The Role of Syntactic Features in Protein Interaction Extraction

Timur Fayruzov[1], Martine De Cock[1], Chris Cornelis[1], Veronique Hoste[1,2]
[1]Ghent University, Department of Applied Mathematics and Computer Science
Krijgslaan 281 (S9), 9000 Gent, Belgium
[2]University College Ghent, School of Translation Studies
Groot-Brittanniëlaan 45, 9000 Gent, Belgium
Timur.Fayruzov@ugent.be, Martine.DeCock@ugent.be
Chris.Cornelis@ugent.be, Veronique.Hoste@hogent.be

ABSTRACT

Most approaches for protein interaction mining from biomedical texts use both lexical and syntactic features. However, the individual impact of these two kinds of features on the effectiveness of the mining process has not yet been thoroughly studied. In this work we perform such a study on a recently published state of the art support vector machine approach that uses both lexical and syntactic features. To this end, we strip this approach down to an algorithm that uses only a subset of the initial syntactic features. Next, we compare the original and the stripped-down method by evaluating them on 5 benchmark datasets as well as by performing 5 additional cross-dataset experiments. Although the original method exploits a very rich feature set including words, parts-of-speech and grammatical relations, it is not significantly better than the stripped-down version; in fact, the former does not even consistently outperform the latter.

1. INTRODUCTION

More and more relevant information on protein interactions is becoming available on the web, not only in specialized structured databases such as IntAct[1] and ontological resources such as the Gene Ontology[2], but also in less structured literature databases such as MEDLINE[3]. The former resources are built and maintained manually and thus require a great deal of knowledge and labor intensive maintenance to stay synchronized with the latest research findings in molecular biology, while the latter is always up to date. However, it is not straightforward to query and find specific information in a text database such as MEDLINE, since it is loosely structured and provides very few metadata about its content.

[1]http://www.ebi.ac.uk/intact/site/index.jsf
[2]http://www.geneontology.org/
[3]http://www.ncbi.nlm.nih.gov/

Therefore over the last decade the need for efficient information extraction techniques in the biomedical domain has become more and more apparent. Many researchers have been tackling the problem with different approaches relying on training corpora as well as on external sources such as thesauri, synonym dictionaries etc. Recent advances in language parsing and constantly growing computational facilities allow to encorporate in the mining process not only explicit data extracted from the text, such as words and parts-of-speech (POS), but also more complex full syntactic trees. Therefore, it is not surprising that many researchers try to employ this kind of information for the task of protein interaction mining. Most of the current state of the art approaches enrich lexical information (words) with shallow syntactic information (e.g. POS) and/or deep syntactic information (grammatical structures), aiming to achieve better results (see e.g. [1, 3, 6, 10, 11, 14, 15, 19, 21, 22, 23]).

The individual impacts of lexical features, shallow syntactic features and deep syntactic features have received little attention in the literature so far. The question what the individual contributions of the different kinds of features are, is however worthwhile investigating because (1) when two kinds of features have a substitute rather than a complementary effect, one of them can be dropped to obtain a more computationally efficient method, and (2) dropping one kind of features might make the mining algorithm more robust. The latter claim is especially plausible for lexical features since lexicons tend to be subdomain specific. Hence, training a system on the lexical features of a corpus in one biomedical subdomain could limit the applicability of this system for texts in another subdomain. A system that uses only syntactic features has the potential to be more robust since grammatical information is more general than lexical information across different subdomains. Most existing work however does not seem to take this into consideration as newly proposed methods are typically trained and tested on the same corpus, e.g. with a cross-validation approach, benefiting from the advantage of a similar lexicon for the training and test data.

Using a C4.5 and a BayesNet classifier relying only on deep syntactic information, in [9] we were able to obtain results comparable with state of the art approaches for protein interaction mining, including in a modest cross-dataset exper-

iment involving the LLL [16] and AIMed [2] datasets. This observation encouraged us to engage in a more systematic study of the individual effects of lexical and syntactic features, which we report on in this paper. In the current work, instead of a C4.5 or a BayesNet classifier, we use a support vector machine (SVM) classifier with kernels, because at the moment this kind seems to be the most popular method for relation mining systems, including protein interaction extraction (see e.g. [3, 6, 11, 15, 19, 21, 23]). Furthermore, instead of proposing our own kernel and comparing this with others, we start from an existing approach [15] that relies on lexical, shallow and deep syntactic features, and strip it of the former two to obtain an approach purely based on deep syntactic information. This setting allows us to compare the net effect of adding lexical and shallow syntactic information on top of deep syntactic features.

To the best of our knowledge, besides us, only [19, 21, 22] have looked into the impact of syntactic features in addition to lexical ones for the protein interaction extraction task (all in the context of SVMs). Shallow syntactic features such as POS are reported not to increase the performance of the classifier in [22], while omitting a rich subset of the lexical features leads to a poor performance of the classifier in [19]. Neither one has however studied how much performance can be obtained by using *only* syntactic features. The closest to our work is [21] who have compared the performance of an interaction extraction system using only lexical features versus using only shallow and deep syntactic features. In this work we go yet one step further in dropping also shallow syntactic information, thus looking into the performance that can still be achieved when using only grammatical relations for the interaction extraction task.

The main contributions of this paper are: (1) an extensive validation of a recently proposed SVM approach [15] on 5 datasets and 5 additional cross-dataset experiments, using both the F-measure and the AUC-measure as evaluation metrics; (2) a comparison of the performance of this system with a stripped down-version taking into account only deep syntactic information, which demonstrates that omitting lexical and shallow syntactic features does not significantly decrease the classifier's performance and sometimes even results in an improvement.

In Section 2 we recall the SVM approach proposed by [15] and we introduce the stripped-down version that uses only grammatical relations. To illustrate that the method from [15] is representative for current developments in the field, we give an overview of related approaches in Section 3. Next we describe the datasets in Section 4 and the experimental results in Section 5, and finally we conclude in Section 6.

2. INTERACTION EXTRACTION

2.1 Problem description

The task of protein interaction extraction from natural language texts is concerned with identifying whether a sentence containing two protein names describes an interaction between those proteins or not. Note that it is assumed that all protein names are recognized and annotated in advance by a named entity recognizer (NER). Named entity recognition is a whole separate area of text mining which is commonly treated separately from the relation mining task (see e.g. [3, 9, 10, 14]).

Example 1. The following sentence contains 4 protein names, giving rise to 6 protein pairs that potentially correspond to an interaction: "In the $\underline{\text{shaA}}_1$ mutant, $\underline{\text{sigma(H)}}_2$-dependent expression of $\underline{\text{spo0A}}_3$ and $\underline{\text{spoVG}}_4$ at an early stage of sporulation was sensitive to external NaCl." The protein interaction extraction algorithm is expected to return (only) the pairs $(2, 3)$ and $(2, 4)$ as the sentence describes (only) an interaction of *sigma(H)* with *spo0A* and an interaction of *sigma(H)* with *spoVG*. □

From the example above it is clear that protein interaction extraction can be thought of as an information retrieval task, with the interacting protein pairs being the relevant instances to retrieve. Alternatively, it can be thought of as a classification task since the point is to decide whether a pair of proteins is a positive instance (corresponding to an interaction) or a negative instance (not corresponding to an interaction). The learning task is typically supervised in the sense that the classifier is trained on a corpus annotated with protein names and interactions.

Virtually all approaches treat protein interaction extraction in this way, hence reducing the problem to a proper choice of a machine learning algorithm and a formal representation of relevant information from the sentence to be used as input for the system. We give a (non-exhaustive) overview of existing approaches in Section 3. For our experimental purposes in this paper we focus on recently published work of Kim et al. [15], i.e. a SVM approach that is reported to achieve state of the art results on the LLL dataset. One of the interesting aspects of the work of Kim et al. is that they consider two formal representations of a sentence that contains a protein pair, namely a subtree of the dependency tree and a flat feature vector. Unlike in [21] where the role of lexical versus shallow and deep syntactic features in Kim et al.'s feature vector based approach was explored further, in this paper we focus on the use of the dependency tree format like in [9].

2.2 Tree representations

The dependency tree for a sentence is a tree containing grammatical and lexical information about the sentence. All nodes of the dependency tree are words of the sentence while edges between nodes represent syntactic dependencies. In typed dependency trees, edges are labeled with syntactic functions (e.g., subj, obj). In Figure 1a the dependency tree for the sentence from Example 2 is shown.

Example 2. The following sentence describes an interaction between the proteins *sigma F* and *sigma E*: "$\underline{\text{Sigma F}}_1$ activity regulates the processing of $\underline{\text{sigma E}}_2$ within the mother cell compartment." The dependency tree for this sentence is depicted in Figure 1a, while Figure 1b depicts the subtree of the dependency tree corresponding to the shortest path between *sigma F* and *sigma E*. Both protein names are replaced with dummy strings NE_1 and NE_2 in order to generalize the interaction pattern. □

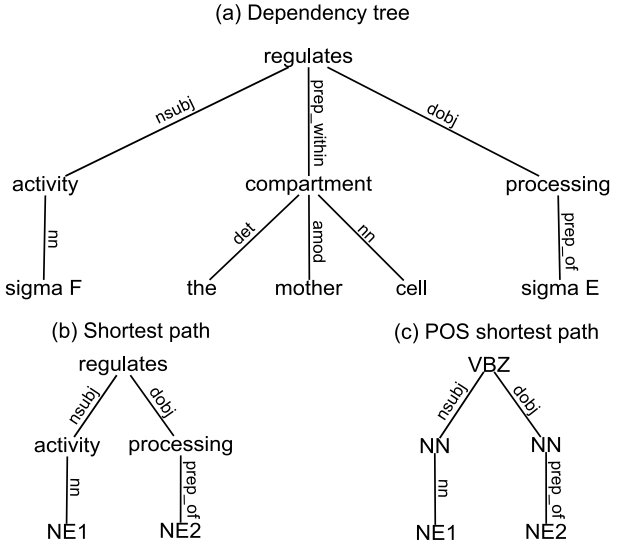

Figure 1: Dependency tree and the shortest paths for Example 2

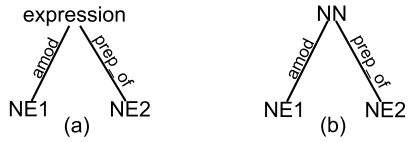

Figure 2: The shortest paths for Example 1

The most relevant part of the dependency tree to analyze the grammatical relation between between two proteins is the subtree corresponding to the shortest path between these proteins. In the remainder, we will restrict ourselves to this subtree but keep refering to it as a (shortest path) dependency tree. Figure 1a depicts the shortest path dependency tree for the interaction between *sigma(H)* and *spo0A* from Example 1.

Finally, the POS (shortest path) dependency tree is obtained by substituting words by their corresponding POS. POS dependency trees corresponding to Examples 1 and 2 are shown in Figure 2b and 1c respectively. Note that a dependency tree contains lexical as well as deep syntactic information, while a POS dependency tree contains shallow and deep syntactic information. To obtain a representation with only deep syntactic information we can take either the dependency tree or the POS dependency tree and simply ignore the labels of the nodes.

In the next subsection we use $t = (N, E, L)$ to denote a (shortest path) dependency tree t consisting of a set of nodes N, a set of edges E, and a function L that maps nodes and edges to their labels. If there is an edge from node m to node n, we denote this edge by $e(m, n)$.

2.3 Building a classifier

SVM is a classification method that views input data as vectors in m-dimensional space and attempts to induce a maximum margin hyperplane between training data points that belong to different classes. The crucial point is that such a hyperplane can always be represented as a linear combination of a subset of training instances (support vectors) [5]. Moreover the selection of the appropriate support vectors can be done by using only the inner product (kernel function) between training examples. Hence, if the inner product can be computed efficiently, SVM can induce a classifier even in a very rich feature space. To deal with various kinds of input data, different strategies to compute inner products (kernels) have been proposed (see e.g. [20]).

In our case, each data point is a dependency tree (or a POS dependency tree) corresponding to a part of the sentence in which the protein pair occurs. Such a tree can be represented as a vector in the m-dimensional space made up by all subtrees in the dataset [4]. More in particular, assuming that all unique subtrees in the dataset are enumerated from 1 to m, the function $h_s(t)$, $s \in [1, m]$, is defined as the number of occurrences of subtree s in tree t. Then each tree t can be represented by a vector $\phi(t) = \{h_1(t), h_2(t), ..., h_m(t)\}$. A kernel function measuring the similarity between trees t_1 and t_2 based on whether they contain the same subtrees, is defined as the inner product

$$K(t_1, t_2) = \langle \phi(t_1), \phi(t_2) \rangle = \sum_s h_s(t_1) \cdot h_s(t_2) \qquad (1)$$

The underlying vector representation is very rich since the number of subtrees of a tree grows exponentially with the tree size, which makes the computation of the inner product intractable. However, the right hand side of (1) can be interpreted as the number of common subtrees of t_1 and t_2. This number can be computed efficiently following a procedure proposed in [4].

Kim et al. [15] follow this procedure in developing an SVM classifier based on a kernel K_{FULL} that is a combination of a kernel K_T comparing dependency trees, with a kernel K_{POS} comparing POS dependency trees. Note that, because of their construction, K_T relies on lexical and deep syntactic information, while K_{POS} is based on shallow and deep syntactic features. We propose a way in which the kernel K_{FULL} can be stripped down to a kernel K_S that uses only deep syntactic features. In Section 5 we compare the performance of K_{FULL} and K_S. For ease of explanation in this section we follow a bottom-up approach by first defining K_S, and then extending it to the full system from [15].

The trees from Figure 1b and Figure 2a have no subtrees in common, unless we ignore the labels of the nodes. In the latter case there is one common subtree, namely the subtree consisting only of the edge *prep_of* and its adjacent nodes. The recursive formula to compute the number of common subtrees between trees t_1 and t_2 in general relies on the notion of "common child pairs" of node n_1 in t_1 and node n_2 in t_2. This is a set of pairs of nodes that have parents n_1 and n_2 respectively, and that are connected to these parents by the same type of edge. When traversing down the trees in search for common subtrees, these are the nodes at which we want to continue our exploration.

Definition 1. Let $t_1 = (N_1, E_1, L_1)$ and $t_2 = (N_2, E_2, L_2)$ be (POS) dependency trees. For $n_1 \in N_1$ and $n_2 \in N_2$, the set of common child pairs, ignoring the labels of the nodes, is defined as $Com(n_1, n_2) = \{(x, y) | (x, y) \in N_1 \times N_2,$

$e(n_1, x) \in E_1,\ e(n_2, y) \in E_2,\ L_1(e(n_1, x)) = L_2(e(n_2, y))\}$.

Note that, since the labels are not taken into account anyway, it does not make a difference whether we use dependency trees or the corresponding POS dependency trees in Definition 1. The same remark holds for Definition 2.

Definition 2. Let $t_1 = (N_1, E_1, L_1)$ and $t_2 = (N_2, E_2, L_2)$ be (POS) dependency trees. For $n_1 \in N_1$ and $n_2 \in N_2$, the number of common subtrees rooted at n_1 and n_2, ignoring the labels of the nodes, is defined as

$$Cm(n_1, n_2) = \begin{cases} 0, & \text{if } Com(n_1, n_2) = \emptyset, \\ \prod_{(x,y) \in Com(n_1,n_2)} & \\ (Cm(x, y) + 2) - 1 & \text{otherwise.} \end{cases}$$

The recursive formula reflects the fact that a new subtree rooted at n_1 and n_2 can be found either by picking 1 of the $Cm(x, y)$ subtrees or by adding the x/y nodes, or just by staying as is (therefore +2). 1 is subtracted from the whole result to exclude one combination with the tree consisting of the n_1/n_2 node only.

Example 3. Let t_1 and t_2 be the dependency trees from Figure 1b and Figure 2a respectively. $Com(n_1, n_2)$ is the empty set for all node pairs with exception of $Com(proc, exp)$ $= \{(NE2, NE2)\}$. Hence

$$Cm(proc, exp) = (Cm(NE2, NE2) + 2) - 1 = 1$$

while $Cm(n_1, n_2) = 0$ for all other node pairs. This means that there is only one common subtree between t_1 and t_2, rooted at the *proc/exp* nodes and ending at NE2. □

Using Definition 2 we are now able to define a kernel K_S that looks only at deep syntactic information. It computes the similarity between trees as the number of grammatical structures that they have in common. Again there is no difference whether we apply Definition 3 to dependency trees or their corresponding POS dependency trees.

Definition 3. The kernel function K_S is defined as

$$K_S(t_1, t_2) = \sum_{n_1 \in N_1, n_2 \in N_2} Cm(n_1, n_2) \tag{2}$$

for (POS) dependency trees t_1 and t_2.

Example 4. Let t_1 and t_2 be the dependency trees from Figure 1b and Figure 2a respectively. Since $|N_1| = 5$ and $|N_2| = 3$, the summation in the right hand side of (2) consists of 15 terms. In Example 3 we already established that all of these terms are 0 with the exception of one term that equals 1. Hence

$$\begin{aligned} K_S = {} & Cm(reg, exp) + Cm(reg, NE1) + Cm(reg, NE2) \\ & + Cm(act, exp) + Cm(act, NE1) + Cm(act, NE2) \\ & + Cm(proc, exp) + Cm(proc, NE1) + Cm(proc, NE2) \\ & + Cm(NE1, exp) + Cm(NE1, NE1) + Cm(NE1, NE2) \\ & + Cm(NE2, exp) + Cm(NE2, NE1) + Cm(NE2, NE2) \\ & = 1 \end{aligned}$$ □

To arrive at kernels that take into account additional lexical and/or shallow syntactic information, we need an extended version of Definition 1 that also looks at the labels of nodes.

Definition 4. Let $t_1 = (N_1, E_1, L_1)$ and $t_2 = (N_2, E_2, L_2)$ be (POS) dependency trees. For $n_1 \in N_1$ and $n_2 \in N_2$, the set of common child pairs, taking into account the labels of the nodes, is defined as $Com^{lab}(n_1, n_2) = \{(x, y) | (x, y) \in Com(n_1, n_2),\ L_1(n_1) = L_2(n_2),\ L_1(x) = L_2(y)\}$.

The superscript "*lab*" refers to the fact that the labels of the nodes are taken into account. The appearance of $Com(n_1, n_2)$ in the definition of $Com^{lab}(n_1, n_2)$ illustrates that the latter builds on the former. Furthermore, it holds that

$$Com^{lab}(n_1, n_2) \subseteq Com(n_1, n_2)$$

indicating that ignoring the labels of the nodes leads to a more general approach (more nodes are explored when traversing down the trees in search for common subtrees).

The number $Cm^{lab}(n_1, n_2)$ of common subtrees rooted at n_1 and n_2, taking into account the labels of the nodes, can now be defined in a recursive manner entirely analogous to Definition 2, however relying on $Com^{lab}(n_1, n_2)$ instead of on $Com(n_1, n_2)$. Since they have different labels at the nodes, the value of $Cm^{lab}(n_1, n_2)$ might be different depending on whether a dependency tree or a POS dependency tree is used. In both cases it holds however that

$$Cm^{lab}(n_1, n_2) \leq Cm(n_1, n_2) \tag{3}$$

Example 5. Let t_1 and t_2 be the dependency trees from Figure 1b and Figure 2a respectively. For all node pairs it holds that $Com^{lab}(n_1, n_2) = \emptyset$ and $Cm^{lab}(n_1, n_2) = 0$. □

Example 6. Let t_1 and t_2 be the POS dependency trees from Figure 1c and Figure 2b respectively. It holds that $Com^{lab}(NN, NN) = \{(NE2, NE2)\}$ and $Cm^{lab}(NN, NN)$ $= 1$, while for all other node pairs $Com^{lab}(n_1, n_2) = \emptyset$ and $Cm^{lab}(n_1, n_2) = 0$. □

The potentially different behaviour of $Cm^{lab}(n_1, n_2)$ on dependency trees and POS dependency trees gives rise to the definitions of the kernel functions K_T and K_{POS} respectively. Both of them still consider the tree structure when computing the similarity between trees, i.e. they both rely on deep syntactic information. In addition, K_T takes the actual words of the sentence into account (lexical information) while K_{POS} considers parts-of-speech (shallow syntactic information).

Definition 5. [4] The kernel function K_T is defined as

$$K_T(t_1, t_2) = \sum_{n_1 \in N_1, n_2 \in N_2} Cm^{lab}(n_1, n_2) \tag{4}$$

for dependency trees t_1 and t_2.

Definition 6. [15] The kernel function K_{POS} is defined as

$$K_{POS}(t_1, t_2) = \sum_{n_1 \in N_1, n_2 \in N_2} Cm^{lab}(n_1, n_2)$$

for POS dependency trees t_1 and t_2.

Finally, Kim et al. [15] combine K_T and K_{POS} into a kernel K_{FULL} that takes into account lexical, shallow and deep syntactic information.

Definition 7. [15] The kernel K_{FULL} is defined as

$$K_{FULL}(t_1, t_2) = K_T(t_1, t_2) + K_{POS}(t_1^{POS}, t_2^{POS}) \quad (5)$$

for dependency trees t_1 and t_2 and their corresponding POS dependency trees t_1^{POS} and t_2^{POS}.

Notice that K_T is a refinement of K_S in the sense that all the information used by K_S is also used in the same way by K_T. As a consequence, K_{FULL} is also a refinement of K_S, enriching the deep syntactic information of K_S by lexical information (through K_T) as well as shallow syntactic information (through K_{POS}).

Example 7. Let t_1 and t_2 be the dependency trees from Figure 1b and Figure 2a respectively, and t_1^{POS} and t_2^{POS} their corresponding POS depencency trees from Figure 1c and Figure 2b respectively. One can verify that

$$K_T(t_1, t_2) = 0$$

and

$$K_{POS}(t_1^{POS}, t_2^{POS}) = 1$$

hence $K_{FULL}(t_1, t_2) = 1$. Notice that although the trees do not show any resemblance on the lexical level, their similarity on the more general syntactic level is picked up by K_{POS}. In Example 4 we found that their syntactic similarity is also already reflected by K_S. □

We will now investigate whether the additional lexical and shallow syntactic features tend to have a complementary or a substitute effect in general.

3. RELATED WORK

In Table 1 we list some of the most important recent approaches to protein interaction extraction and provide their characteristics most relevant to this paper, namely the information that was used to describe the sentence in which the proteins co-occur, the machine learning algorithm employed and the datasets that were used to perform an evaluation.

The approaches listed in Table 2 are closest to our work as they use structured kernels. Structured or convolution kernels were introduced in [13] by Haussler who proposed how to compute a kernel for structured objects by calculating the similarity of their substructures. This work gave rise to many tree kernel methods in the text mining domain. Collins and Duffy [4] developed a tree kernel that counts the number of common subtrees, but used it for parsing, not for interaction extraction. Recently, Kim et al. [15] applied this kernel to a dependency tree and to a modified dependency tree with POS instead of words, and defined a combined kernel as the sum of these two. The resulting system is reported to achieve state of the art performance on the LLL dataset.

Method	Information	Algorithm	Datasets
[3]	lexical	SVM	AIMed
[10]	lexical shallow deep	Hand-built rules	HPRD50
[9]	shallow deep	C4.5 BayesNet	AIMed LLL
[11]	lexical shallow	SVM	AIMed LLL
[14]	lexical deep deep	BayesNet NaiveBayes K-nearest neighbour Ensembles	AIMed
[21]	lexical shallow deep	SVM	AIMed HPRD50 IEPA LLL
[22]	lexical shallow deep	Maximum entropy	IEPA
[23]	lexical shallow deep	SVM	AIMed

Table 1: General approaches for protein interaction extraction

As can be seen from Table 1 and 2 all the approaches mentioned above (except our own previous work [9]) utilize lexical information (words) along with syntactic information. The main research question tackled in this paper is how much performance can still be achieved when using no lexical information whatsoever.

4. DATASETS

In the line of recent work devoted to protein interaction extraction ([17] and [21]) we use the following 5 datasets for our evaluation purposes: AIMed [2], Bioinfer [18], HPRD50 [10], LLL [16] and IEPA [8]. These datasets are built independently for different biological subdomains, which on one hand provides a close to real-world conditions setting for the evaluation, while on the other hand it also causes fundamental differences among the datasets' structure as well as among annotation schemes. For example, AIMed consists of full abstracts, while the other 4 datasets contain only separate sentences; LLL and IEPA consider asymmetric interactions (where proteins are annotated as agents and targets), while the rest consider symmetric ones; in IEPA interacting verbs are annotated separately; in LLL for every sentence a dependency tree and lemmas for the words are provided, etc.

Furthermore, as it was mentioned in [19], the same dataset annotations are interpreted differently by different researchers, which makes the exact comparison of different approaches infeasible, even when evaluated on the same dataset. Pyysalo et al. [17] proposed conversion software that casts all corpora and annotations to a ground XML format. This allows to avoid most of the above mentioned problems at the cost of loosing non-standard (but probably useful) annotation. Nevertheless, they point out that different corpora have dif-

Method	Information	Algorithm	Datasets
[1]	lexical shallow deep	Sparse RLS	AIMed BioInfer HPRD50 IEPA LLL
[15]	lexical shallow deep	SVM	LLL
[19]	lexical deep	SVM	AIMed

Table 2: Convolution kernel-based approaches for protein interaction extraction

Table 3: Corpora statistics

Dataset	# of interaction instances	
	positive	negative
LLL	164	166
HPRD50	163	270
IEPA	411	482
AIMed	1057	4790
BioInfer	1381	8964

ferent complexity which should be taking into account in the evaluation process.

A characteristic of the complexity that significantly influences the performance that can be achieved on a corpus is the ratio of positive versus negative instances. Table 3 shows that the datasets exhibit quite a different distribution of positive and negative instances. Such a big difference in the ratio of positive versus negative examples is caused by different annotation strategies. In the LLL dataset e.g. only proteins that take part in interactions are annotated, while for other datasets all proteins that occur in the text are annotated. Since we consider all possible protein pairs within a sentence, the number of negative instances grows exponentially with the number of annotated proteins that do not occur in interactions, while the number of positive instances remains the same. The F-measure, a standard evaluation metric that most text mining methods use, is very sensitive to the ratio of positives versus negatives; hence it might not be the best choice for an evaluation metric to compare the performance of a protein interaction system on different corpora. To overcome this issue and to provide a unified measure for interaction extraction systems, Airola et al. [1] propose to use the AUC (area under the ROC curve) metric [12] for evaluation. For completeness, we use both metrics in Section 5.

5. RESULTS AND DISCUSSION

5.1 Experimental setup

To conduct our experiments we used the ppi-conversion software [17] to cast the corpora to a common format, the Stanford Parser[4] to construct dependency trees, the LibSVM[5] package to train the classifier and the AUCCalculator [7] to evaluate it.

We performed cross-validation experiments for all 5 datasets, and 5 cross-dataset validations, where 4 datasets were used to train the classifier and the fifth corpus to evaluate it. This kind of cross-dataset experiment is not common practice yet although it is very instructive to do it because it evaluates the generalization power of a classifier. Training corpora exist only for a few subdomains thus it is an important asset for a classifier to achieve good performance on related, but different data.

Saetre [19] and Airola [1] made an important observation concerning a cross-validation setup. As many approaches deal with interaction instances instead of full sentences, it is possible that two virtually identical instances derived from one sentence could fall in different folds. For example, in the sentence "prot1 and prot2 ... were bound to prot3", the pairs prot1-prot3 and prot2-prot3 could be separated to test and train folds although they are virtually the same, thus making the evaluation process incorrect. As Saetre shows in [19] this could lead up to a difference of 17.5% in the F-score. To fix this problem, he proposes to build splits on the sentence or even the abstract level, so that all instances from one abstract fall into one fold. We apply this strategy for all datasets, however in different fashions. The AIMed dataset is the only dataset that contains abstracts, thus we split abstracts in 10 folds. All other datasets contain separate sentences, thus we build splits on the sentence level.

To evaluate the role of syntactic features for interaction extraction, for each dataset setup we perform two experiments, one using the full kernel K_{FULL} from Definition 7 and one using the syntactic kernel K_S defined in Definition 3. As the kernels are applied under the same conditions and use exactly the same input data, this comparison gives us a clear understanding of the syntactic features' importance.

As mentioned above, the LLL dataset contains a dependency tree and lemmas for each sentence. An important distinction between our and Kim's original experiment [15] is that we do not use the syntactic information provided with the LLL dataset. Instead we generate all syntactic information with the Stanford parser. This allows us to unify the experimental setup for all 5 datasets, and hence make the results comparable.

As we have explained in Section 4, we use both the AUC- and the F-score to measure the performance of the methods. As usual, let TP denote the number of true positives, i.e. the number of positive instances that are classfied as such, let FP denote the number of false positives, i.e. the number of negative instances that are incorrectly classified as positive, and analogously, let TN and FN stand for the number of true negatives and false negatives respectively.

[4] `http://nlp.stanford.edu/downloads/lex-parser.shtml`

[5] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

The following metrics can then be defined:

$$\begin{array}{lcl} \text{recall} & = & TP/(TP+FN) \\ \text{precision} & = & TP/(TP+FP) \\ \text{true positive rate} & = & TP/(TP+FN) \\ \text{false positive rate} & = & FP/(FP+TN) \end{array}$$

The F-score is the harmonic mean of recall and precision, defined as

$$F = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$$

The false positive rate together with the true positive rate correspond to a point in ROC space. By varying the tradeoff between these two metrics one obtains a curve in ROC space. The AUC-score is the area under this ROC-curve. It can be interpreted as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

5.2 Result analysis

The evaluation results for all datasets provided in Table 4 show that for the cross-validation experiments the syntactic kernel and the full kernel are roughly comparable. There is no "clear winner" for all cases, not even within the same evaluation metric. This is an interesting observation as both methods use deep syntactic information in the same way, but the full kernel uses additional shallow syntactic and lexical information. While for the larger datasets (AIMed and BioInfer) there does seem to be some advantage in using this extra information, for the smaller datasets the gain is questionable, and the use of additional features is sometimes even harmful.

First of all, on the LLL dataset the F-score of the syntactic kernel is slightly lower than that of the full kernel, but the former outperforms the latter in terms of AUC. This corresponds to the fact that the syntactic kernel based classifier achieves a higher recall at the cost of a slightly lower precision, which fits the idea that a classifier trained only on deep syntactic features is more general, hence allows to retrieve more instances, while at the same time not generating a lot of extra "noise". Next, on the HPRD50 dataset the performances of both classifiers are very similar. Together with the results on the LLL dataset, this is a good example that a system that performs best in terms of the F-score does not necessarily perform best in terms of the AUC-score, and vice versa.

On the IEPA dataset the syntactic kernel performs better for both metrics, indicating that it achieves a higher recall as well as a higher precision, although the difference is again nearly neglectable. Finally, when going to the larger datasets, the full kernel starts to exhibit a better performance than the syntactic kernel. However, omitting lexical information influences the quality of the classifer significantly only on the BioInfer dataset.

In the cross-dataset experiments the classifiers are trained on 4 of the 5 corpora and tested on the remaining corpus. This means that on one hand there is more data available to learn from, but on the other hand the test data is not so closely related anymore to the training data. Furthermore, since in the larger data sets there is a significant imbalance between the number of positive and negative examples, and in the cross-dataset experiments each training corpus contains at least one of the larger datasets, all of the cross-dataset experiments depart from an imbalanced dataset for training purposes. The test dataset however remains the same, i.e. both in experiments (1) and (6) for instance we only look into the balanced LLL-dataset as the test corpus.

Table 4: Performance comparison for syntactic and full kernels

	Dataset	F-measure		AUC	
		K_S	K_{FULL}	K_S	K_{FULL}
(1)	LLL	0.74	0.76	0.81	0.73
(2)	HPRD50	0.57	0.56	0.69	0.72
(3)	IEPA	0.75	0.72	0.81	0.80
(4)	AIMed	0.33	0.38	0.69	0.69
(5)	BioInfer	0.29	0.34	0.69	0.78
(6)	LLL vs all	0.44	0.39	0.68	0.74
(7)	HPRD50 vs all	0.48	0.55	0.72	0.75
(8)	IEPA vs all	0.35	0.29	0.68	0.67
(9)	AIMed vs all	0.33	0.39	0.67	0.71
(10)	BioInfer vs all	0.30	0.31	0.69	0.70

A first interesting observation when moving from the cross-validation to the cross-dataset experiments, is the significant drop in F-score for those cases in which the classifiers used to be trained on a (fairly) balanced corpus, i.e. experiment (6) versus experiment (1), and experiment (7) versus experiment (2). The AUC-score is overall a lot less affected. In the cross-validation experiments we already noticed that the full kernel shows its advantage for larger training corpora. This trend is continued in the cross-dataset experiments where all training corpora are large. Indeed, the AUC-score for the full kernel is either better or only slightly worse than for the syntactic kernel. The differences in performance between the two methods however ranges only between 0.01 and 0.06, which is a fairly small margin considering that the full kernel uses significantly more information.

These results are in line with our previous work [9] that a C4.5 and a BayesNet classifier relying only on deep syntactic information can achieve state of the art performance, as well as with the findings of [21] who compare the impact of lexical features on one hand with the impact of syntactic features (both shallow and deep) on the other hand. In [21] it is reported that dropping the lexical features, which make up 85-90% of the feature vector in their approach, barely impacts the F-score. The results in Table 4 show that additionally dropping shallow syntactic features in our method corresponds at most to a limited damage in terms of the F-score and the AUC-score and in some cases even improves the performance slightly.

6. CONCLUSIONS

State of the art approaches for protein interaction extraction from natural language texts typically use lexical information (words) combined with shallow syntactic information (POS) and/or deep syntactic information (grammatical relations) about the sentences in which protein names co-occur. In this paper we investigated the individual impacts of these differ-

ent kinds of information on the performance of a prototypical SVM with structured kernel based approach to interaction extraction that was recently published in [15]. We performed 5 cross-validation experiments and 5 cross-dataset experiments involving the LLL, HPRD50, IEPA, AIMed and BioInfer datasets, and looked at both the F-measure and the AUC-score.

Our general finding is that most of the performance can be achieved when using only deep syntactic information, i.e. grammatical relations (we referred to this method as "syntactic kernel"). When the training corpus is small, involving more features can even cause a drop in the performance of the system. The advantage of using additional lexical and shallow syntactic information (i.e. the "full kernel" approach) seems to grow with the size of the training corpus. Still, in the cross-dataset experiments that we performed, the differences in performance range only between 0.01 and 0.06, which is a fairly small margin considering that the full kernel uses significantly more information. Blocking out a specific kind of features in a system for testing purposes does not seem to require much effort in general, and it can help to make the method significantly lighter. We would therefore like to encourage future evaluations of protein interaction extraction methods to report on the individual impacts of the different kinds of features.

7. REFERENCES

[1] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing at ACL'08*, p. 1–9, 2008.

[2] R. Bunescu, R. Ge, R. J. Kate, E. M. Marcotte, R. J. Mooney, A. K. Ramani, and Y. W. Wong. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial Intelligence in Medicine*, 33(2):139–155, 2005.

[3] R. C. Bunescu and R. J. Mooney. Subsequence kernels for relation extraction. *Advances in Neural Information Processing Systems*, 18:171–178, 2006.

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

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[6] A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL)*, p. 423–429, 2004.

[7] J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning (ICML)*, p. 233–240, 2006.

[8] J. Ding, D. Berleant, D. Nettleton, and E. S. Wurtele. Mining medline: Abstracts, sentences, or phrases? In *Proceedings of the Pacific Symposium on Biocomputing*, p. 326–337, 2002.

[9] T. Fayruzov, M. De Cock, C. Cornelis, and V. Hoste. Deeper: A full parsing based approach to protein relation extraction. *Lecture Notes in Computer Science*, 4973: 36–47, 2008.

[10] K. Fundel, R. Küffner, and R. Zimmer. Relex - relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2007.

[11] C. Giuliano, A. Lavelli, and L. Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, p. 401–408, 2006.

[12] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

[13] D. Haussler. Convolution kernels on discrete structures, Technical report, University of California at Santa Cruz, 1999.

[14] S. Katrenko and P. Adriaans. Learning relations from biomedical corpora using dependency tree levels. In *Proceedings of the Fifteenth Dutch-Belgian Conference on Machine Learning (Benelearn)*, 2006.

[15] S. Kim, J. Yoon, and J. Yang. Kernel approaches for genic interaction extraction. *Bioinformatics*, 24(1):118–126, 2008.

[16] C. Nedellec. Learning language in logic - genic interaction extraction challenge. In *Proceedings of the ICML-2005 Workshop on Learning Language in Logic (LLL05)*, p. 31–37, 2005.

[17] S. Pyysalo, A. Airola, J. Heimonen, J. Björne, F. Ginter, and T. Salakoski. Comparative analysis of five protein-protein interaction corpora. *BMC Bioinformatics*, 9(Suppl 3):S6, 2008.

[18] S. Pyysalo, F. Ginter, J. Heimonen, J. Björne, J. Boberg, J. Järvinen, and T. Salakoski. BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, 8:50, 2007.

[19] R. Saetre, K. Sagae, and J. Tsujii. Syntactic features for protein-protein interaction extraction. In *Short Paper Proceedings of the Second International Symposium on Languages in Biology and Medicine (LBM)*, 2007.

[20] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. MIT Press, 2001.

[21] S. Van Landeghem, Y. Saeys, B. De Baets, and Y. Van de Peer. Extracting protein-protein interactions from text using rich feature vectors and feature selection. To appear in *Proceedings of Third International Symposium on Semantic Mining in Biomedicine (SMBM)*, 2008.

[22] J. Xiao, J. Su, G. Zhou, and C. Tan. Protein-protein interaction extraction: a supervised learning approach. In *Proceedings of the 1st International Symposium on Semantic Mining in Biomedicine (SMBM)*, p. 51–59, 2005.

[23] A. Yakushiji, Y. Miyao, T. Ohta, Y. Tateisi, and J. Tsujii. Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, p. 284–292, 2006.