journal homepage: www.intl.elsevierhealth.com/journals/ijmi

Protein–protein interaction extraction by leveraging multiple kernels and parsers

Makoto Miwa[a,*], Rune Sætre[a], Yusuke Miyao[a], Jun'ichi Tsujii[a,b,c]

[a] Department of Computer Science, the University of Tokyo, Japan
[b] School of Computer Science, University of Manchester, United Kingdom
[c] National Center for Text Mining, United Kingdom

ARTICLE INFO

Article history:
Received 31 October 2008
Received in revised form
26 February 2009
Accepted 29 April 2009

Keywords:
Protein–protein interaction extraction
Relation extraction
Support vector machine
Machine learning

ABSTRACT

Protein–protein interaction (PPI) extraction is an important and widely researched task in the biomedical natural language processing (BioNLP) field. Kernel-based machine learning methods have been used widely to extract PPI automatically, and several kernels focusing on different parts of sentence structure have been published for the PPI task. In this paper, we propose a method to combine kernels based on several syntactic parsers, in order to retrieve the widest possible range of important information from a given sentence. We evaluate the method using a support vector machine (SVM), and we achieve better results than other state-of-the-art PPI systems on four out of five corpora. Further, we analyze the compatibility of the five corpora from the viewpoint of PPI extraction, and we see that some of them have small incompatibilities, but they can still be combined with a little effort.

1. Introduction

Extraction of biomedical relations from the literature is an important research topic in the field of biomedical natural language processing (BioNLP). With a rapidly growing number of research papers, researchers have difficulty finding the papers that they are looking for. Relationships between entities, mentioned in these papers, can help biomedical researchers find the specific papers they need. Among biomedical relation types, protein–protein interaction (PPI) extraction has been most widely researched, because PPI information is critical in understanding biological processes. Finding PPIs automatically can help in constructing PPI databases, that have so far mostly been manually constructed, like BIND [1].

Sentence-based pair-wise PPI extraction is one of the most common ways of doing PPI. Several corpora, with slightly different aspects, are available for this task [2]. The goal of the task is to find a criteria to judge whether a sentence, which contains a pair of proteins, actually implies interaction between the pair or not. Detection of PPI was initially tackled by using simple methods based on word co-occurrences [3], while more sophisticated NLP techniques have been used later [4]. For example, NLP tools are used to lemmatize surface words and tag them with their part-of-speech (POS) information. Dependency relations in sentences can also be revealed by syntactic parsers [5]. While NLP techniques make this information explicit, other appropriate techniques should be applied to use all the information col-

* *Corresponding author at*: The University of Tokyo, Graduate School of Information Science and Technology, Bunkyo-ku, Tokyo 113-0033, Japan. Tel.: +81 3 5803 1697; fax: +81 3 5802 8872.
E-mail addresses: mmiwa@is.s.u-tokyo.ac.jp (M. Miwa), rune.saetre@is.s.u-tokyo.ac.jp (R. Sætre), yusuke@is.s.u-tokyo.ac.jp (Y. Miyao), tsujii@is.s.u-tokyo.ac.jp (J. Tsujii).

1386-5056/$ – see front matter © 2009 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.ijmedinf.2009.04.010

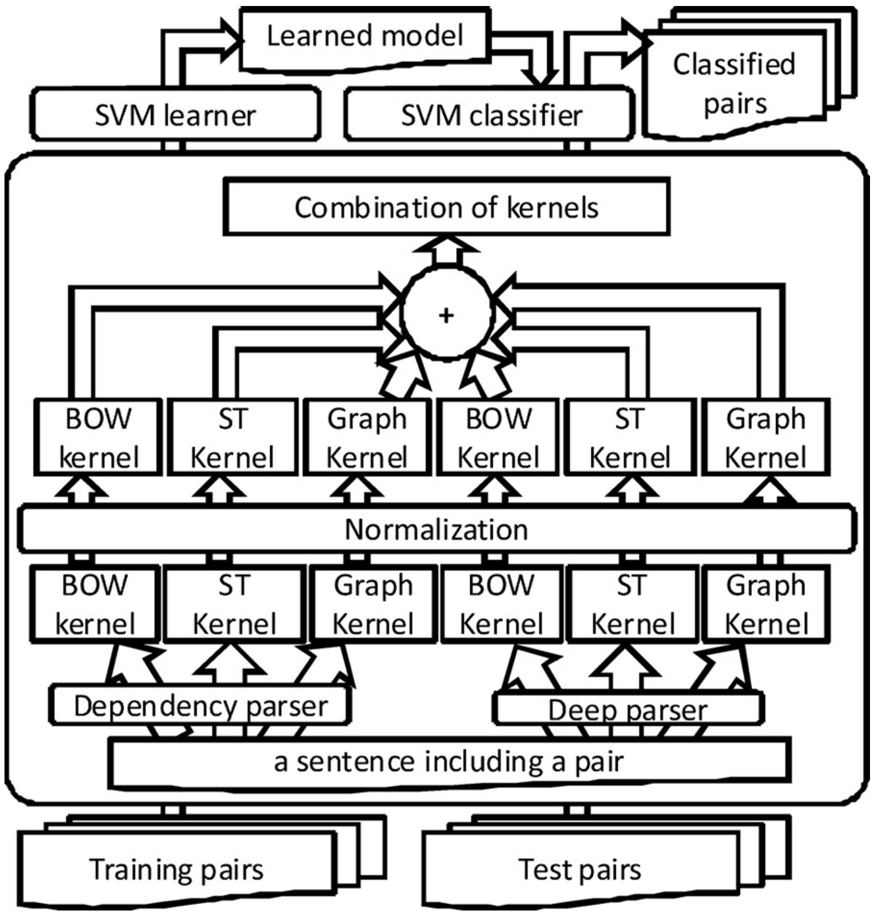

Fig. 1 – Overview of our PPI extraction system (BOW: bag-of-words and SP: shortest path).

lectively when judging whether a sentence is relevant for PPI or not.

For this purpose, several kernels have been proposed. Kernels are constructed to calculate the similarity between their target structures in training (and test) example sentences. Several useful linear classification algorithms, like support vector machines (SVMs) [6], can be easily modified to utilize structured data, and then perform the classification in that higher order dimensional space, by using kernels. Previously published kernels include subsequence kernels [7], tree kernels [8,9], shortest path kernels [10], and graph kernels [11]. Each kernel utilizes a portion of a sentence's structure to calculate a similarity measure, and kernel methods achieve much better results than co-occurrence or rule-based systems [11]. One specific kernel type cannot retrieve all the important information that may be retrieved by other kernels, so multiple kernels are needed.

In this paper, we propose a way of combining kernels, based on several syntactic parsers, for PPI extraction. In order to retrieve the widest possible range of important information from a given sentence, all available kernels must be used. Using SVM, and all the useful information from the combination of kernels, we achieved better results than other state-of-the-art PPI systems on four out of five corpora. We also analyzed the compatibility of the five corpora from the viewpoint of PPI extraction, and found that some are almost compatible, although they cannot be combined without some caution.

2. Materials and methods

In recent years, parsing technology has improved rapidly, and many different types of parsers have been proposed. Some of the parsers are retrained using biomedical corpora, and adapted to biomedical texts using domain adaptation techniques [12].

Kernel methods, applicable to structured data, have also been extensively researched. We have adapted several kernels to each parser's output, and applied them to PPI extraction. The parsers produce different types of structures, and therefore provide different information regarding the target sentences. Each kernel uses different aspects to extract and utilize some portion of the information, either from the output of the parsers or directly from the sentences. We combine all the kernels to draw out as much information as possible from the sentences. To realize our method, the PPI extraction method from [9,12] is extended. We adopt two types of parsers and three types of kernels. Fig. 1 summarizes the way in which the combination of kernels is constructed, and shows an overview of the whole system.

Raf-1$_{p1}$ was activated by **JAK2**$_{p2}$ in the presence of **p21ras**$_{prot}$.

Fig. 2 – A subsentence including an interacting pair (p1–p2) (AIMed, PMID 8876196, sentence 4, pair 5).

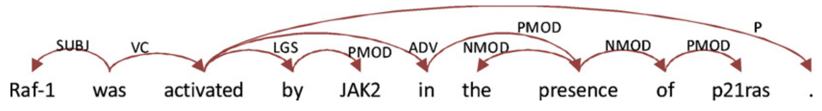

Fig. 3 – Output produced by the dependency parser LRDEP (in CoNLL-X dependency tree format).

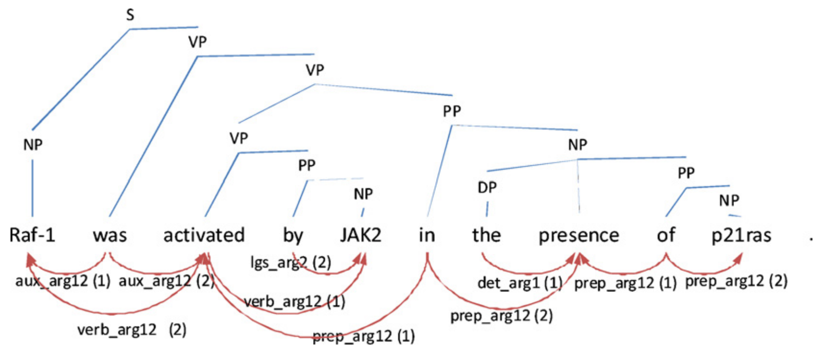

Fig. 4 – Example output produced by the deep parsers Enju. The upper relations represent the syntactic tree structure, and the lower relations represent the predicate argument structures (PASs).

2.1. *Corpora*

We used five corpora[1]: AIMed, BioInfer, HPRD50, IEPA, and LLL [11,2] for our evaluation. The annotation policies for each corpus are slightly different (as summarized in [2]). AIMed and BioInfer are the two largest corpora for sentence-based pair-wise PPI, and the three other corpora are relatively small.

Entities, and their relations, in every corpus are annotated according to the annotation policies of that corpus. Fig. 2 shows an example phrase, including an actual interacting pair (p1 and p2), from AIMed. In this phrase, the pair of “Raf-1 and JAK2” is annotated as an interacting pair, and the other pairs (like “Raf-1 and p21ras”) are not annotated. We treated the interacting pair as a positive example, and the other pairs as negative examples of PPI.

2.2. *Syntactic parsers*

There are many types of parsers, and they each output different syntactic structures. These structures reveal different types of useful information that is latent in the sentence. We focus on two types of parsers, as described below.

2.2.1. *Dependency parser*

The task of a dependency parser is to take a sentence as a sequence of words, and to construct a dependency tree with dependency links between all the words. Fig. 3 shows a parse tree produced by a dependency parser.

2.2.2. *Deep parser*

A deep parser also takes an input sentence as a sequence of words, like a dependency parser, but it constructs a more detailed graph structure that represents theory-specific syntactic/semantic relations among the words. A predicate argument structure (PAS) is often used to represent the *semantic* structure. Deep parsers also treats deeper relations that may include reentrant structures. Fig. 4 shows a parse tree/graph produced by a deep parser. The upper relations represent the syntactic structure of the sentence, and the lower relations represent the PAS. We note that the PAS related to the verb *activated* (in passive voice) is normalized to the same form as the PAS of the active voice (*a* ctivate). E.g. in both cases *arg1* of the verb is JAK2, and *arg2* is Raf-1.

2.3. *Kernels*

Syntactic parsers produce parse trees or graphs that are undoubtfully useful, but how to extract the useful information from these structures is an open problem. Several kernels have been proposed to extract useful information from such structures. Single words, directly from the sentences, are also useful features, so several kernels are also used to treat the combination of words. We describe the kernels we use below. In all the kernels, for generalization purposes, the protein names in a sentence are converted to ENTITY1, ENTITY2, or PROT, according to which pair is currently being processed.

2.3.1. *Bag-of-words (BOW) kernel*

A bag-of-words kernel calculates the similarity between two unordered sets of words, given as feature vectors. As input, three different types of word-sets are used. The different-sets contain the lemma forms of the words “before”, “inside of”, and “after” the pair [9] and the frequency for these words. The

[1] http://mars.cs.utu.fi/PPICorpora/GraphKernel.html.

before	–
middle	activate:1, be:1, by:1
after	PROT:1, in:1, of:1, presence:1, the:1

Fig. 5 – BOW features.

lemmas in the sets are limited to the top 1000 most frequent lemmas. Fig. 5 shows BOW features of the sentence in Fig. 2. A polynomial kernel is applied to each feature vector, and the outputs are summed up into a single output from the BOW kernel.

2.3.2. *Subset tree (ST) kernel*

A subset tree kernel [8] calculates the similarity between two input trees by counting the number of common subtrees. Subset tree kernels are applied to the shortest path between a pair (extracted from the parse tree). The shortest path is calculated by Dijkstra's algorithm [13], and includes reverse relations, to preserve the direction of the parse tree relations. As explained in Section 2.2.2, the deep parser assigns the same normalized PAS to *active* and *p* assive voice verbs. To make efficient use of the path structure, the shortest path for the deep parser is converted into a tree form with no reverse relations (towards the root of this new tree). The predicate type information (in PASs, from the deep parser), which was unused in previous works [9,14], is used to represent the dependency types. An example of shortest path features can be found in Fig. 6.

2.3.3. *Graph kernel*

A graph kernel [15,11] calculates the similarity between two input graphs by comparing the relations between common vertices (nodes). Our graph kernel is called an all-dependency-paths graph kernel. The weights of the relations are calculated using all walks (possible paths) between each pair of vertices.

The graph consists of two directed subgraphs, a parse graph, and a graph representing the linear order of words. As the vertices in the first subgraph, the dependency, the lemma form, and the POS information from the parser's output are used. The dependencies and lemmas that are in the shortest path are distinguished from the ones not in the path by adding "in the shortest path" (IP). A vertex in the second subgraph is labeled with the lemma, the POS, and the place information. The place is separated into three by the target pair like for BOW features (before, middle, and after).

Fig. 7 is an example of the subgraphs made from the parse tree in Fig. 3. Our graph kernel representation is different from the graph kernel in [15,11]. First, each word in the shortest path has two labels, and the relations in the shortest path are not replaced, but duplicated in the first subgraph. Second, the shortest path is calculated by using the constituents in the PAS structure. The words in the constituents in the shortest path are distinguishably marked as being "in the shortest path" (IP). Finally, the POS information for protein names are not attached.

For the calculation, two types of matrices are used: a label matrix **L**, and an edge matrix **A**. The label matrix is a (sparse) $N \times L$matrix, where N is the number of vertices (nodes), and L is the number of labels. It represents the correspondence between labels and vertices. $\mathbf{L}_{ij}$ is 1 if the ith vertex corresponds to the jth label, and 0 if otherwise. The edge matrix is a (sparse) $N \times N$ matrix, and represents the relation between pairs of vertices. $\mathbf{A}_{ij}$ is a weight w_{ij} if the ith vertex is connected to the jth vertex, and 0 if otherwise. The weight is a predefined constant (described in the caption of Fig. 7). Using the Neumann Series, a graph matrix **G** is calculated as:

$$\mathbf{G} = \mathbf{L}^{\mathrm{T}} \sum_{n=1}^{\infty} \mathbf{A}^n \mathbf{L} = \mathbf{L}^{\mathrm{T}}((\mathbf{I} - \mathbf{A})^{-1} - \mathbf{I})\mathbf{L}. \quad (1)$$

This matrix sums up the weights of all the walks between any pair of vertices so, as a result, each entry represents the strength of the relation between a pair of vertices. Using two input graph matrices **G** and **G**′, the graph kernel $k(\mathbf{G}, \mathbf{G}')$ is defined as:

$$k(\mathbf{G}, \mathbf{G}') = \sum_{i=1}^{L} \sum_{j=1}^{L} \mathbf{G}_{ij} \mathbf{G}'_{ij}. \quad (2)$$

This kernel sums up the products of the common relations' weights.

For fast calculation and performance, the graph kernels of our two subgraphs are calculated separately, and the normalized outputs of the subgraph kernels are summed up. In the evaluation, we calculated the matrices beforehand, and used a sparse feature vector representation of the calculated elements in the matrices.

2.4. *Combination of kernels*

Different parsers treat different layers of syntax/relations. The dependency parsers ignore some deep information, and conversely, the deep parsers do not output certain shallow relations. Every kernel has different aspects, with different advantages and disadvantages. The BOW kernels can easily combine the words, but they ignore the internal word order and inter-word relations. The subset tree kernels can calculate the similarity of two shortest paths, but they ignore the words, the paths outside of the shortest path, and cycles in the parsed graphs. The graph kernels can treat the parser's output and word features at the same time. However, they cannot treat them properly without tuning the kernel parame-

(*DEPENDENCY* (rSBJ (ENTITY1 be))(VC (be activate))(LGS (activate by))(PMOD (by ENTITY2)))

(*DEEP* (activate (verb_arg12:arg1 ENTITY2)(verb_arg12:arg2 ENTITY1)))

Fig. 6 – Shortest path features.

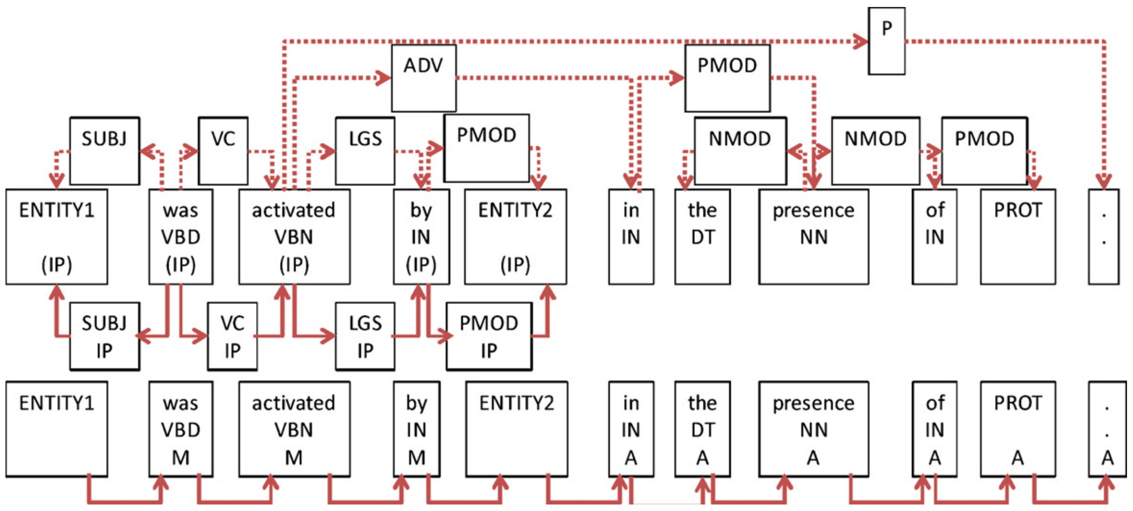

Fig. 7 – Graph kernel with two directed subgraph features. The upper subgraph is made from a parse graph. Relations with status (IP: in the shortest path) and the lemmas with POS (and status) are used as labels of vertices in the subgraph. It should be noticed that this structure is different from [11] because each word in the shortest path has two labels (a label with IP and the other label without IP), and because the relations in the shortest path are not replaced, but duplicated, in the first graph. The lower subgraph represents the linear order of words. Lemma forms with POS and place information (B: before, M: middle, and A: after) are used as labels of vertices in this subgraph. Weights are assigned to the edges in the subgraphs as in [11]: 0.9 for edges in the shortest paths and in the second graph (represented with full lines), and 0.3 for other edges (represented with dotted lines).

Table 1 – Effect of different preprocessing and evaluation methods on the AIMed corpus (PREP: preprocessing and SPLIT: splitting). The number of positive (POS) and negative (NEG) examples, precision (P), recall (R), F-score (F), and AUC are shown. We used two differently preprocessed versions of the AIMed corpus, and evaluated with 10-fold CV using two different splitting methods using all the available kernels and parsers.

PREP	SPLIT	POS	NEG	P	R	F	σ_F	AUC	σ_{AUC}
[11]	[11]	1000	4834	55.0	68.8	60.8	6.6	0.868	0.033
[11]	[18]	1000	4834	56.8	67.8	61.6	3.4	0.876	0.025
[18]	[11]	1005	4643	56.1	71.3	62.3	5.3	0.871	0.034
[18]	[18]	1005	4643	58.9	68.8	63.2	5.2	0.876	0.029

ters. They may also miss some distant words, and similarities of paths among more than three elements.

For these reasons, different kernels calculate different aspects of the similarity between two sentences. Combining all the similarity scores can reduce the danger of missing some important features, and it can produce a new useful similarity measure. To realize this combination of the different types of kernels, based on different parse structures, we sum up the normalized output of all the kernels k_{ij} like this:

$$k(X, X')= \sum_{i \in \{BOW,ST,Graph\}} \sum_{j \in \{Dependency,Deep\}} \frac{k_{ij}(X, X')}{\sqrt{k_{ij}(X, X)}\sqrt{k_{ij}(X', X')}}, \quad (3)$$

where i represents the type of kernel, and j is related to different parsers. This is a very simple combination, but the resulting kernel function contains all of the kernels' information.

2.5. *Experimental settings*

Our system is based on the AKANE++ PPI system[2][9]. We utilized LRDEP alpha2[3][5] as the dependency parser, and the Enju parser 2.3[4][14] as the deep parser. Both parsers were retrained using the GENIA Treebank corpus[5][16]. As for the LRDEP, we used the GENIA Tagger[6] for the lemma information. We used SVM for the machine learning evaluation. The performance was measured in an abstract-wise 10-fold cross-validation (CV), and using the one-answer-per-occurrence criterion, which have been commonly used for the evaluation of other PPI extraction methods before [17]. We controlled the position of the separating hyperplane of the SVM by varying

[2] http://www-tsujii.is.s.u-tokyo.ac.jp/~satre/akane/.
[3] http://www.cs.cmu.edu/~sagae/parser/.
[4] http://www-tsujii.is.s.u-tokyo.ac.jp/enju/.
[5] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/.
[6] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/tagger/.

the threshold, and we calculated the average. Other parameters were fixed, and the regularization parameter C was set to 1.

We used AUC (area under the ROC [receiver operating characteristics] curve) and F-scores to evaluate our system. The ROC curve is a plot of the true positive rate (TPR) vs the false positive rate (FPR) for different thresholds. F-Score is the harmonic mean of precision and recall. As the threshold was varied, we used the point with the highest averaged F-score as the final F-score. The averages of precision, recall, and F-score were calculated independently. This is a good estimation of the F-score obtained with 10-fold CV on other training data [18]. Because of these two different points of view, the best result in AUC differs from the best result in the F-score. Which result is actually better depends on the given task; we have thus reported both results. Additionally, we also calculated the standard deviations for the F-score and AUC [11], for reliable comparison with other systems.

For the 10-fold CV, we split the corpora in the same way as recommended in [11].

3. Results

3.1. The effect of preprocessing and different evaluation methods on the AIMed corpus

In Table 1, we evaluate the AIMed corpus using different types of preprocessing and 10-fold splitting. The preprocessing includes tag fixes and sentence splitting. The resulting F-scores differ by 1–2%, because the preprocessing causes different numbers of sentence-based protein pairs. For the 10-fold splitting, we used abstract-wise 10-fold CV using two splitting methods. The results show that the choice of evaluation method affects the results, especially the AUC value is affected. These differences are statistically ignorable, but they are still causing some noise in the comparison, so we need to decide on just one method. We use the corpus and splitting recommended in [11] through the rest of this evaluation.

3.2. Accuracy improvements by combinations

In Table 2, we see that the performance improves with addition of more kernels except for the BOW kernel. This shows that both the combination of kernels, and the combination of parsers, are effective for PPI extraction. The classifier utilizing all kernels and parsers was statistically one of the best classifiers, and the classifier utilizing multiple parsers and kernels performed much better than the classifier using only one kernel and one parser. This result is different from [18], because slightly different parsers, kernels, preprocessing and evaluation methods were used.

Table 2 – F-Score (F) and AUC on the AIMed corpus (L: LRDEP, E: Enju, C: co-occurrence (baseline), B: BOW, T: subset tree, G: graph, +: summation of kernels).

	L		E		L + E	
	F	AUC	F	AUC	F	AUC
C	30.2	-	-	-	-	-
B	52.7	0.822	52.5	0.819	52.8	0.821
T	55.1	0.799	54.5	0.794	58.2	0.825
G	59.1	0.854	59.3	0.853	59.5	0.859
T + B	58.5	0.849	58.3	0.853	60.5	0.859
G + B	57.0	0.847	57.1	0.850	57.8	0.852
T + G	62.0	0.873	61.0	0.870	61.9	0.876
T + G + B	59.9	0.863	60.9	0.865	60.8	0.868

The results from comparison with other related PPI methods on the AIMed corpus is summarized in Table 3.

The results cannot be compared directly, because of differences in data preprocessing, different number of target protein pairs, and different evaluation methods, but we compare our method with other methods based on the evaluation proposed in these other PPI papers. We use F-score for all the comparisons, except for the comparison with [11], which uses AUC for the first time in PPI extraction.

Our method outperformed all the PPI extraction methods evaluated with the abstract-wise 10-fold CV. The results in [9,12] were obtained evaluating self-interacting pairs [18]. They used the AKANE++ PPI system, and the results are expected to be similar to the result using T + B and L + E in Table 2. Our method is different from [11] in that they performed the leave-one-document-out CV on the training data to tune the parameters of the classifier. Our method performed 5.2% better than theirs using F-score and 0.028 better in AUC. Ref. [17] used a different evaluation method, but as reported in [11], the result was an F-score of 52.4%. Refs. [7,21,22] also used differ-

Table 3 – Comparison with previous results of the PPI extraction methods with the abstract-wise 10-fold CV on the AIMed corpus.

	POS	NEG	P	R	F	σ_F	AUC	σ_AUC
Ours	1000	4834	58.7	66.1	61.9	7.4	0.876	0.028
[11]	1000	4834	52.9	61.8	56.4	5.0	0.848	0.023
[19]	1107	4524	54.2	42.6	47.7			
[20]			33.7	33.1	33.4			
[12]	1059	4589	54.9	65.5	59.5			
[9]	1068	4573	64.3	44.1	52.0			
[21]			59.6	60.7	60.0			
[17]			60.9	57.2	59.0			
[7]			65.0	46.4	54.2			
[22]			45.0	68.4	53.4			

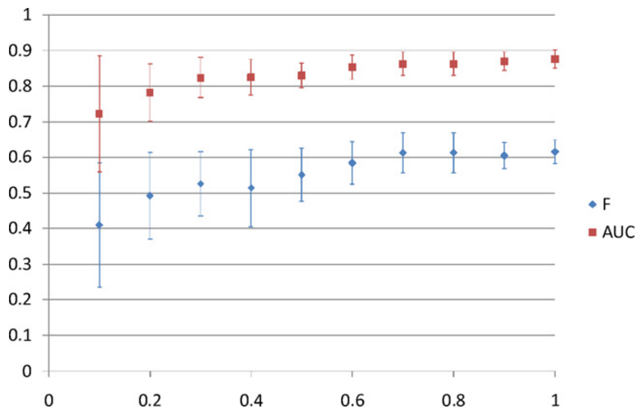

Fig. 8 – Learning curve on AIMed.

ent evaluation methods. Their systems are expected to give a lower performance [9,11] with our evaluation method.

Fig. 8 shows the learning curve on increasing portions of the AIMed corpus. Both *F*-score and AUC are continuously increasing with larger number of labeled data, so even the whole AIMed corpus is not enough to learn the best possible PPI, and the performance can be unstable. To show the effectiveness of our method, we also evaluated our method on other corpora.

Fig. 4 shows the performance on the five corpora described in [2], while using all available parsers and kernels. The results are better than the results in Ref. [11], except for the IEPA corpus. HPRD50 and LLL are smaller than AIMed, and the variance in *F*-score and AUC are larger than for the AIMed corpus. Interestingly, the variance of IEPA is smaller than that of BioInfer, although IEPA is smaller than BioInfer in AUC. *F*-Score can be unstable even while using the same threshold (because of the distribution of positive and negative examples in each fold), and the results in AUC show that learning IEPA was more stable than learning BioInfer.

3.3. *Cross-corpora evaluation*

The learning curve of AIMed is still increasing when all the training data has been used, so we need more training examples. To examine the compatibility of the five corpora from the viewpoint of PPI relation extraction, we trained our classifier on each corpus using all parsers and kernels, and we classified the other corpora using the learned model. The *F*-scores from the optimal thresholds are shown in Table, and the corresponding AUC values are shown in Table.

The classifiers trained with BioInfer and IEPA performed well. This is surprising because IEPA is much smaller than AIMed and BioInfer, and usually learning with larger corpora produces better performance. The results on the LLL corpus using classifiers trained on IEPA are better than the 10-fold CV result using LLL corpus itself for training. This is because IEPA is much larger than LLL but also because IEPA and LLL can be said to be similar in the viewpoint of PPI. The results using other corpora are not better than the results from internal 10-fold CV in the same corpora, except for the case with the LLL corpus. This is not surprising, because the annotation policies are different, and the classifiers cannot predict these differences. The model based on an original corpus performs better than the models based on other corpora in other cases, but the results are maximum 12.5% better *F*-score than for the best performing model based on other corpora. This shows that a corpus and the best neighboring corpus are almost compatible, and that they can be merged with small efforts.

Table 4 – Results of our PPI extraction method, evaluated on five corpora, while using all parsers and kernels.

	POS	NEG	P	R	F	σ_F	AUC	σ_AUC
AIMed	1000	4834	55.0	68.8	60.8	6.6	0.868	0.033
BioInfer	2534	7119	65.7	71.1	68.1	3.2	0.859	0.044
HPRD50	163	270	68.5	76.1	70.9	10.3	0.822	0.063
IEPA	335	482	67.5	78.6	71.7	7.8	0.844	0.042
LLL	164	166	77.6	86.0	80.1	14.1	0.863	0.108

Table 5 – *F*-Score for the cross-corpora evaluation using all parsers and kernels. Rows correspond to the training corpora, and columns the test corpora. For the comparison, we show the 10-fold CV results from Table 4 in parentheses.

	AIMed	BioInfer	HPRD50	IEPA	LLL
AIMed	(60.8)	53.1	**68.3**	68.1	73.5
BioInfer	**49.6**	(68.1)	**68.3**	**71.4**	76.9
HPRD50	43.9	48.6	(70.9)	67.8	72.2
IEPA	40.4	**55.8**	66.5	(71.7)	**83.2**
LLL	38.6	48.9	64	65.6	(80.1)

Highest values in each column are shown in bold face.

Table 6 – AUC for the cross-corpora evaluation using all parsers and kernels. Rows correspond to the training corpora, and columns the test corpora. For the comparison, we show the 10-fold CV results from Table 4 in parentheses.

	AIMed	BioInfer	HPRD50	IEPA	LLL
AIMed	(0.868)	0.748	**0.815**	0.774	0.773
BioInfer	**0.802**	(0.859)	0.811	**0.818**	0.835
HPRD50	0.755	0.685	(0.822)	0.780	0.754
IEPA	0.727	**0.756**	0.759	(0.844)	**0.892**
LLL	0.673	0.693	0.739	0.762	(0.863)

Highest values in each column are shown in bold face.

4. Conclusions

In this paper, we proposed an approach using a combination of kernels for PPI extraction. This combination can extract and combine several different layers of information from a sentence and its syntactic structures by using several parsers. Each kernel extracts some aspects of the information from the sentence while losing other aspects of the available information. The combination of multiple kernels can gather up more information and cover some of losses. To show the usefulness of combining multiple parsers and kernels for PPI extraction, we evaluated our method using five different corpora. We achieved better results than other state-of-the-art PPI systems on four out of the five corpora. Our system is available

Summary points

"What was already known on the topic of this study"

- A single kernel with a single parser is useful for PPI extraction;
- PPI corpora are made for different purposes.

"What has this study added to our knowledge"

- Multiple kernels with multiple parsers perform better than each kernels alone for PPI extraction;
- Different PPI corpora can be combined with a small effort to improve PPI extraction performance.

upon request. As a cross-corpora evaluation, we tried to predict the interactions in a corpus by using models constructed from another corpus. The results show that the five corpora have some compatibilities, and although they cannot be combined directly, we think it is possible to merge them with small efforts.

We summed up the values provided by different kernels using a simple formula (as shown in Section 2.4), but we still need to study the effect of different weighting coefficients for each kernel to achieve further improvements.

The learning curve in Fig. 8 showed that one corpus is not large enough to saturate our learning system, even when using AIMed, which is one of the largest available corpora for sentence-based pair-wise PPI. For further improvements, we need to find a way to make use of multiple corpora (and other biomedical resources) simultaneously, in order to exploit the full potential of our method.

Acknowledgments

This work was partially supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan) and Genome Network Project (MEXT, Japan).

REFERENCES

[1] S. Mathivanan, B. Periaswamy, T. Gandhi, K. Kandasamy, S. Suresh, R. Mohmood, Y. Ramachandra, A. Pandey, An evaluation of human protein–protein interaction data in the public domain, BMC Bioinformatics 7 (Suppl. 5) (2006) 19.
[2] S. Pyysalo, A. Airola, J. Heimonen, J. Björne, F. Ginter, T. Salakoski, Comparative analysis of five protein–protein interaction corpora. BMC Bioinformatics 9 (Suppl. 3) (2008), p. S6.
[3] C. Blaschke, M.A. Andrade, C. Ouzounis, A. Valencia, Automatic extraction of biological information from scientific text: protein–protein interactions, in: Proceedings of the AAAI Conference on Intelligent Systems in Molecular Biology, 1999, pp. 60–67.
[4] R.C. Bunescu, R. Ge, R.J. Kate, E.M. Marcotte, R.J. Mooney, A.K. Ramani, Y.W. Wong, Comparative experiments on learning information extractors for proteins and their interactions, Artificial Intelligence in Medicine 33 (2) (2005) 139–155.
[5] K. Sagae, J. Tsujii, Dependency parsing and domain adaptation with LR models and parser ensembles, in: Proceedings of the EMNLP-CoNLL 2007, 2007.
[6] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, Cambridge, U.K., 2000.
[7] R.C. Bunescu, R.J. Mooney, Subsequence kernels for relation extraction, in: NIPS 2005, 2005.
[8] A. Moschitti, Making tree kernels practical for natural language processing, in: Proceedings of the EACL 2006, 2006.
[9] R. Sætre, K. Sagae, J. Tsujii, Syntactic features for protein–protein interaction extraction, in: LBM 2007 short papers, 2007.
[10] R.C. Bunescu, R.J. Mooney, A shortest path dependency kernel for relation extraction, in: HLT'05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Morristown, NJ, USA, 2005, pp. 724–731.
[11] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, T. Salakoski. A graph kernel for protein–protein interaction extraction, in: Proceedings of the BioNLP 2008 Workshop, 2008.
[12] Y. Miyao, R. Sætre, K. Sagae, T. Matsuzaki, J. Tsujii, Task-oriented evaluation of syntactic parsers and their representations, in: Proceedings of the 45th Meeting of the Association for Computational Linguistics (ACL'08:HLT), 2008.
[13] E.W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271, http://jmvidal.cse.sc.edu/library/dijkstra59a.pdf.
[14] Y. Miyao, J. Tsujii, Feature forest models for probabilistic HPSG parsing, Computational Linguistics 34 (1) (2008) 35–80.
[15] T. Gärtner, P.A. Flach, S. Wrobel, On graph kernels: hardness results and efficient alternatives, in: Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop, 2003.
[16] J.-D. Kim, T. Ohta, Y. Tateisi, J. Tsujii, GENIA corpus—a semantically annotated corpus for bio-textmining, Bioinformatics 19 (2003) i180–i182.
[17] C. Giuliano, A. Lavelli, L. Romano, Exploiting shallow linguistic information for relation extraction from biomedical literature, in: Proceedings of the EACL 2006, 2006.
[18] M. Miwa, R. Sætre, Y. Miyao, T. Ohta, J. Tsujii, 2008. Combining multiple layers of syntactic information for protein–protein interaction extraction. In: Salakoski, T., Rebholz-Schuhmann, D., Pyysalo, S., (Eds.). Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM 2008), Turku Centre for Computer Science (TUCS), Turku, Finland, pp. 101–108.
[19] T. Mitsumori, M. Murata, Y. Fukuda, K. Doi, H. Doi, Extracting protein–protein interaction information from biomedical text with SVM, IEICE—Transactions on Information and Systems E89-D (8) (2006) 2464–2466.
[20] A. Yakushiji, Y. Miyao, Y. Tateisi, J. Tsujii, Biomedical information extraction with predicate-argument structure patterns, in: Proceedings of the First International Symposium on Semantic Mining in Biomedicine, 2005.
[21] G. Erkan, A. Ozgur, D.R. Radev, Semi-supervised classification for extracting protein interaction sentences using dependency parsing, in: EMNLP 2007, 2007.
[22] S. Katrenko, P. Adriaans, Learning relations from biomedical corpora using dependency trees, in: Proceedings of the KDECB, 2006, pp. 61–80.