Submitted for publication

Learning Information Extractors for Proteins and their Interactions

Razvan Bunescu RAZVAN@CS.UTEXAS.EDU
Ruifang Ge GRF@CS.UTEXAS.EDU
Rohit J. Kate RJKATE@CS.UTEXAS.EDU
Raymond J. Mooney MOONEY@CS.UTEXAS.EDU
Yuk Wah Wong YWWONG@CS.UTEXAS.EDU
Department of Computer Sciences, University of Texas, Austin, TX 78712, USA

Edward M. Marcotte MARCOTTE@ICMB.UTEXAS.EDU
Arun Ramani ARUN@ICMB.UTEXAS.EDU
Institute for Cellular and Molecular Biology and Center for Computational Biology and Bioinformatics, University of Texas, Austin, TX 78712, USA

Abstract

We present results from a variety of learned information extraction systems for identifying human protein names in Medline abstracts and subsequently extracting interactions between the proteins. We demonstrate that machine learning approaches using support vector machines and hidden Markov models are able to identify human proteins with higher accuracy than several previous approaches. We also demonstrate that various rule induction methods are able to identify protein interactions more accurately than manually-developed rules.

1. Introduction

An incredible wealth of biological information is stored in published articles in scientific journals. Summaries of more than 11 million such articles are available in the Medline database. However, retrieving and processing this information is very difficult due to the lack of formal structure in the natural-language narrative in these documents. Automatically extracting information from biomedical text holds the promise of easily consolidating large amounts of biological knowledge in computer-accessible form. A number of recent projects have focused on the manual development of information extraction (IE) systems for extracting information from biomedical literature (Fukuda et al., 1998; Blaschke & Valencia, 2001). Unfortunately, manual engineering of IE systems for particular applications is a tedious and time-consuming process. Consequently, significant recent research in information extraction has focused on using machine learning techniques to help automate the development of IE systems (Cardie, 1997). Recently, several machine learning methods have been used to develop Medline IE systems (Tanabe & Wilbur, 2002; Raychaudhuri et al., 2002).

We are exploring the use of a variety of machine learning methods to automatically develop IE systems for extracting information on gene/protein name and interactions from Medline abstracts. Approximately 40,000 human genes are known from the sequences of the human genome (Venter et al., 2001), yet fewer than 5,000 are well characterized and likely to be described in the literature. Unlike other organisms, such as yeast or *Escherichia coli*, human gene names have no standardized naming convention, and thus represent a more difficult extraction problem. In this paper, we present cross-validated results on identifying human proteins and their interactions by training and testing on a set of approximately 1,000 manually-annotated Medline abstracts that discuss human proteins. Previous projects on extraction from Medline typically present results for a single method with limited or no comparison to other methods. By contrast, we present uniform results of a wide variety of methods on a single, reasonably large, human-annotated corpus, thereby giving a broader picture of the relative strengths of different approaches.

2. Biomedical Corpora

In order to generate a corpus for testing the extraction of protein names and interactions, we manually tagged approximately 1,000 abstracts from among the 11 million abstracts available in Medline. 750 abstracts containing the word "human" are used for testing protein name extraction. Of these, 61.3% discussed proteins, with a total of 5,206 protein references. 200 abstracts previously known to contain protein interactions were obtained from the Database of Interacting Proteins (`http://dip.doe-mbi.ucla.edu/`) and tagged for 1,101 interactions and 4,141 protein names. As negative examples of interactions were rare in these abstracts, an extra set of 30 abstracts were collected by scanning approximately 5,000 abstracts for sentences that mention at least two proteins that do not interact. The resulting 230 abstracts are used to test interaction extraction.

3. Protein Name Identification

In this section we explore the problem of recognizing references to human proteins using the tagged data described in the previous section.

3.1. IE Methods

3.1.1. Dictionary-based Extraction

The success of a protein tagger depends on how well it captures the regularities of protein naming as well as name variations. In the dictionary-based approach, we started with an extensive set of protein names extracted from two fairly comprehensive sources: (1) the file `human.seq`, downloaded from the Human Proteome Initiative (HPI) of Expasy;[1] and (2) the file `feb2002-tables.tar.gz`, downloaded from the Gene Ontology Database.[2]

Altogether, these dictionaries contain 42,172 protein names (synonyms included). This collection of protein names, henceforth referred to as the original dictionary, was further extended by isolating and replacing numbers with $\langle n \rangle$, Roman letters with $\langle r \rangle$ and Greek letters with $\langle g \rangle$. We tag a textual n-gram as a protein name only if it is an instance of one of these generalizations. The aim was to extend the coverage of the original set, while at the same time trying to minimize any decrease in accuracy. Table 1 shows some examples of name generalizations.

We used this generalized dictionary-based tagger for supplying a pre-tagged input to some of the learning methods that will be discussed in the following sections.

3.1.2. Rapier and BWI

Rapier (Califf & Mooney, 1999) is a rule learning algorithm that acquires unbounded patterns for extracting information from text. Each extraction rule consists of a pre-filler pattern that matches text immediately preceding a filler, a filler pattern that matches the extracted substring, and a post-filler pattern that matches the text immediately following the filler.

Table 1. Dictionary generalizations.

Protein Names	Generalized Names
NF-IL6-beta	NF IL $\langle n \rangle$ $\langle g \rangle$
NF-kappa B	NF $\langle g \rangle$ $\langle r \rangle$
TR2	TR $\langle n \rangle$

To help Rapier capture generalities that are not evident from the words alone, we supplied part-of-speech (POS) tags to every word in the text. POS tags are potentially useful because certain types of words (e.g. cardinal numbers and proper nouns) are likely candidates of being parts of a protein name. In another experiment, we replaced the POS tags with the output of the dictionary-based tagger in order to incorporate domain knowledge into the learning algorithm.

We also performed similar experiments with Boosted Wrapper Induction (BWI) (Freitag & Kushmerick, 2000), a method that repeatedly learns extraction rules composed of simple contextual patterns called *wrappers*.

3.1.3. Hidden Markov Models

Hidden Markov models (HMMs) are learning with stochastic finite state automata. They have proved to be highly effective in a number of information extraction tasks (Bikel et al., 1999; Ray & Craven, 2001).

Following the approach used in (Ray & Craven, 2001), we built HMMs for identifying protein names based on words and POS information (and/or the output of the dictionary-based tagger). It involved creating a *positive model* for recognizing sentences containing protein names, and a *null model* for recognizing sentences without any protein names. The positive model was trained on sentences tagged with protein names, while the null model was trained on the remaining sentences. In the extraction phase, a sentence is deemed as positive (i.e. containing protein names) only if the likelihood of emission by the positive model is greater than the likelihood of emission by the null model.

[1] URL: `http://us.expasy.org/sprot/hpi/`

[2] URL: `http://www.godatabase.org/dev/database/`

3.1.4. Token Classification and Support Vector Machines

Since our tagged Medline abstracts do not contain any protein names that directly abut each other, we can reduce the named-entity recognition problem to classification of individual words. Protein names are extracted by identifying the longest sequences of words that have been classified as being part of a protein name. Similar approaches have been applied successfully to the task of *text chunking*, which is identifying simple phrases such as non-recursive noun and verb phrases (Roth & van den Bosch, 2002).

In our experiments, we used support vector machines (SVMs) as the token classifier (Vapnik, 1998).[3] SVMs are generally considered to be the currently best technique for text classification (Joachims, 1998). Building SVMs involves finding an optimal margin hyperplane that separates positive (tokens annotated as proteins) and negative examples with minimal training errors. For each token, we built a feature vector consisting of the current word, the previous and following N words, and their corresponding POS tags (or the output of the dictionary-based tagger). To capture morphological similarities and alleviate the problem of unseen words, we included as features the last one, two, and three characters of each word in the feature vector, which we henceforth refer to as the *suffix* features. We also included as features the class labels of the N preceding tokens. Since the class labels were not given in the test data, they were decided dynamically during the tagging of previous tokens (Kudoh & Matsumoto, 2000). For each extracted sequence of tokens, we used the minimal distance from the separating hyperplane as a quantitative measure of confidence.

The training set for the token classification problem is highly imbalanced. Out of the 209,022 tokens in our corpus, only 10,175 of them (4.87%) are protein names. As pointed out by Kubat et al. (1998), the induced classifiers tend to be highly accurate on negative examples but also produce many false negatives which lead to low recall. By sampling the training set and feeding the learner with only negative examples surrounding the positive ones, we can shift the resulting hyperplane and potentially reduce the number of false negatives. Our experiments supported this claim and showed that we could attain very high recall at the expense of precision.

3.1.5. Existing Protein Name Identification Systems

For comparison, we tested two existing protein name identification systems. The first is KEX version 1.21 (Fukuda et al., 1998), which has a set of hand-built pattern matching rules that makes use of POS information. The second system is Abgene (Tanabe & Wilbur, 2002), which uses a transformation-based tagger to produce initial tagging, and several dictionaries and contextual rules for weeding out false positives and recovering false negatives.

3.2. Experimental Results

3.2.1. Experimental Methodology

The 750 Medline abstracts annotated with protein tags were tokenized using simple pattern rules developed for the Penn Treebank project.[4] For programs requiring sentence-segmented input, we used the sentence segmenter from the KEX tagger with additional rules for bulleted lists. To produce POS tags, we used Brill's POS tagger, which we trained using 10,000 untagged Medline abstracts. Those abstracts were obtained the same way we did for the set of 750 abstracts. No stemming or stopword filtering was performed during the experiments.

We performed ten-fold cross validation on each learning algorithm with a particular parameter setting. Each extracted protein name in the test data was compared to the human-tagged data, with the positions taken into account. Two protein names are considered a match if they consist of the same character sequence in the same position in the text. This detects circumstances where common English words are incorrectly recognized as protein names, and ensures that all references to proteins are recognized.[5] We measured precision (percentage of extracted names that are correct), recall (percentage of correct names that are found), and F-measure (harmonic mean of precision and recall).

3.2.2. Quantitative Results

Table 2 summarizes results for the protein taggers presented, along with any additional sources of information used. For the SVM tagger, we use the full set of negative examples for training, and use $N = 2$ since it provides the best performance. For systems which output confidences that allow trading-off precision and

[3] We have tried a number of classifiers, and SVMs seem to perform the best. Please refer to (Bunescu et al., submitted 2002) for details.

[4] URL: `http://www.cis.upenn.edu/~treebank/`

[5] Since Abgene provides no positional information, we assume that all occurrences of the extracted strings are recognized as proteins.

recall (i.e. BWI, SVM, and HMM), results are presented for the maximum achievable recall.

For ease of comparison, we show precision-recall curves in Figure 1, using the version of each system that gave the best F-measure (as shown in bold in Table 2). For those IE methods that output extraction confidences, we show curves indicating the precision for each achievable level of recall. For the SVM tagger, we achieve higher recall by gradually reducing the number of negative examples in the training set, until only those negative examples adjacent to positive examples remain. At the point of maximum recall (81.82%), 53.17% of the training examples are positive, and we are able to achieve 10.83% precision. Single precision-recall points are shown for all other methods.

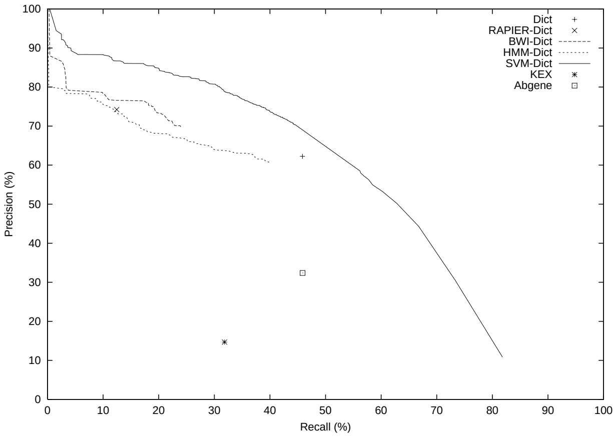

Figure 1. Precision-recall curves for protein taggers in their best settings.

3.2.3. Discussion of Results

Overall, the results show limited utility of POS tags. The use of POS tags in Rapier and SVM does not improve F-measure significantly according to a paired t-test ($p > 0.05$). An exception is for HMM, where the F-measure increases significantly when both the POS tags and the output of the dictionary-based tagger are available. This is because the inclusion of POS tags results in a model with more states, allowing for more accurate modeling of the data. On the other hand, the dictionary-based tagger is generally helpful in boosting precision or recall. The SVM tagger also shows better precision with the inclusion of suffix features.

While none of the learning methods achieve a significant improvement over the dictionary-based tagger in terms of F-measure, several of them can produce much higher precision. In particular, SVM is able to achieve arbitrarily high precision by adjusting the level of recall. This is because the extraction confidence seems to truly reflect the probability of correctness. Since high precision is needed to extract accurate knowledge from text, this is a significant contribution. We can also obtain much higher recall by sampling the training data for use by token classification methods. The proposed proteins may be subject to further filtering to produce a more accurate set.

All of our IE methods perform significantly better than two existing protein taggers, KEX and Abgene. Given that these systems were developed for different distributions of proteins, this is not surprising; however it does illustrate the relative difficulty of identifying human proteins. The hand-built rules used in KEX were developed and tested on a rather confined set of proteins different from the human proteins in our data. Abgene uses transformation-based learning (TBL) to learn a protein tagger; however, the specific tagger we obtained was not trained specifically for human proteins. Our own experiments with TBL showed that even when specifically trained for human proteins, a TBL-based tagger was still inferior to many other learning approaches.[6]

4. Protein Interaction Extraction

This section discusses our work on identifying human-protein interactions assuming that the proteins themselves have already been tagged. We also present comparison of our machine-learning systems with human-written extraction rules.

4.1. IE Methods

4.1.1. Rapier and BWI

In order to adapt slot-filling IE systems that extract individual entities (like Rapier and BWI) to the problem of extracting *relations*, we developed two approaches. The first approach we call the *Interfiller* approach. Given two tagged entities participating in a relationship, the text fragment between them is called the interfiller. If a slot-filling IE system extracts an interfiller, the tagged entities before and after it can be extracted as participating in the targeted relation.

The second approach we call the *Role-filler* approach. In this approach, we extract the two related entities into different role-specific slots. For protein interactions, we named the roles *interactor* and *interactee.* We then assume that all interacting proteins appear in

[6] Please refer to (Bunescu et al., submitted 2002) for details.

Table 2. Performance of protein taggers in different settings.

IE Methods and Additional Information Used	Precision	Recall	F-measure
Dictionary-based with generalized dictionary	62.27%	45.85%	**52.81%**
Rapier			
words only	76.11%	9.97%	17.63%
part-of-speech	70.84%	11.05%	19.12%
dictionary-based tagger	74.49%	12.22%	**21.00%**
BWI (300 iterations, 2 lookaheads, max. recall)			
words only	70.67%	11.52%	19.81%
dictionary-based tagger	71.01%	24.06%	**35.94%**
HMM (max. recall)			
part-of-speech	49.21%	25.93%	33.96%
dictionary-based tagger	51.24%	33.73%	40.68%
part-of-speech and dictionary-based tagger	60.29%	39.95%	**48.05%**
SVM ($N = 2$, full training set, max. recall)			
preceding class labels	69.16%	19.74%	30.72%
preceding class labels and part-of-speech	70.18%	19.72%	30.79%
preceding class labels and dictionary-based tagger	65.00%	45.43%	53.48%
with additional suffix features	70.38%	44.49%	**54.42%**
KEX	14.68%	31.83%	**20.09%**
Abgene	32.39%	45.87%	**37.97%**

the same sentence and extract the related pairs using the following heuristics. (1) The interactors and interactees appearing in the same sentence form a sequence of role fillers. This sequence is separated into segments at the points where an interactee is immediately followed by an interactor. Interactors and interactees can only be paired within the same segment. (2) Each interactor is associated with the next occuring interactee in the segment. (3) If there are fewer interactors (interactees) than interactees (interactors) in the segment, use the last interactor (interactee) in constructing the remaining pairs. In our human-tagged interaction corpus, assuming interactors and interactees are properly tagged, this approach identifies all the interacting pairs with 99.2% accuracy.

Both of these approaches have been used to train BWI to extract interacting proteins, and the Role-filler approach has been used to train Rapier to extract interactions. Rapier could not learn to extract interfillers successfully, since, in the worst case, the time complexity of its generalization algorithm can grow exponentially in the length of a filler. Since extracted entities are usually fairly short, this is typically not a problem in standard slot-filling IE. However, the long interfillers in many protein interactions prevented us from running Rapier with the Interfiller approach.

4.1.2. ELCS

We have also developed a new method for directly learning patterns for extracting relations between previously tagged entities. Blaschke and Valencia (2001; 2002) manually developed rules for extracting interacting proteins. Our method ELCS (Extraction using Longest Common Subsequences) automatically learns such rules from the training data.

ELCS' rule representation is similar to that in (Blaschke & Valencia, 2001; Blaschke & Valencia, 2002), except that it currently does not use POS tags, but allows disjunctions of words. Figure 3 shows some examples of rules learned by ELCS. Words in square brackets separated by '|' indicate disjunctive lexical constraints, i.e. one of the given words must match the sentence at that position. The numbers in parentheses between adjacent constraints indicate the maximum number of unconstrained words allowed between the two (called a *word gap*). A sentence matches the rule if and only if it satisifes the word constraints in the given order and respects the respective word gaps. One of the ELCS' versions allows conjunctions of the sequences of words (last two examples in Figure 3) and in order to match such a rule a sentence must match each of these sequences.

Sentences in the training data that contain interacting proteins are called positive sentences and others are called negative sentences. Note that a positive sentence may contain more than two proteins and more than one pair of interacting proteins. In order to extract the interacting pairs, the rules should be trained to pick out exactly the interacting proteins from the positive sentences. To do this we replicate positive

Sentence 1: The self - association site appears to be formed by interactions between helices 1 and 2 of beta spectrin$_1$ repeat 17 of one dimer with helix 3 of alpha spectrin$_1$ repeat 1 of the other dimer to form two combined alpha - beta triple - helical segments .

Sentence 2: Title - Physical and functional interactions between the transcriptional inhibitors Id3$_2$ and ITF - 2b$_2$.

Generalization using longest common subsequence (LCS):
- (7) interactions (0) between (5) PROT (9) PROT (17) .

Generalization using edit-distance (ED):
[self|Title] (0) - (4) [be|Physical] (0) [formed|and] (0) [by|functional] (0) interactions (0) between (2) [and|the] (0) [2|transcriptional] (0) [of|inhibitors] (0) PROT (8) [of|and] (0) PROT (17) .

Generalization using conjunctions (CJ):
{ - (7) interactions (0) between (5) PROT (9) PROT (17) . } $\bigwedge$ {- (11) and (6) PROT (9) PROT (17) . }

Figure 2. Generalizations of two sentences using different methods. Protein names have been underlined and same sub-script numbers indicate interactions between them. Token 'PROT' stands for potein name.

sentences having n proteins ($n > 2$) into C_2^n sentences such that each one has exactly two of the proteins tagged, with the rest of the protein tags omitted. If the tagged proteins interact, then the replicated sentence is added to the set of positive sentences, otherwise it is added to the set of negative sentences. During testing also we replicate sentences containing more than two protein names in a similar way.

ELCS induces rules using a bottom-up approach. It starts with positive sentences and repeatedly generalizes them to form rules. We have developed three methods for generalizing rules. The first method finds the *longest common subsequence* (LCS) of words between the rules. Efficient algorithms for computing an LCS are presented in (Gusfield, 1997). After finding the LCS between two rules, we determine the size of word gaps between every two adjacent words in their LCS as the larger of the number of words plus the sum of existing word gaps between the two LCS words where they are found in the orignal two rules.

Our second method of generalization uses *edit distance* (ED) (Gusfield, 1997) and creates more specific rules that contain disjunctive constraints. The most common edit distance is Levenshtein distance (Levenshtein, 1966), defined as the minimum number of edit operations (adding, deleting, or replacing an item) required to convert one sequence into another. We use the minimal edit-operation sequence obtained when computing Levenshtein distance to generalize two rules. We preserve the common word constraints between the rules, make disjunctions of constraints when one item is replaced by another in the edit sequence, and drop constraints that are added or deleted in the edit sequence. Finally, we introduce word gaps using the method described for the LCS-based generalization.

interactions (0) between (4) PROT (0) and (4) PROT (16) .

PROT (0) / (0) PROT (10) heterodimers (36) .

[binding | substitution | AB | addition | Interestingly | TI | interactions] (0) [of | - | ,] (3) PROT (19) [to | for | : | same | with] (10) PROT (30) [nM | binding | 1 | CDK6 | CCR8 | death] (9) .

[linker | TI | armadillo | b558 | of] (0) [- | , | a] (5) PROT (13) [and | / | with | to | containing] (0) PROT (2) .

{, (11) PROT (25) and (8) to (9) PROT (66) .} $\bigwedge$ {, (11) PROT (16) bind (18) PROT (66) .}

{, (10) PROT (5) for (7) PROT (9) .} $\bigwedge$ {, (10) PROT (4) binding (6) PROT (9) .}

Figure 3. Some example rules learned by ELCS; the first two were learned using LCS generalization, the next two using ED generalization and the last two using CJ generalization.

The third generalization method finds all common sequences between the two rules and considers their *conjunction* (CJ) as the generalization. Unlike the previous two methods, this method is associative, i.e. we get the same generalization of a set of rules irrespective of the order in which we generalize two of them at a time. If there is any common pattern among the base rules then this property guarantees that the pattern will also appear in the generalization (note that it is possible to lose such a common pattern while taking LCS of two rules at a time). Word gaps are then introduced as in the previous two methods. Figure 2 shows generalization of two sentences obtained by each of these methods.

Using one of these generalization methods, a greedy-covering, bottom-up rule-induction method is used to learn a small set of rules that cover all the positive sentences without covering many negative ones. We use an algorithm similar to beam search and consider only the n best rules for generalization at any time. We start with n randomly selected positive examples. These n rules are generalized with one of the remaining positive examples to obtain n more rules. Out of these $2n$ rules we select n rules with the highest confidence level and allow further generalization with the remaining positive examples. After iterating over the remaining positive examples in this way, the n best rules are finally included in the set of learned rules and the positive examples covered by them are removed. The entire process is repeated till we exhaust the set of positive examples.

We measure the confidence levels of our rules using m-estimate (Cestnik, 1990) which is a measure of expected accuracy of a rule. It is defined as: $\textit{confidence level}(\textit{rule}) = \frac{p+m.p^+}{p+n+m}$, where p and n are the number of positive and negative examples covered by the rule, p^+ is the prior probability of positive examples and m is a parameter which should be set according to the amount of noise in the data. We set p^+ as the fraction of examples in the training data which are positive and set m based on pilot studies. The m-estimate prevents over-fitting as it assigns more confidence to those rules which cover many positive examples along with a few negative examples than to the rules which although cover no negative example but cover a very few positive examples. Figure 3 shows some sample rules learned by ELCS.

4.2. Experimental Results

4.2.1. Experimental Methodology

Medline abstracts were pre-processed as described in Section 3.2.1. All our systems for extracting interactions require sentence segmentation since only two proteins within a sentence are considered when identifying interactions. We also compared our systems with manually-written rules from (Blaschke & Valencia, 2002) which use POS tags and (Blaschke & Valencia, 2001) in which the POS tags are replaced by typical words indicating interactions such as *activation*, *phosphorylation* or *interaction* for nouns and *activates*, *binds* or *phosphorylates* for verbs.

Our current experiments only evaluate the performance of interaction extraction, assuming all protein names have already been correctly tagged. As in Section 3.2.1, performance is evaluated using ten-fold cross validation and measuring recall and precision. We consider an extracted interaction correct if and only if the names and positions of both proteins exactly match the human-tagged interactions.

4.2.2. Quantitative Results

Figure 4 shows precision-recall results for protein-interaction extraction when tested on abstracts that have been manually tagged for protein names. We plotted precision-recall curves for BWI by utilizing its extraction confidences and for ELCS using the confidence levels of the rules which extract the interactions. Since Rapier and human-written rules do not produce confidences, only a single precision-recall point is shown for each of them.

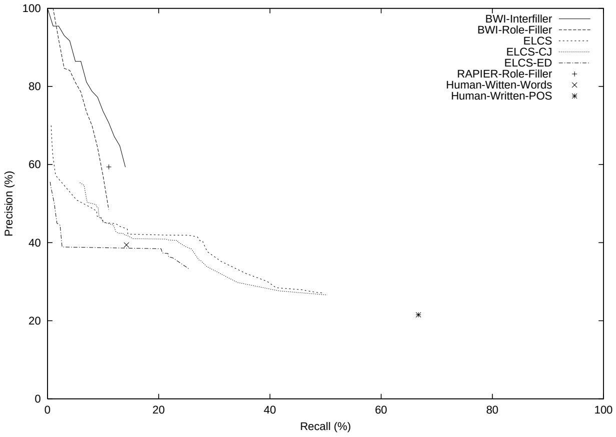

Figure 4. Precision-recall graphs for protein interaction extraction.

4.2.3. Discussion of Results

BWI gives varying degrees of high precision, but its recall is generally quite low. Rapier also gives relatively high precison but low recall. ELCS tends to give higher recall with only a modest decrease in precision compared to BWI and Rapier.

These results generally demonstrate that machine learning out-performs human-written rules. The human-written rules with POS tags give higher recall but very low precision. When the human-written rules are made more precise by replacing the POS tags by specific key words, as in (Blaschke & Valencia, 2001), then the machine learning systems clearly out-perform the human-written rules in terms of both precision and recall. In order to avoid over-loading human curators with too many false positives when extracting knowledge from large volumes of text, a general emphasis towards higher precision seems appropriate. The machine learning systems also offer a wide range of precision-recall trade-off which can be suitably utilized by a user depending upon the need of an application.

5. Conclusions

After comparing a number of methods for extracting human protein names and interactions, we obtained the best performance for protein tagging with an SVM-based token classification method that exploits a generalized protein-name dictionary. For extracting protein interactions, we found that several methods for learning extraction rules generally out-perform handwritten rules, particularly with respect to precision.

Acknowledgements

We would like to thank members of the Marcotte lab for helping tag Medline abstracts. We are grateful to Kristie Seymore, Lorraine Tanabe, Soumya Ray & Mark Craven, Kenichiro Fukuda, Mary Elaine Califf, Dayne Freitag & Nicholas Kushmerick, Thorsten Joachims, Eric Brill, Ellen Riloff and Christian Blaschke for making us available their respective systems/rules. This work was supported in part by the National Science Foundation (IIS-0117308), the Welch Foundation (F-1515), the National Science Foundation (ITR-0219061), and the Texas Advanced Research Program.

References

Bikel, D. M., Schwartz, R., & Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine Learning*, *34*, 211–232.

Blaschke, C., & Valencia, A. (2001). Can bibliographic pointers for known biological data be found automatically? protein interactions as a case study. *Comparative and Functional Genomics*, *2*, 196–206.

Blaschke, C., & Valencia, A. (2002). The frame-based module of the Suiseki information extraction system. *IEEE Intelligent Systems*, *17*, 14–20.

Bunescu, R., Ge, R., Kate, R. J., Marcotte, E. M., Mooney, R. J., Ramani, A. K., & Wong, Y. W. (submitted 2002). Learning to extract proteins and their interactions from Medline abstracts. *Artificial Intelligence in Medicine*.

Califf, M. E., & Mooney, R. J. (1999). Relational learning of pattern-match rules for information extraction. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)* (pp. 328–334). Orlando, FL.

Cardie, C. (1997). Empirical methods in information extraction. *AI Magazine*, *18*, 65–79.

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. *Proceedings of the Ninth European Conference on Artificial Intelligence* (pp. 147–149). Stockholm, Sweden.

Freitag, D., & Kushmerick, N. (2000). Boosted wrapper induction. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)* (pp. 577–583). Austin, TX: AAAI Press / The MIT Press.

Fukuda, K., Tsunoda, T., Tamura, A., & Takagi, T. (1998). Information extraction: Identifying protein names from biological papers. *Proceedings of the 3rd Pacific Symposium on Biocomputing* (pp. 707–718).

Gusfield, D. (1997). *Algorithms on strings, trees and sequences*. New York: Cambridge University Press.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Proceedings of the 10th European Conference on Machine Learning* (pp. 137–142). Berlin: Springer-Verlag.

Kubat, M., Holte, R. C., & Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, *30*, 195–215.

Kudoh, T., & Matsumoto, Y. (2000). Use of support vector learning for chunk identification. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 142–144). Lisbon, Portugal.

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

Ray, S., & Craven, M. (2001). Representing sentence structure in hidden Markov models for information extraction. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)* (pp. 1273–1279). Seattle, WA.

Raychaudhuri, S., Chang, J. T., Sutphin, P. D., & Altman, R. B. (2002). Associating genes with gene ontology codes using a maximum entropy analysis of biomedical literature. *Genome Research*, *12*, 203–214.

Roth, D., & van den Bosch, A. (Eds.). (2002). *Proc. of the sixth conference on natural language learning*. Taipei, Taiwan: Association for Computational Linguistics.

Tanabe, L., & Wilbur, W. J. (2002). Tagging gene and protein names in biomedical text. *Bioinformatics*, *18*, 1124–1132.

Vapnik, V. N. (1998). *Statistical learning theory*. John Wiley & Sons.

Venter, J. C., et al. (2001). The sequence of the human genome. *Science*, *Feb 16;291(5507)*, 1304–1351.