant and Levenshtein matching but the best result in terms of AUC on AImed corpus we could reach was 0.56. Thus we have resigned from evaluating this kernel in depth.

4.2.4 *k*-Band Shortest Path Spectrum Kernel (kBSPS)

The kernel presented in this section reflects the argument of Airola et al. (2008) that the shortest path is not enough for relation extraction. Except for the shortest path between the candidate entities, dependency graph nodes and dependencies within distance k from the shortest path are included in the selected context. The exact method for comparing two sentences is the following:

1. In both sentences, find the shortest path P between the candidate entities e_1 and e_2.
2. In both sentences, select all dependency graph nodes which belong to P or which are within the distance k from P. The distance between a node n and P is equal to the length of the shortest path between n and node m, where $m \in P$.
3. For the contexts selected in both sentences, all possible q-grams of lengths between q_{min} and q_{max} are generated. So two sets of q-grams are obtained, one for each sentence.
4. For each q-gram from the first set, the best match having the same q from the second set is found.
5. The match scores of the best matches are summed together. This sum represents the value of the kBSPS kernel function for the two input sentences.

Allowing for several q-s between q_{min} and q_{max} bears some resemblance to the listing of all possible graph paths in all-paths graph kernel of Airola et al. (2008). However, we have evaluated the kBSPS kernel only with a very limited set of q_{min} and q_{max} max values (see Section 6.2).

We have tested both tolerant matching and Levenshtein distance variants of the kBSPS kernel. Tolerant matching has shown much better accuracy and speed than Levenshtein distance. The results in terms of AUC as well as precision, recall and f-measure can be seen in Chapter 7.

5 Evaluation Methods and Metrics

As noted earlier in Chapter 2, the use of a statistical classifier has two phases: the first one being the learning on an annotated corpus and the second being the actual classification of new, previously unseen examples. When devising a novel classification method, it is important to be able to assess how accurate the method is in the second phase. In this chapter we will firstly present metrics, which can be used for the performance quantification and secondly, we will explain the concept of *cross-validation* – a method for systematic partitioning of the annotated resource into training and evaluation part.

5.1 Precision, Recall and F-Measure

Precision, recall and F-measure are a *de facto* standard for the evaluation of information extraction methods. To define themselves, we need to define some basic notions first:

Definition 5.1: Gold standard (GS) – a set of examples with proper classification labels assigned to them. "Proper" means here that for the purposes of evaluation, the are no doubts about the correctness of the labels. The correctness is usually guaranteed by some trusted instance, e.g. a human expert.

Definition 5.2: Prediction – a set of examples, which contains the same examples as GS, but they are labeled by the method, which is being evaluated.

Definition 5.3: Positives – the subset of prediction, which contains only examples labeled as positive by the method.

Definition 5.4: Negatives – the subset of prediction, which contains only examples labeled as negative by the method.

The evaluation measures defined here are based on the comparison of prediction with the gold standard. Any mismatch between the prediction and gold standard is considered to be an error. There are errors of two kinds:

Definition 5.5: False positives (FP) – the set of examples, which were labeled as positive by the method, but are labeled as negative in the GS.

Definition 5.6: False negatives (FN) – the set of examples, which were labeled as negative by the method, but are labeled as positive in the GS.

Analogically, the examples, which were labeled correctly by the evaluated method, can be divided into two "true" groups:

Definition 5.7: True positives (TP) – the set of examples, which were labeled as positive by the method and are labeled as positive in the GS.

Definition 5.8: True negatives (TN) – the set of examples, which were labeled as negative by the method and are labeled as negative in the GS.

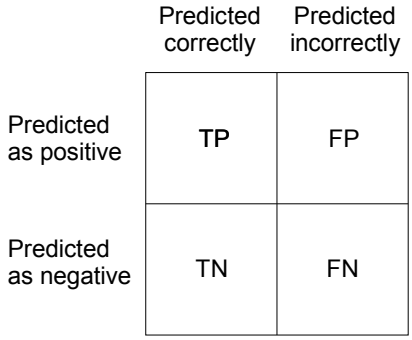

Figure 5.1: TP, FP, TN, FN Diagram.

Having defined the building blocks, we can continue with the definition of precision and recall.

Definition 5.9: Precision (P)

$$P = \frac{|TP|}{|TP|+|FP|}$$

Vertical bars denote cardinality, i.e. the number elements of the given set. E.g. | *M* | stands for *the number of elemenst of the set M*.

Definition 5.10: Recall (R)

$$R = \frac{|TP|}{|TP|+|FN|}$$

Definition 5.11: F-measure (F) is a combination of precision and recall, namely their weighted harmonic mean:

$$F = \frac{2 \cdot P \cdot R}{P + R}$$

"F-score" and "F_1 measure" are synonyms of "F-measure".

F-measure as a general reference measure in information extraction has been severely criticized by Pyysalo et al. (2008) and Airola et al. (2008). They have made an extensive comparative evaluation of two extraction methods on five distinct corpora using F-measure. Their results have revealed, that the choice of corpus has a stronger impact on the result in terms of F-measure than the choice of the extraction method. As a consequence of this, an F-measure performance of different methods cannot be meaningfully compared, when computed on different evaluation resources.

Airola et al. (2008) argue, that this is due to the fact, that F-measure is very sensitive to the underlying positive/negative pair distribution in the corpus:

> "[F]or example, halving the number of negative test examples is expected to approximately halve the number of false positives at a given recall point. Thus, the greater the fraction of true interactions in a corpus is, the easier it is to reach high performance in terms of F-score."

5.2 Area Under the Receiver Operating Characteristics Curve (AUC)

Airola et al. (2008) propose the *area under the receiver operating characteristics curve* (AUC) measure (Hanley and McNeil, 1982) as a replacement for F-measure. Unlike F-measure, AUC is invariant to the class distribution of the used dataset. The AUC measure corresponds to the probability that given a randomly chosen positive and negative example, the system will be able to correctly distinguish which one is which (Airola et al., 2008). The AUC values range between 0.5 (random classifier) and 1.0 (perfect classifier).

Airola et al. (2008) present the following definition of AUC:

Definition 5.12:

$$AUC = \frac{\sum_{i=1}^{m_+} \sum_{j=1}^{m_-} H(x_i - y_j)}{m_+ \cdot m_-}$$

where m_+ and m_- are the numbers of positive and negative examples, respectively, and $x_1, \ldots, x_{m_+}$ are real valued predictions of the system for the positive, and $y_1, \ldots, y_{m_+}$ for the negative examples, and H is defined as

Definition 5.13:

$$H(r) = \begin{cases} 1 & \text{if } r > 0 \\ 0.5 & \text{if } r = 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that "positive" and "negative" in this definition mean positive and negative in the gold standard (GS). Hence, to draw a link with the true/false positive/negative terminology of Section 5.1, the following relationships can be stated:

$$m_+ = |TP| + |FN|$$

$$m_- = |TN| + |FP|$$

The definition 5.12 of AUC is intended for systems, which produce real valued outputs in such a way, that the examples considered to be most likely to belong to the positive class should receive high output values and *vice versa*, the examples considered to be most likely to belong to the negative class should receive low output values. Note, that there are no bounds or normalization of the system outputs imposed by the definition. The only thing that matters is, if the difference of the outputs for a given pair of golden positive and golden negative is lower than, equal to, or greater than zero.

However, the situation is slightly simpler with binary classifiers which return only two fixed labels, e.g. "+1" and "-1" for positive and negative instances respectively. Within this theses, the SVM is viewed as such one. For such classifiers, the AUC definition can be reformulated to mention only the numbers of TNs, TPs, FPs and FNs, while the function H and the $[x_i, y_j]$ pairs will be used only implicitly.

Thus, suppose that the system under evaluation returns only two fixed labels "+1" for positive and "-1" for negative class. Then there are four possible valuations of the expression $x_i - y_j$ from the Definition 5.12:

1. x_i is a true positive and y_j is a true negative: $x_i - y_j = 1 - (-1) = 2$

2. x_i is a true positive and y_j is a false positive: $x_i - y_j = 1 - 1 = 0$
3. x_i is a false negative and y_j is a true negative: $x_i - y_j = -1 - (-1) = 0$
4. x_i is a false negative and y_j is a false positive: $x_i - y_j = -1 - 1 = -2$

These valuations also directly determine the value of $H(x_i - y_j)$:

1. x_i is a true positive and y_j is a true negative:
$$H(x_i - y_j) = H(1 - (-1)) = H(2) = 1$$
2. x_i is a true positive and y_j is a false positive:
$$H(x_i - y_j) = H(1 - 1) = H(0) = 0.5$$
3. x_i is a false neg. and y_j is a true neg.:
$$H(x_i - y_j) = H(-1 - (-1)) = H(0) = 0.5$$
4. x_i is a false neg. and y_j is a false positive:
$$H(x_i - y_j) = H(-1 - 1) = H(-2) = 0$$

Further, we know, how often each of the named valuations occurs in the explicit enumeration of all possible $[x_i, y_j]$:

1. there are $|TP| \cdot |TN|$ pairs with true positive x_i and true negative y_j.
2. there are $|TP| \cdot |FP|$ pairs with true positive x_i and false positive y_j
3. there are $|FN| \cdot |TN|$ pairs with false negative x_i and true negative y_j
4. there are $|FN| \cdot |FP|$ pairs with false negative x_i and false positive y_j

Hence we do not need to bother with explicit enumerating of all possible $[x_i, y_j]$ and summing their $H(x_i - y_j)$ together. Instead, it is enough to sum up the known numbers of the four possible evaluations multiplied with their H:

Definition 5.14:

$$AUC = \frac{|TP| \cdot |TN| \cdot H(2) + |TP| \cdot |FP| \cdot H(0) + |FN| \cdot |TN| \cdot H(0) + |FN| \cdot |FP| \cdot H(-2)}{m_+ \cdot m_-}$$

$$= \frac{|TP| \cdot |TN| + |TP| \cdot |FP| \cdot 0.5 + |FN| \cdot |TN| \cdot 0.5}{(|TP| + |FN|) \cdot (|TN| + |FP|)}$$

The goal of this reformulation is purely practical: it is easier just to count TNs, TPs, FPs and FNs, than to explicitly enumerate all $[x_i, y_j]$ pairs. This especially holds for those systems which already provide precision and recall as a measure of their performance. In such systems, the TN, TP, FP and FN numbers are usually already there for the computation of precision and recall. To compute AUC, these numbers only need to be plugged into the reformulated AUC formula. This is exactly the case for SVM^{light} software used in experiments which are documented later in this thesis. Only a minor modification to the original source code of Thorsten Joachims was needed to output the AUC together with precision, recall and F-measure.

5.3 Cross-Validation

To estimate the performance of a statistical classifier trained on an annotated resource, one needs another resource on which the accurateness of the classifier could be tested. This testing resource should be annotated in the same way as the training corpus, but it must not share any single example with the training corpus. This is important, because training a statistical model on the same (or overlapping) data as is used for proving its correctness can lead to overestimation of the method's performance (Hastie et al., 2001).

In the field of information extraction the annotated resources are very costly. Cross-validation is a method which allows for training and testing with sparse resources. It prescribes, how to repeatably partition the resource itself into a training and testing part. The partitioning rule usually produces some array of splits, on which the method is trained and tested sequentially several times. The final result of the evaluation (either in terms of F-measure or in terms of AUC) is obtained through averaging the individual runs' results (Hastie et al., 2001).

In *k*-fold cross-validation, the rule is to divide the resource into *k* parts of roughly equal size. The method under evaluation is then run *k*-times, each time taking one of the parts for testing and the union of the remaining parts for training.

Another variant of cross-validation is the *leave-one-out* strategy. It is actually a special case of *k*-fold cross-validation, namely the one having *k* equal to the number of examples of the whole resource.

Airola et al. (2008) drive the data independence provision to further subtleties: They argue, that the cross-validation splits must be done at least on the level of documents. They raise the

following objection against folds on the level of sentences or named entity pairs (i.e. below the document level):

> "[C]onsider two interaction candidates extracted from the same sentence, e.g. from a statement of the form "P1 and P2 [...] P3", where "[...]" is any statement of interaction or non-interaction. Due to the near-identity of contexts, a machine learning method will easily learn to predict that the label of the pair (P1, P3) should match that of (P2, P3). However, such "learning" will clearly not generalize. This approach must thus be considered invalid, because allowing pairs generated from the same sentences to appear in different folds leads to an information leak between the training and test sets."

In accordance with Airola et al. (2008) the classifiers presented within this thesis will be evaluated using 10-fold document-level cross-validation on the learning format corpora.

Antti Airola has kindly provided the splits which were used in Airola et al. (2008). Therefore, the results presented here will be directly comparable with the results presented in Airola et al. (2008). Airola et al. (2008) report, that the same splits for Aimed corpus were also used by Bunescu and Mooney (2006), Giuliano et al. (2006), Van Landeghem et al. (2008) and possibly some others.

5.4 Run Time Measurement

The times needed for the learning and for the actual classification are very important benchmarks of the evaluated kernels. We have used the SVM^{light} build-in time measurement. It is based on subtracting the `clock()` C function return values at the end of the program execution from those at the beginning of the program execution. This method makes the measured time dependent on the machine load during execution and on the processor speed. Further, it must be noted, that we have used two distinct machines with different – albeit similar – processors and possibly different load across time. Thus, the run times measured for ST, SST, PT and spectrum tree kernel implementations are only roughly comparable.

The times for kBSPS kernel are probably overoptimistic for higher q_{max}-s as the shortest path search and q-grams listing were accomplished within preprocessing. The time benchmarks for $q_{max} = 2$ are less biased as there is no overhead for listing bigrams – these are already there in the output of any dependency parser.

6 Experimental Setup

In this chapter, we provide detailed description of our experimental setup.

6.1 Preprocessing

6.1.1 Entity-Token Mapping

The Section 3.1.1 has demonstrated, that in the learning format, there is no guarantee of a one-to-one correspondence between named entity boundaries and token boundaries. However, the PPI extraction methods evaluated within this chapter make use of syntactic information and the smallest part of the sentence text, they can wok with, is token. This is also the finest level, where they can recognize named entities. So, for these methods to work, it is necessary to adopt some clear entity-token mapping concept.

Actually, the same problem must have been solved by Airola et al. (2008) for their all paths kernel to work. In the named paper, no details on such a mapping can be found. Antti Airola has revealed their solution in personal communication: every token, which at least partly overlaps with an entity, is marked as entity. This strategy will be adopted also in the experiments presented later in this chapter.

6.1.2 Entity Blinding

Entity blinding is a common preprocessing step in relation extraction systems. It is a replacement of all named entity occurrences in an example sentence with some generic string, e.g. _ENTITY_. Its effect is twofold: (1) it is a way to “inform” the classifier, where in the example sentence the named entities are located and (2) it ensures the generality of the learned model, as we are not interested in a model which can predict a presence of relation between some particular entities. Instead we want to obtain a model, which works for any entity in the given context.

As noted in Section 3.1, the examples used by a statistical classifier correspond to named entity pairs rather than sentences. So, out of a sentence with n pairs, exactly n examples are generated. For this reason, it is important to distinguish between the named entities which belong to the pair under consideration from those, which do not. So, two labels, e.g.

_ENTITY_1_ and _ENTITY_2_ are designated for use in place of the first and the second entity of the pair under consideration, respectively, and another special label, e.g. _ENTITY_ is used in place of named entities, which do not belong to the pair under consideration.

Due to the fact, that named entities may overlap on the level of tokens, the situation is possible, in which a single token overlaps both with the first and the second entity. In such cases, the proposal of Antti Airola (personal communication) will be followed, according to which a special label, e.g. _ENTITY_1_AND_2_ will be dedicated for this situation.

We demonstrate the entity blinding on an example. Consider the sentence d108.s909 from AImed corpus:

p53 transcriptional activation mediated by coactivators **TAFII40** and **TAFII60** .

There are 3 named entities annotated in this sentence; they are highlighted in **bold**. 3 distinct pairs can be formed out of these entities:

1. + [**p53**, **TAFII40**]
2. + [**p53**, **TAFII60**]
3. – [**TAFII40**, **TAFII60**]

The pairs 1 and 2 depict a relation, while pair 3 depicts no relation. Out of these 3 pairs, 3 blinded learning examples can be generated in the following way:

1. + **_ENTITY_1_** transcriptional activation mediated by coactivators **_ENTITY_2_** and **_ENTITY_** .
2. + **_ENTITY_1_** transcriptional activation mediated by coactivators **_ENTITY_** and **_ENTITY_2_** .
3. – **_ENTITY_** transcriptional activation mediated by coactivators **_ENTITY_1_** and **_ENTITY_2_** .

In this way, we have obtained the examples for training our kernels.

6.1.3 Special Preprocessing Steps for some Kernels

The kernels of other authors presented in Section 4.1 are defined for trees. Hence, it is not possible to apply them directly to dependency graphs contained in the learning format

corpora. Thus, to use these corpora together with the named kernels, constituent trees must be generated in a special preprocessing step. Chaniak-Johnson-McCloski Parser (see Chapter 10 Software and Corpora) was used for this. Further, the terminal symbols of the constituent tree parses were mapped to the charOffsets of the original sentence text. This was necessary for the blinding of named entities in the constituent tree parse. Finally, the blinded parses were formated so that they comply with the expectations of the given kernel's implementation.

For the kBSPS kernel introduced in Section 4.2.4, the surface tokens were stemmed using Porter stemmer (Porter, 1980). This has improved AUC by about 4%.

6.2 Parameters and their Optimization

As we have already seen in Chapters 2 and 4, both SVM and the evaluated kernels have several parameters which have substantial impact on the overall performance of the system. To determine the best possible performance for the given kernel, optimal values for these parameters must be found. To accomplish this, we used a simple parameter space search. For each parameter several values across its domain were chosen and then all parameter value combinations were tested. Which values were explored for which kernels can be seen in Table 6.1. Later in Chapter 7, only the results for the best performing parameter set are presented.

SVM Parameters

c – characterization of the soft margin as introduced in Chapter 2; actually a trade-off between training error and margin (from SVM^{light} documentation).

j – cost-factor, by which training errors on positive examples outweight errors on negative examples (from SVM^{light} documentation).

Kernel Parameters

λ – decay factor for the length of the child sequences, applicable to ST, SST and PT kernels. It penalizes subtrees built on child subsequences that contain gaps (Moschitti, 2006a).

μ – decay factor for the height of the tree, applicable to PT kernel. It penalizes larger trees (Moschitti, 2006a).

q – length of the counted q-grams, applicable to spectrum tree kernel.

Normalization – all of the evaluated kernels allowed for normalizing the counts to an interval between 0 and 1. None of the kernels showed better performance with normalization turned on in random tests. Therefore the normalization was always off.

kBSPS Kernel Parameters

minq, ***maxq*** – minimum and maximum length of the counted *q*-grams.

t – the matching strategy

If the matching strategy t is "t(olerant)", there are the following additional parameters applicable to the kBSPS:

k – the width of the shortest path context.

l – the score for a surface token match.

e – the score for a named entity match.

d – the score for a dependency match.

Kernel		Param.	Explored Values	
			HPRD50, IEPA, LLL	Almed
ST		*c*	0.015625, 0.0625, 0.25, 1, 4, 8, 16, 64, 128, 256, 512	8, 16
		j	0.5, 1, 2	1, 2
		λ	0.2, 0.4, 0.6, 0.8, 1.0	0.4, 0.6, 0.8
SST		*c*	0.015625, 0.0625, 0.25, 1, 4, 8, 16, 64, 128, 256, 512	1, 8, 16, 32, 64, 128
		j	0.5, 1, 2	1, 2
		λ	0.2, 0.4, 0.6, 0.8, 1.0	0.4, 0.6, 0.8
PT		*c*	0.015625, 0.0625, 0.25, 1, 4, 8, 16, 64, 128, 256, 512	8, 16
		j	0.5, 1, 2	1, 2
		λ	0.2, 0.4, 0.6, 0.8, 1.0	0.4, 0.6, 0.8
		μ	0.2, 0.4, 0.6, 0.8, 1.0	0.4, 0.6, 0.8
SpT		*c*	0.015625, 0.0625, 0.25, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512	0.015625, 0.0625, 0.25, 1, 4, 16, 64, 256
		j	0.5, 1, 2	0.5, 1, 2
		q	2, 3	2, 3
kBSPS		*c*	0.015625, 0.0625, 0.25, 1	0.015625, 0.0625, 0.25, 1
		j	0.5, 0.8, 1, 1.2, 2	0.8, 1, 1.2
		qmin	1	1
		qmax	2, 3	2, 3
		k	0, 1	0, 1
		t	t(olerant), l(evenshtein), e(xact)	t, l, e
	for *t* = t	*l*	1, 3, 6	1, 3, 6
		e	1, 3, 6	1, 3, 6
		d	1, 3, 6	1, 3, 6

Table 6.1: Parameter spaces for the individual kernels

6.3 Kernel Implementations

For the ST, SST and PT kernels introduced in Section 4.1, the implementation of Alessandro Moschitti was used (see Chapter 10 Software and Corpora). For the evaluation of the spectrum tree kernel, Tetsui Kuboyama has kindly provided his Ruby implementation, which we have recoded in C so that it could be integrated into SVM^{light}.

The kBSPS kernel was implemented in C by us. Its implementation is rather simplistic than sophisticated. The shortest path search as well as *q*-gram listing are accomplished in a preprocessing step. The graph labels are replaced by numbers for faster comparisons. The

time of these preprocessing steps is not included in the learning times in Tables 7.1 and 7.4.

7 Experimental Results

We present our experimental results in four tables. The best results we could reach for the given corpus and kernel combination can be seen in Table 7.1. The parameter sets which were used in the experiments listed in Table 7.1 are viewable in Table 7.2. Table 7.3 compares the AUC and F-measure value ranges from Table 7.1. Finally, in Table 7.4, we demonstrate some properties of our kBSPS kernel.

Experim. ID	Corpus	Kernel	AUC	P	R	F	Learn Sec.	Classif. Sec.	SV Num.
1	AImed	ST	0.611	0.399	0.333	0.358	229.4	18.2	2581
2	AImed	SST	0.584	0.481	0.219	0.299	210.2	19.3	2918
3	AImed	PT	0.579	0.520	0.202	0.284	2069.5	188.3	2371
4	AImed	SpT	0.598	0.314	0.389	0.340	199.9	20.7	2663
5	AImed	kBSPS	0.672	0.494	0.447	0.461	18.5	1.1	1802
6	AImed	All-paths	0.848	**0.529**	**0.618**	0.564			
7	AImed	Miwa	**0.879**			**0.635**			
8	HPRD50	ST	0.617	0.581	0.458	0.494	0.7	0.1	286
9	HPRD50	SST	0.607	0.670	0.336	0.423	0.5	0.1	337
10	HPRD50	PT	0.627	0.636	0.417	0.477	12.1	1.9	359
11	HPRD50	SpT	0.625	0.495	0.721	0.579	1.1	0.3	274
12	HPRD50	kBSPS	0.769	**0.667**	**0.802**	**0.709**	0.1	0.0	233
13	HPRD50	All-paths	**0.797**	0.643	0.658	0.634			
14	IEPA	ST	0.691	0.650	0.604	0.620	3.9	0.4	521
15	IEPA	SST	0.679	0.650	0.558	0.591	3.1	0.4	542
16	IEPA	PT	0.677	0.669	0.545	0.594	46.2	7.5	582
17	IEPA	SpT	0.680	0.626	0.618	0.615	4.0	0.9	548
18	IEPA	kBSPS	0.758	**0.704**	0.730	0.708	0.5	0.0	444
19	IEPA	All-paths	**0.851**	0.696	**0.827**	**0.751**			
20	LLL	ST	0.724	0.803	0.622	0.685	1.1	0.0	201
21	LLL	SST	0.690	0.742	0.665	0.682	1.7	0.0	225
22	LLL	PT	0.723	0.743	0.789	0.748	28.7	1.2	214
23	LLL	SpT	0.628	0.697	0.621	0.606	1.7	0.3	203
24	LLL	kBSPS	0.785	**0.768**	**0.918**	**0.822**	0.1	0.0	179
25	LLL	All-paths	**0.834**	0.725	0.872	0.768			

Table 7.1: The best results for the individual corpus and kernel combinations.

Except for performance benchmarks in terms of AUC, precision, recall and F-measure, Tables 7.1 and 7.4 contain average times needed for learning and for the actual classification. The last column in these tables presents the number of support vectors of the given classifier. This number can be interpreted as a measure of the quality of the classifier or a measure of the underlying kernel's ability to separate the data (lower is better in both cases).

For comparison, we also cite benchmarks for two state-of-the-art kernels in Table 7.1. These are the all-paths graph kernel of Airola et al. (2008) and the combined kernel of Miwa et al (2008). We have introduced them briefly in Section 4.1.5.

Experim. ID	Corpus	Kernel	Parameter Set
1	Almed	ST	c=16, j=2, λ=0.8
2	Almed	SST	c=8, j=2, λ=0.4
3	Almed	PT	c=16, j=2, λ=0.4, μ=0.6
4	Almed	SpT	c=0.0625, j=2, q=2
5	Almed	kBSPS	c=0.25, j=1.2, qmin=1, qmax=2, k=0, t=t, l=3, e=6, d=1
8	HPRD50	ST	c=8, j=1, λ=0.6
9	HPRD50	SST	c=1, j=1, λ=0.4
10	HPRD50	PT	c=4, j=2, λ=0.8, μ=0.6
11	HPRD50	SpT	c=0.0625, j=2, q=2
12	HPRD50	kBSPS	c=0.015625, j=2, qmin=1, qmax=3, k=0, t=t, l=1, e=1, d=6
14	IEPA	ST	c=128, j=0.5, λ=0.8
15	IEPA	SST	c=16, j=0.5, λ=0.2
16	IEPA	PT	c=8, j=2, λ=0.4, μ=0.6
17	IEPA	SpT	c=0.03125, j=1, q=2
18	IEPA	kBSPS	c=0.25, j=0.5, qmin=1, qmax=2, k=0, t=t, l=1, e=6, d=6
20	LLL	ST	c=64, j=0.5, λ=0.2
21	LLL	SST	c=128, j=0.5, λ=0.2
22	LLL	PT	c=128, j=0.5, λ=0.2, μ=0.6
23	LLL	SpT	c=2, j=0.5, q=3

Table 7.2: Parameter sets which were used for the computation of the results presented in Table 7.1.

In terms of AUC, all-paths graph kernel of Airola et al. (2008) and the combined kernel of

Miwa et al. (2008) are clearly superior to all kernels evaluated here. On the other hand, again in terms of AUC, our kBSPS kernel performs better than the four kernels from the literature.

On AImed, which was the biggest from the corpora used here, the ordering given by AUC is the same as the ordering given by F-measure. However, there are exceptions to this rule on the other three smaller corpora: On HPRD50, the AUC ordering

SST, ST, SpT, PT, kBSPS, All-paths

does not match the one given by F-measure:

SST, PT, ST, SpT, All-paths, kBSPS

Similarly, on LLL (the smallest of the used corpora), the ranking

SpT, SST, PT, ST, kBSPS, All-paths

given by AUC does not match the ranking given by F-measure:

SpT, SST, ST, PT, All-paths, kBSPS

In the IEPA rankings, there is only a minor mismatch between the position of SST and PT, the difference between the AUC scores and F-measure scores being very small (0.002 and 0.003 respectively).

Notably, in terms of F-measure on the two smallest corpora (HPRD50 an LLL), our kBSPS kernel performs better than the all-paths graph kernel.

Our results seem to justify the claim that AUC is more suitable for measuring the accuracy of classifiers than F-measure. The AUC results on different corpora are much more stable than the F-measure results. This can be clearly seen in Table 7.3 where we list the differences between best and the worst results of the individual kernels in terms of both AUC and F-measure. While AUC differences range between 0.054 and 0.144 (averagely 0.102) the F-measure differences range between 0.204 and 0.464 (average 0.34).

Kernel	AUC best	AUC worst	AUC diff	F best	F worst	F diff
ST	0.724	0.611	0.113	0.685	0.358	0.327
SST	0.690	0.584	0.106	0.682	0.299	0.382
PT	0.723	0.579	**0.144**	0.748	0.284	**0.464**
SpT	0.680	0.598	0.082	0.615	0.340	0.275
kBSPS	0.785	0.672	0.113	0.822	0.461	0.361
All-paths	0.851	0.797	**0.054**	0.768	0.564	**0.204**
average			0.102			0.34

Table 7.3: Comparison of the stability of AUC and F-measure.

Despite some positive bias (see Section 5.4), the kBSPS kernel is very fast. We believe that it would sustain a comparison with the other time benchmarks presented here even when the time needed for the preprocessing steps would be added to times presented in Table 7.1.

Further, the time benchmarks seem to disagree with Moschitti's (2006a) claim that PT complexity is lower than the one of SST. SST is an order of magnitude faster than PT. On the other hand we can approve another observation of Moschitti (2006a) that SST performs very similarly to PT. Hence, allowing additional structures in PT does not seem to bring any substantial advantage against SST.

ST performs surprisingly well on the most of the used corpora. Though working with a fairly limited tree substructure space, on AImed, IEPA and LLL, it has outperformed SST, PT and SpT kernels which make use of much larger spaces of substructures.

Experim. ID	t	Other Parameters	AUC	P	R	F	Learn Sec.	Classif. Sec.	SV Num.
26	e	c=1, j=1.2, k=0, qmin=1, qmax=2	0.617	0.420	0.340	0.367	16.2	1.2	1919
27	l	c=1, j=1.2, k=0, qmin=1, qmax=2	0.655	0.440	0.433	0.429	60.1	3.4	1873
28	t	c=0.25, j=1.2, k=0, qmin=1, qmax=3, l=1, e=1, d=1	0.664	0.532	0.408	0.454	26.1	1.9	1882
29	t	c=0.0625, j=1.2, k=0, qmin=1, qmax=3, l=3, e=6, d=6	0.667	0.535	0.415	0.460	24.0	1.7	1872
5	t	c=0.25, j=1.2, k=0, qmin=1, qmax=2, l=3, e=6, d=1	0.672	0.494	0.447	0.461	18.5	1.1	1802

Table 7.4: Benchmarks for several variants of kBSPS kernel on the AImed corpus. *t* is the matching strategy with values e(xact), l(evenshtein) and t(olerant). For other parameters see Section 6.2.

Impact of several kBSPS kernel features on its performance can be seen in Table 7.4. Firstly, when comparing the Experiment 26 with Experiments 27 and 5, the impact of fuzzy matching (see Section 4.2.2.1) is clearly recognizable. In experiment 26, the best AUC result was reached with exact matching, while the experiments 27 and 5 show the best results with Levenshtein and tolerant matching, respectively. Tolerant matching outperforms Levenshtein both in terms of accuracy and speed. Note that though the time benchmarks in Table 7.4 are also positively biased (see Section 5.4), they are all biased in the same way. Thus, they are comparable among each other.

Secondly, the difference between the Experiments 28 and 5 of Table 7.4 reveals the influence of different scores for Ls, Es and Ds in tolerant matching (see Section 4.2.2.1.1). In experiment 28, L, E and D-matches were scored equally with 1, whereas in experiment 5 different (optimal) scores were used. The difference between the experiments 28 and 5 in terms of AUC is not large. However, note that experiment 28 uses $q_{max} = 3$ which makes its classifier slower.

Thirdly, Experiments 29 and 5 of Table 7.4 can be used to see the impact of using $q_{max} = 3$ rather than $q_{max} = 2$. As already noted the classifiers with $q_{max} = 3$ are slower than those with

q_{max} = 2. While the difference between q_{max} = 3 and q_{max} = 2 in terms of AUC does not seem to be significant, there is a recognizable difference in precision and recall: q_{max} = 3 results in higher precision

Fourthly, the comparison of Experiment 4 of Table 7.1 with Experiment 26 of Table 7.4 can be interpreted as showing contribution of using dependency graphs instead of constituent trees. This is due to the fact that both experiments were performed with very similar kernels while using different syntax representations: constituent trees were used in Experiment 4 of Table 7.1 and dependency graphs were used in Experiment 26 of Table 7.4. The AUC improvement by 0.019 is less than we have originally expected; it has later led us to the idea of fuzzy matching.

Finally, both experiments 5 and 29 of Table 7.4 were performed with k = 0. This means, that selecting *only* the graph elements on the shortest path between the candidate entities for the q-gram production has brought the best results. Thus, expanding the shortest path with nodes neighboring with the shortest path (i.e. using e.g. k = 1) does not bring any improvement.

8 Conclusion

We have surveyed several relation extraction methods which make use of syntactic information. We have experimented with 4 kernel functions from the literature and we have designed some kernels ourselves. We have evaluated our most successful kernel and the four kernels from the literature using a methodology which made it possible to compare our results with those which are nowadays considered state-of-the-art. Our results show that a fairly simple method can outperform the sophisticated methods from literature. Compared with the state-of-the-art benchmarks, our method lacks in accuracy. However, due to its simplicity, our method is very fast.

Our method can be possibly improved by taking special care for self-interactions as done by Miwa et al. (2008). Similarly, we consider the idea to experiment with different weighting of short and long q-grams worth of experimenting.

9 References

Airola, A., Pyysalo, S., Björne, J., Pahikkala, T., Ginter, F., Salakoski, T. (2008): All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. BMC Bioinformatics 2008, 9(Suppl 11):S2, published: 19 November 2008 doi:10.1186/1471-2105-9-S11-S2

Bader, G.D., Betel, D., Hogue, C.W. (2003): BIND: the Biomolecular Interaction Network Database. Nucleic Acids Res. 2003 Jan 1;31(1):248-50.

Blaschke, C., Andrade, M.A., Ouzounis, C., Valencia, A. (1999): Automatic extraction of biological information from scientific text: protein-protein interactions. Proc Int Conf Intell Syst Mol Biol. 1999:60-7.

Bunescu, R., Mooney, R. (2005): A shortest path dependency kernel for relation extraction. In Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing Association for Computational Linguistics; 2005:724-731.

Bunescu, R., Ge, R., Kate, R.J., Marcotte, E.M., Mooney, R.J., Ramani, A.K., Wong, Y.W. (2005b): Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. Artif Intell Med, Summarization and Information Extraction from Medical Documents, 33(2):139-155.

Bunescu, R., Mooney, R. (2006): Subsequence Kernels for Relation Extraction. In Advances in Neural Information Processing Systems 18, MIT Press; 2006:171-178.

Charniak, E. (2000): A Maximum-Entropy-Inspired Parser. NAACL'00, pp. 132-139.

Charniak, E., Johnson, M. (2005): Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. ACL'05.

Charniak, E., Lease, M. (2005): Parsing biomedical literature. In Proceedings of IJCNLP'05, pages 58–69.

Chatr-aryamontri A., Ceol, A., Palazzi, L.M., Nardelli, G., Schneider, M.V., Castagnoli, L., Cesareni, G. (20069): MINT: the Molecular INTeraction database. Nucleic Acids Research, 2007, Vol. 35, Database issue D572-D574

Clegg, A.B., Shepherd, A.J. (2007): Benchmarking natural-language parsers for biological applications using dependency graphs. BMC Bioinformatics. 2007 Jan 25;8:24.

Collins, M., Duffy, N. (2001): Convolution Kernels for Natural Language. In Proc. of Neural Information Processing Systems (NIPS'2001).

Cortes, C., Vapnik, V. (1995): Support Vector Networks, Machine Learning 273-297

Culotta, A., Sorensen, J. (2004): Dependency tree kernels for relation extraction. In Proceedings of the 42nd Annual Meeting on Association For Computational Linguistics (Barcelona, Spain, July 21 - 26, 2004). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 423. DOI= http://dx.doi.org/10.3115/1218955.1219009

Ding, J., Berleant, D., Nettleton, D., Wurtele, E. (2002): Mining MEDLINE: abstracts, sentences, or phrases? Proceedings of PSB'02 2002:326-337.

Donaldson, I., Martin, J., de Bruijn, B., Wolting, C., Lay, V., Tuekam, B., Zhang, S., Baskin, B., Bader, G.D., Michalickova, K., Pawson, T., Hogue, C.W. (2003). PreBIND and Textomy--mining the biomedical literature for protein-protein interactions using a support vector machine. BMC Bioinformatics 4(1): 11.

Fukunaga, K. (1990): Introduction to Statistical Pattern Recognition. Academic Press.

Fundel, K., Küffner, R., Zimmer, R. (2007): RelEx--relation extraction using dependency parse trees. Bioinformatics. 2007 Feb 1;23(3):365-71. Epub 2006 Dec 1. PMID: 17142812

Giuliano, C., Lavelli, A., Romano, L. (2006): Exploiting Shallow Linguistic Information for Relation Extraction From Biomedical Literature. Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics 2006.

Hakenberg, J., Plake, C., Leser, U., Kirsch, H., Rebholz-Schuhmann, D. (2005): LLL'05 Challenge: Genic Interaction Extraction with Alignments and Finite State Automata. Proc Learning Language in Logic Workshop (LLL'05) at ICML 2005, pp. 38-45. Bonn, Germany.

Han, J., Kamber, M. (2000): Concepts and Techniques. Data Mining, Morgan Kaufmann.

Hanley, J.A., McNeil, B.J. (1982): The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology. 1982 Apr;143(1):29-36.

Hastie, T., Tibshirani, R. and Friedman, J. H. (2001). Elements of Statistical Learning.

Springer.

Haussler, D. (1999): Convolution kernels on discrete structures. Technical Report, University of Santa Cruz, UCSC-CRL-99-10

Joachims, T. (1999): Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press.

Kim, S., Yoon, J. and Yang, J. (2008): Kernel approaches for genic interaction extraction. Bioinformatics, 2008. 24(1): p. 118-26.

Klein, D., Manning, Ch. D. (2003a): Fast Exact Inference with a Factored Model for Natural Language Parsing. In Advances in Neural Information Processing Systems 15 (NIPS 2002), Cambridge, MA: MIT Press, pp. 3-10.

Klein, D., Manning, Ch. D. (2003b): Accurate Unlexicalized Parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

Kuboyama, T., Kashima, H., Aoki-Kinoshita, K.F., Hirata, K., Yasuda, H. (2007): A Spectrum Tree Kernel, Journal of Japanese Society of Artificial Intelligence, Vol.22, No.2, 2007.

Van Landeghem, S., Saeys, Y., Van de Peer, Y., De Baets, B. (2008): Extracting Protein-Protein Interactions from Text using Rich Feature Vectors and Feature Selection. Proceedings of the Third International Symposium on Semantic Mining in Biomedicine 2008:77-84.

Levenshtein, V. I. (1966): Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (1966):707–710.

Lodhi, H., Shawe-Taylor, J., Cristianini, N., Watkins, Ch. J. C. H. (2000): Text classification using string kernels. In NIPS, pages 563–569.

Marcus, M.P., Santorini B., Marcinkiewicz M.A. (1994): Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics 1994, 19(2):313-330.

de Marneffe, M.C., MacCartney, B., Manning, C.D. (2006): Generating Typed Dependency Parses from Phrase Structure Parses. In Proceedings of 5th International Conference on Language Resources and Evaluation (LREC2006) Genoa, Italy; 2006.

de Marneffe, M.C., Manning, C.D. (2008): Stanford typed dependencies manual. Technichal report. http://nlp.stanford.edu/software/dependencies_manual.pdf last visited 2008-04-09.

McClosky, D., Charniak, E. (2008) Self-Training for Biomedical Parsing. Proceedings of the Association for Computational Linguistics (ACL 2008, short papers), Columbus, Ohio

Miyao, Y., Sagae, K., Sætre, R., Matsuzaki, T., Tsujii, J. (2008): Evaluating Contributions of Natural Language Parsers to Protein-Protein Interaction Extraction. Bioinformatics. 2008 Dec 9. PMID: 19073593

Miwa, M., Sætre, R., Miyao, Y., Ohta, T., Tsujii, J. (2008): Combining Multiple Layers of Syntactic Information for Protein-Protein Interaction Extraction. Proceedings of the Third International Symposium on Semantic Mining in Biomedicine 2008:101-108.

Moschitti, A. (2006a): Efficient convolution kernels for dependency and constituent syntactic trees. In Proceedings of The 17th European Conference on Machine Learning, pages 318-329, Berlin, Germany.

Moschitti, A. (2006b): Making tree kernels practical for natural language learning. In Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL2006), pages 113-120, Treato, Italy.

Nédellec, C. (2005): Learning language in logic - genic interaction extraction challenge. Proceedings of LLL'05 2005:31-37.

Özgür, A., Vu, T., Erkan, G., and Radev, D. R. (2008): Identifying gene-disease associations using centrality on a literature mined gene-interaction network. Bioinformatics 24, 13 (Jul. 2008), i277-i285. DOI= http://dx.doi.org/10.1093/bioinformatics/btn182

Porter, M.F. (1980): An Algorithm for Suffix Stripping, Program, 14(3): 130-137

Pyysalo, S., Ginter, F., Heimonen, J., Björne, J., Boberg, J., Järvinen, J., Salakoski, T. (2007): BioInfer: A Corpus for Information Extraction in the Biomedical Domain. BMC Bioinformatics 2007, 8(50).

Pyysalo, S., Airola, A., Heimonen, J., Björne, J., Ginter, F., Salakoski, T. (2008): Comparative analysis of five protein-protein interaction corpora. BMC Bioinformatics. 2008 Apr 11;9 Suppl 3:S6. PMID: 18426551

van Rijsbergen, C.V. (1979): Information Retrieval. London; Boston. Butterworth, 2nd Edition.

Vapnik, V. (1995): The Nature of Statistical Learning Theory. Springer-Verlag.

Vishwanathan, S., Smola, A. (2002): Fast kernels on strings and trees. In: Proceedings of NIPS.

Wittgenstein, L. (1961 [1921]): Tractatus-Logico Philosophicus. Routledge & Kegan Paul.

Xenarios, I., Rice, D.W., Salwinski, L., Baron, M.K., Marcotte, E.M., Eisenberg, D. (2008): DIP: the database of interacting proteins. Nucleic Acids Res. 2000 Jan 1;28(1):289-91.

Zanzoni, A., Montecchi-Palazzi, L., Quondam, M., Ausiello, G., Helmer-Citterich, M., Cesareni, G. (2002): MINT: a Molecular INTeraction database. FEBS Lett. 2002 Feb 20; 513(1):135-40.

Zelenko, D., Aone, C., Richardella, A. (2003): Kernel methods for relation extraction. Journal of Machine Learning Research 2003, 3:1083-1106.

10 Software and Corpora

SVM^{light}: http://svmlight.joachims.org/, maintained by Thorsten Joachims, last visited 2009-03-16; see also Joachims (1999).

Tree Kernels In SVM-light: http://dit.unitn.it/moschitti/Tree-Kernel.htm, maintained by Alessandro Moschitti from University of Trento, last visited: 2009-03-28; see also Moschitti (2006a and 2006b). The "1.5-to-be-released" version used here is not publicly available yet – Alessandro Moschitti has kindly sent it to us via e-mail.

Stanford Tools: http://nlp.stanford.edu/software/lex-parser.shtml maintained by The Stanford Natural Language Processing Group, last visited 2009-04-07; see also de Marneffe et al (2006).

Charniak-Lease Parser: see next and Charniak and Lease (2005).

Charniak-Johnson-McCloski Parser: actually a Charniak-Johnson parser (Charniak and Johnson, 2005) with David McCloski's model trained on biomedical resources (McClosky and Charniak, 2008); http://bllip.cs.brown.edu/resources.shtml maintained by Brown Laboratory for Linguistic Information Processing (BLLIP), last visited 2009-04-07

Five corpora (AImed, BioInfer, HPRD50, IEPA and LLL) in the learning format: http://mars.cs.utu.fi/PPICorpora/GraphKernel.html, maintained by Department of Information Technology of University of Turku, last visited 2009-03-16; see also Pyysalo et al. (2008).