Proceedings of the learning language in logic ICML joint workshop and the challenge task, extracting relations from biomedical texts

James Cussens, Claire Nédellec

Foreword

This Learning Language in Logic workshop (LLL05) (`http://www.cs.york.ac.uk/aig/lll/lll05/`) is the fourth in a series of LLL workshops. Previous workshops took place in Bled (LLL99), Lisbon (LLL00) and Strasbourg (LLL01). This year LLL takes place as part of the 22^{nd} International Conference on Machine Learning (ICML 2005) in Bonn, Germany. We would like to thank the ICML organisers, particularly Hendrik Blockeel, for their help in publicising and organising LLL05.

The purpose of the workshop is to provide a focus for work which applies machine learning using logical representations to natural language. Much existing work in this area uses techniques from inductive logic programming (ILP), and increasingly statistical relational learning. There is currently much interest in applying logic-based machine learning to the problem of Information Extraction from biomedical texts, so the major innovation this year was to have an associated *Genic Interaction Extraction Challenge* (`http://genome.jouy.inra.fr/texte/LLLchallenge/`) organised by the Mathématique, Informatique et Génome (MIG) laboratory at the French National Institute for Agronomics Research (INRA). The challenge organisers made available data consisting of sentences describing interactions between genes and proteins, where proteins are the agents of interaction and genes are the targets. The goal of the challenge was to use this data to induce rules for extracting gene/protein interactions, and a test set was provided by the organisers to evaluate performance on this task.

Significantly, the training data contained considerable linguistic annotation. Basic annotation consisted of word segmentation plus the identification of agents, targets and their interactions. Enriched annotation added lemmas (using a named-entity dictionary) and syntactic dependencies, both manually-checked. Extracting relational events (e.g. an interaction) from text requires this sort of linguistically structured data. We would like to thank all those at MIG-INRA and LIPN-Université Paris 13 who contributed to creating this data and running the challenge competition. This work would not have been possible without funding from the following four projects: the French Caderige project, the French ExtraPloDocs project, the FP6 Alvis project and the FP6 Pascal Network of Excellence.

Six research groups, coming from Germany, the UK, the Netherlands, the Czech Republic and the USA, took up the LLL genic IE challenge and the results of their endeavours can be found in Part II of these proceedings. We would like to thank the scientific committee and additional reviewers for the challenge task who reviewed these papers to a very tight deadline. One of us (CN) has provided an overview of the IE challenge, and the reader is directed to that paper (page 31) for further information.

Although relational IE is a very important application of the LLL approach to learning from natural language it is, of course, not the only one. We are

therefore very pleased that Part I of these proceedings includes three papers not connected to the challenge task. A notable feature of all of these papers is the combination of a symbolic/logical approach to natural language learning (the defining feature of LLL) with other methods: memory-based learning, finite state automata and the EM algorithm, respectively. We would like to thank the LLL05 PC for their help in reviewing these papers and for other valuable input towards the organisation of this workshop.

These proceedings and the individual papers are also available via the LLL bibliography available from the LLL home page `http://www.cs.york.ac.uk/aig/lll/`. Please go to this page for further information on LLL research.

June 2005

James Cussens, University of York, UK
Claire Nédellec, MIG-INRA, France

Contributors to the Challenge task

Scientific committee

Érick Alphonse	MIG-INRA, France
Philippe Bessières	MIG-INRA, France
Christian Blaschke	Alma Bioinformatica, Spain
Fabio Ciravegna	University of Sheffield, UK
Nigel Collier	National Institute of Informatics, Japan
Mark Craven	University of Wisconsin, USA
James Cussens	University of York, UK
Walter Daelemans	University of Antwerp, Belgium
Luc Dehaspe	PharmaDM, Belgium
Rob Gaizauskas	University of Sheffield, UK
Eric Gaussier	Xerox Research Center, France
Udo Hahn	Jena University, Germany
Mélanie Hilario	University of Geneva, Switzerland
Lynette Hirschman	MITRE, USA
Adeline Nazarenko	LIPN-Paris13, France
Jude Shavlik	University of Wisconsin, USA
Junichi Tsujii	University of Tokyo, Japan
Alfonso Valencia	University of Madrid, Spain
Anne-Lise Veuthey	SIB, Switzerland

Additional reviewers

Mark Goadrich	University of Wisconsin, USA
Mark Greenwood	University of Sheffield, UK
José Iria	University of Sheffield, UK
Jee-Hyub Kim	University of Geneva, Swtizerland

Organisation committee

Érick Alphonse	MIG-INRA, France
Sophie Aubin	LIPN-Paris13, France
Jérôme Azé	MIG-INRA & LRI-Paris11, France
Gaël Déral	MIG-INRA, France
Julien Gobeill	MIG-INRA, France
Alain-Pierre Manine	MIG-INRA, France
Thierry Poibeau	LIPN-Paris13, France

Contribution to the data preparation

Biological annotation	Philippe Bessières
XML editor	Gilles Bisson
Data format	Gaël Déral
Syntactic parsing	Sophie Aubin, Érick Alphonse and Julien Gobeil
Named-entity recognition	Gaëtan Lehmann, Gaël Déral and Alain-Pierre Manine

Funding for the challenge task

Research and software development

Caderige Project, Inter-EPST bioinformatics, 1999-2003
ExtraPloDocs, RNTL: 2002-2005
Alvis, FP6-IST-STREP: 2004-2007

Contribution to the data preparation

Pascal FP6-IST-NoE: 2004-2007

LLL05 Programme Committee

Zoltán Alexin	University of Szeged, Hungary
Érick Alphonse	INRA, France
Mary Elaine Califf	Illinois State University, USA
Vincent Claveau	Université de Montréal, Canada
James Cussens	University of York, UK
Walter Daelemans	University of Antwerp, Belgium
Sašo Džeroski	Jožef Stefan Institute, Slovenia
Tomaž Erjavec	Jožef Stefan Institute, Slovenia
Dimitar Kazakov	University of York, UK
Suresh Manandhar	University of York, UK
Claire Nédellec	INRA, France
Luboš Popelínský	Masaryk University, Czech Republic
Stephen Pulman	University of Oxford, UK
Dan Roth	University of Illinois at Urbana-Champaign, USA
Pascale Sébillot	IRISA, France
Jude Shavlik	University of Wisconsin-Madison, USA

Table of Contents

Part I

General LLL papers

Rule Meta-learning for Trigram-Based Sequence Processing

Sander Canisius S.V.M.Canisius@uvt.nl
ILK / Computational Linguistics and AI, Tilburg University, The Netherlands

Antal van den Bosch Antal.vdnBosch@uvt.nl
ILK / Computational Linguistics and AI, Tilburg University, The Netherlands

Walter Daelemans walter.daelemans@ua.ac.be
CNTS, Department of Linguistics, University of Antwerp, Belgium

Abstract

Predicting overlapping trigrams of class labels is a recently-proposed method to improve performance on sequence labelling tasks. In this method, sequence elements are effectively classified three times, therefore some procedure is needed to post-process those overlapping classifications into one output sequence. In this paper, we present a rule-based procedure learned automatically from training data. In combination with a memory-based leaner predicting class trigrams, the performance of this meta-learned overlapping trigram post-processor matches that of a handcrafted post-processing rule used in the original study on class trigrams. Moreover, on two domain-specific entity chunking tasks, the class trigram method with automatically learned post-processing rules compares favourably with recent probabilistic sequence labelling techniques, such as maximum-entropy markov models and conditional random fields.

1. Introduction

Many tasks in natural language processing involve the complex mapping of sequences to other sequences. One typical machine-learning approach to such sequence labelling tasks is to rephrase the sequence-to-sequence mapping task (where a sequence can have a variable length) as a decomposition into a sequence of local classification steps. In each step, one fixed-length feature vector is mapped to an isolated token in the output sequence. After all local classifications have been made, a simple concatenation of the predicted output tokens yields the complete output sequence.

The standard representational approach to decompose sequence processes into local-classification cases, is *windowing*. Within a window, fixed-length subsequences of adjacent input symbols, representing a certain contextual scope, are mapped to one output symbol, typically associated with one of the input symbols, for example the middle one. The fact that the classifier is only trained to associate subsequences of input symbols to single output symbols as accurately as possible is a problematic restriction: it may easily cause the classifier to produce invalid or impossible output sequences, since it is incapable of taking into account any decisions it has made earlier, or even decisions it might have to make further on in the input sequence.

Techniques attempting to circumvent this restriction can be categorised mainly into two general classes. One of those improves upon naive methods by optimising towards the most likely sequence of class labels, rather than the sequence of individually most likely labels. In order to find this most likely sequence in tasks where the class labels of sequence elements are strongly interrelated, it may be necessary to match sequence elements with class labels that are deemed sub-optimal by the underlying classifier, which only bases its decisions on local information. This class of techniques includes recently proposed methods such as maximum-entropy markov models (McCallum et al., 2000) and conditional random fields (Lafferty et al., 2001).

The other class is formed by techniques that rely more on the quality of the predictions made by the underlying classifier; they do not consider label sequences other than those derived from the class labels sug-

Appearing in *Proceedings of the 4^{th} Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

gested by the classifier to be the most-likely ones. Representatives of this class that have been used for sequence labelling tasks in the past are a feedback-loop method as used for example by Daelemans et al. (1996) with memory-based learning, and by Ratnaparkhi (1996) with maximum-entropy modelling, in which previous decisions are encoded as features in the current input of the classifier, and stacking (Wolpert, 1992), a term referring to meta-learning systems that learn to correct errors made by lower-level classifiers.

Predicting overlapping trigrams of class labels, a new method for performing sequence labelling proposed by Van den Bosch and Daelemans (2005), can be seen as a member of the latter class. That is, the technique can be applied to any classifier able to predict a most-likely class label for a given test instance; there is no requirement that the classifier also models the likelihood of other, sub-optimal classes as is required for the methods in the first class. However, with the trigram class method, the final label sequence is not simply obtained by concatenating the classifications for the individual sequence elements. With the trigram class method each sequence element is effectively classified three times; for this reason, some post-processing is required to determine the final label to be assigned to each sequence element.

Van den Bosch and Daelemans (2005) use a simple voting technique with a tie-breaking rule based on classifier-dependent confidence values for this purpose, but emphasise that this is just one possible, and rather simple technique to combine overlapping trigram classes. Other post-proecssing methods may prove to be more appropriate, for example by resulting in better performance scores, or by allowing the full potential of the n-gram class method (with $n > 3$) to be reached. This paper presents one such alternative, based on rule induction where features correspond to various logical assertions about the identity of the overlapping trigrams, or equalities between the three overlapping predictions.

The learned post-processing rules are tested in a pair of benchmark experiments, where a memory-based learner, which was the better performing classifier out of three different classifiers tested by Van den Bosch and Daelemans (2005), is combined with the class trigram method to perform two domain-specific entity chunking task. On these tasks, we evaluate the effect of using rule induction for combining overlapping trigrams and show that the class trigram method with a post-processing method based on rule induction is able to improve upon the baseline performance, with margins comparable to those obtained with a handcrafted combination procedure.

To measure the relative performance of the memory-based learning/class trigram combination, we also perform a series of experiments where three state-of-the-art probabilistic sequence labelling techniques – conditional markov models, maximum-entropy markov models, and conditional random fields – are applied to the same benchmark tasks. On one of the two tasks, the memory-based learner with trigram classes, outperforms all three probabilistic learners; on the other task, conditional random fields prove to be superior, with the memory-based learner ending second.

The structure of the paper is as follows. First, we introduce the two chunking sequence segmentation tasks studied in this paper, in Section 2. Section 3 introduces two approaches for automatically learning class trigram combination rules, and reports on experiments that evaluate the performance of the resulting combination procedures. Next, the class trigram method is empirically compared with three recent probabilistic sequence labelling methods in Section 4. Finally, Section 5 sums up and discusses the main empirical results.

2. Data and Methodology

The two data sets that have been used for this study are examples of sentence-level entity chunking tasks: concept extraction from general medical encyclopedic texts (henceforth MED), and labelling of DNA, RNA, protein, cellular, and chemical terms in MEDLINE abstracts (GENIA). MED is a data set extracted from a semantic annotation of parts of two Dutch-language medical encyclopedias. On the chunk-level of this annotation, there are labels for various medical concepts, such as disease names, body parts, and treatments, forming a set of twelve concept types in total. Chunk sizes range from one to a few tokens. Using a 90%–10% split for producing training and test sets, there are 428,502 training examples and 47,430 test examples.

> *Bij* [*infantiel botulisme*]$_{disease}$ *kunnen in extreme gevallen* [*ademhalingsproblemen*]$_{symptom}$ *en* [*algehele lusteloosheid*]$_{symptom}$ *optreden.*

The GENIA corpus (Tateisi et al., 2002) is a collection of annotated abstracts taken from the National Library of Medicine's MEDLINE database. Apart from part-of-speech tagging information, the corpus annotates a subset of the substances and the biological locations involved in reactions of proteins. Using a 90%–10% split for producing training and test sets, there are

458,593 training examples and 50,916 test examples.

> *Most hybrids express both* $[KBF1]_{protein}$ *and* $[NF\text{-}kappa\ B]_{protein}$ *in their nuclei, but one hybrid expresses only* $[KBF1]_{protein}$.

Apart from having a similar size, both data sets are alike in the sense that most words are outside chunks; many sentences may even contain no chunks at all. Thus, the class distributions of both tasks are highly skewed. In this respect the tasks differ from, for example, syntactic tasks such as part-of-speech tagging or base-phrase chunking, where almost all tokens are assigned a relevant class. However, for all tasks mentioned, whenever chunks are present in a sentence, there is likely to be interaction between them, where the presence of one chunk of a certain type may be a strong indication of the presence of another chunk of the same or a different type in the same sentence.

2.1. Experimental Setup

Van den Bosch and Daelemans (2005) tested their class trigram method with three different classifiers as base classifier; of those three classifiers, memory-based learning performed best; hence, the experiments in Section 3, where the class trigram method is evaluated, are performed using the memory-based learning or k-nearest neighbour algorithm (Cover & Hart, 1967) as implemented in the TiMBL software package (version 5.1) (Daelemans et al., 2004). The combination rules for merging the overlapping class trigrams produced by the base classifier are induced using RIPPER (Cohen, 1995).

The memory-based learning algorithm has algorithmic parameters that bias its performance; for example, the number of nearest neighbours, the distance metric, etc. The optimal values for these parameters may differ depending on the task to be learned. To obtain maximum performance, we optimised the parameter settings on each task using wrapped progressive sampling (WPS) (Van den Bosch, 2004), a heuristic automatic procedure that, on the basis of validation experiments internal to the training material, searches among algorithmic parameter combinations for a combination likely to yield optimal generalisation performance on unseen data. We used wrapped progressive sampling in all experiments.

The experiments described in Section 4 are performed using three different probabilistic sequence learning techniques. Conditional markov models and maximum-entropy markov models have been implemented on top of the maximum-entropy toolkit (version 20041229) by Zhang Le[1]. For conditional random fields, we used the implementation of MALLET (McCallum, 2002).

Instances for all experiments are generated for each token of a sentence, with features for seven-word windows of words and their (predicted) part-of-speech tags. The class labels assigned to the instances form an IOB encoding of the chunks in the sentence, as proposed by Ramshaw and Marcus (1995). In this encoding the class label for a token specifies whether the token is inside (I), outside (O), or at the beginning of a chunk (B). An additional type label appended to this symbol denotes the type of the chunk. The instances are used in exactly this form in all experiments for all algorithms; no feature selection or construction is performed to optimise the instances for a specific task or classifier. Keeping the feature vectors unchanged over all experiments and classifiers is arguably the most objective setup for comparing the results.

Generalisation performance is measured by the F-score ($\beta = 1$) on correctly identified and labelled entity chunks in test data. Experimental results are presented in terms of a mean score, and an approximate 90%-confidence interval; both of those are estimated with bootstrap resampling (Noreen, 1989). Confidence intervals are assumed to be centred around the mean, where the width of the halves at both sides of the mean is taken to be the maximum of the true widths obtained in the resampling process.

3. Predicting Class Trigrams

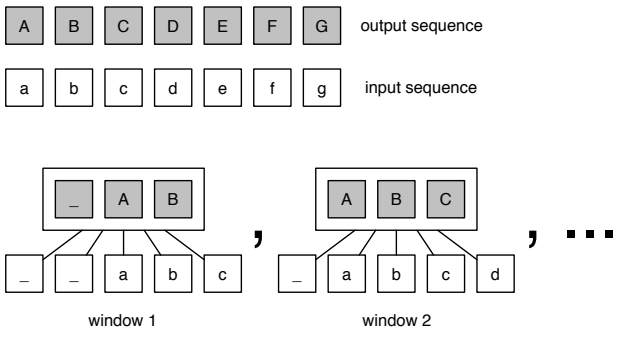


Figure 1. Windowing process with trigrams of class symbols. Sequences of input symbols and output symbols are converted into windows of fixed-width input symbols each associated with, in this example, trigrams of output symbols.

[1] `http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`

As Van den Bosch and Daelemans (2005) argue, there is no intrinsic bound to what is packed into a class label associated to a windowed example. For example, complex class labels can span over trigrams of singular class labels. Figure 1 illustrates the procedure by which windows are created with class trigrams. Each windowed instance maps to a class label that incorporates three atomic class labels, namely the focus class label that was the original unigram label, plus its immediate left and right neighbouring class labels.

While creating instances this way is trivial, it is not entirely trivial how the output of overlapping class trigrams recombines into a normal string of class sequences. When the example illustrated in Figure 1 is followed, each single class label in the output sequence is effectively predicted three times; first, as the right label of a trigram, next as the middle label, and finally as the left label. The redundancy caused by predicting each class label three times may be exploited to do better than the classifier that only predicts each class label once. What is needed then, is a combination procedure that intelligently determines the final class label for each token, given a number of overlapping predictions.

Van den Bosch and Daelemans (2005) propose a simple procedure based on the observation that, in the case of overlapping class label trigrams, it is possible to vote over them. The voting scheme proposed by Van den Bosch and Daelemans (2005) returns the class label which receives the majority of votes (in this case, either two or three), or when all tree votes disagree (i.e. when majority voting ties), returns the class label of which the classifier is most confident. Classifier confidence, needed for tie-breaking, is a classifier-specific metric expressing the classifier's confidence in the correctness of the predicted class; for the memory-based learner it may be heuristically estimated by, for example, taking the distance of the nearest neighbour, which is the approach adopted by (Van den Bosch & Daelemans, 2005).

Clearly this scheme is one out of many possible schemes: other post-processing rules may be used, as well as other values of n (and having multiple classifiers with different n, so that some back-off procedure could be followed). Another interesting possibility, and the approach taken in the current study, is to apply meta-learning to the overlapping outputs of the first-stage classifier, so that a data-driven post-processing procedure is learned automatically from examples extracted from a first-stage classifier producing overlapping class trigrams.

In the current study, we developed two meta-learning combination procedures based on rule induction, in which the instances for the meta-learner describe the overlapping class trigrams in some form, and the classes to be predicted correspond to the combined unigram class label. The first method is a rather straightforward approach in which instances consist of features for the three overlapping class trigrams; the other is a more sophisticated procedure, where features correspond to logical assertions about matches between components of the overlapping trigrams. The exact details of both approaches are described in the remainder of this section, where in addition, their performance is evaluated on the two benchmark tasks MED, and GENIA.

Meta-learning based on class trigram features

A straightforward design for a meta-learner for combining multiple overlapping classifications into a single class label is a classifier trained on instances that simply encode the overlapping classifications as features, and the class labels for these meta-learning instances correspond to the combined classes to be predicted.

To generate training data for the rule inducer, an internal cross-validation experiment has been performed on the training data, resulting in a realistic set of examples of first-stage class trigram outputs. On the training set thus obtained, a rule inducer has been trained, leading to a rule set that predicts a single unigram class label given the three overlapping class trigrams predicted by the first-stage MBL classifier. The performance of this rule set on both the MED, and GENIA entity chunking tasks is presented in Table 1 under the "Learned 1" column, where it is compared with the performance of a unigram-class producing MBL classifier, and with the performance of the handcrafted post-processing procedure of Van den Bosch and Daelemans (2005).

Compared with the baseline performance, the class trigram method with this learned combination procedure attains a substantial performance increase, thereby both confirming the advantage of predicting class trigrams for sequence labelling tasks, and showing the possibility of using a rule inducer to automatically produce combination rules that successfully exploit the redundancy present in the overlapping class trigrams. However, the learned combination rules do not perform better than the handcrafted procedure, which outperforms the learned rules by approximately one point on both tasks.

Table 1. Comparison of generalisation performances of the baseline MBL classifier, and three trigram class approaches using different combination procedures. The best performance per task is printed in bold.

		Trigram post-processing		
Task	Baseline	Handcrafted	Learned 1	Learned 2
MED	64.7 ±0.95	67.5 ±1.09	66.7 ±0.88	**67.7** ±1.07
GENIA	55.7 ±1.14	60.1 ±1.01	58.9 ±1.11	**60.4** ±1.12

Meta-learning based on class overlap features

A huge disadvantage of the previous approach is the fact that the class trigrams encoded as features in the instances can only be treated as atomic symbols by the rule inducer; there is no way for a rule to refer to the left component of a trigram, let alone to draw parallels between a component of one trigram and that of another. To circumvent this limitation, we designed a more fine-grained description of the overlapping trigrams based on logical assertions about the components of the overlapping trigrams, and about matches between two trigrams regarding the class label of the token in focus. For example, the following overlapping trigrams

$t_{-1} = (O, O, A)$
$t_0 = (A, B, A)$
$t_{+1} = (A, O, C)$

would be encoded as follows.

$\neg match\_on\_focuspos(t_{-1}, t_0)$,
$match\_on\_focuspos(t_{-1}, t_{+1})$,
$\neg match\_on\_focuspos(t_0, t_{+1})$,
$\neg all\_agree\_on\_focuspos$,

$focussym(t_{-1}) = A$,
$focussym(t_0) = B$,
$focussym(t_{+1} = A)$

Here, the $focussym$ function returns the component label of the argument trigram describing the token currently in focus; that is, for t_{-1}, it returns the right symbol, for t_0, the middle, and for t_{+1}, the left. The $match\_on\_focuspos$ relation can then be defined as

$match\_on\_focuspos(x, y) \iff$
$focussym(x) = focussym(y)$

By using this representation language for the overlapping class trigrams, the rule inducer is guided to focus on the information about the focus class present in the trigrams. Again, training material for this experiment has been generated by performing an internal cross-validation on the training set.

As can be seen in Table 1 in the column marked "Learned 2", this instance description turns out to be a better choice than our previous attempt, allowing the rule inducer to produce combination rules that outperform those produced by the method described previously. It outperforms both the baseline approach, and the previous meta-learned post-processing method. Compared with the handcrafted method, the meta-learned post-processing procedure appears to perform slightly better, although this difference is nowhere near significance.

Analysis

The experiments described in this section show that rule induction can be used to produce a set of post-processing rules with which the class trigram method can outperform the baseline classifier. However, the fact that the best-performing learned combination procedure does not significantly outperform the handcrafted post-processing rule might seem surprising at first. It should be noted though that the majority voting rule is rather high-level rule, not easily expressed in the description language used in our experiments. Suppose we would replace the confidence-based tie-breaking rule in the handcrafted post-processing procedure by a rule that always selects the focus symbol of the middle trigram. In that case, a logical formulation of the majority voting procedure would probably look like the following decision list.

$\forall X($
$focussym(t_{-1}) = focussym(t_{+1}) = X \Rightarrow X$,
$focussym(t_0) = X \Rightarrow X)$

However, such use of variables is not possible in our description language. The best rule induction can do to match majority voting behaviour is to invent it separately for each class.

$focussym(t_{-1}) = focussum(t_{+1}) = I\text{–}disease \Rightarrow I\text{–}disease$,
$focussum(t_{-1}) = focussum(t_{+1}) = B\text{–}disease \Rightarrow B\text{–}disease$,
...
$focussym(t_0) = I\text{–}disease \Rightarrow I\text{–}disease$,
$focussym(t_0) = B\text{–}disease \Rightarrow B\text{–}disease$,
...

Table 2. Percentage of agreement between handcrafted post-processing and the meta-learned post-processing procedure in case there is a majority vote among the three overlapping class trigrams, and in case there is no such majority.

Task	Majority	No majority
med	99.16	30.80
genia	97.89	29.46

Given that the performance of the best learned combination procedure roughly equals that of the handcrafted post-processing rule, the interesting question is whether rule induction did in fact reinvent majority voting, or whether it produced an entirely different rule set that coincidentally results in comparable performance.

An inspection of the rules produced for med offers a mixed view. For many classes, there are high-priority rules that dictate always believing the middle trigram, be it in various different formulations. For other classes, the highest-priority rule does first check for an overruling majority, before assigning the class suggested by the middle position in the middle trigram. In addition to these rules, which may be interpreted as voting-like behaviour, there are also more original rules that select the class suggested by, for example, the right trigram in favour of the one predicted by the middle trigram; or even rules that might be interpreted as small error correction rules.

For a more definite answer to the question whether the learned rules emulate a majority voting procedure, we computed two overlap metrics between the output of the handcrafted post-processing procedure and that of the meta-learned post-processing procedure: the first measures the percentage of agreement between the two in case there is a majority vote among the three overlapping class trigrams; the other measures the percentage of agreement in case there is no majority. The two metrics computed for both benchmark tasks are listed in Table 2. As can be seen, if there is a majority vote among the overlapping trigrams, there is almost exact agreement between the two different combination procedures. However, in the case of no majority, both methods agree on only 30 percent of the classifications.

These findings lead to the conclusion that the learned combination procedure does indeed implement majority voting. However, it differs from the original handcrafted post-processing rule in the way it deals with ties. This is hardly surprising since the information used for tie-breaking in the original voting rule – classifier confidence – is not available to the rule inducer generating the learned combination procedure.

4. Predicting Class Trigrams versus Probabilistic Output Sequence Optimisation

The class trigram method proposed by Van den Bosch and Daelemans (2005), and evaluated in a slightly modified form in the previous section, is one method for basing decisions for an individual token on the wider sequential context of this token. Both Van den Bosch and Daelemans (2005) and the current study show that predicting class trigrams is an effective method to improve upon a baseline classifier that classifies each token with respect to only a small local context. In order to evaluate the class trigram method with respect to more competitive reference scores, we also compared the method with another popular approach for improving sequence labelling: probabilistic sequence labelling techniques. In this section, three different methods based on this general approach are applied to the sample tasks: conditional markov models, maximum-entropy markov models, and conditional random fields.

4.1. Conditional Markov Model

Conditional markov models (cmm), as used for example by Ratnaparkhi (1996), supplement a standard maximum-entropy model with a feedback loop, in which a prespecified number of previous decisions of the classifier are fed back to the input as features for the current test instance. However, as maximum-entropy models do not simply predict a single most-likely class label, but rather model the entire conditional class probability distribution, classification of a token does not yield one partial labelling, namely, the partial labelling up to the current token followed by the current classification, but as many partial labellings as there are target labels, namely the partial labelling until the current token followed by any of the possible target labels.

As the use of a feedback loop makes the current classification depend on the results of previous classifications, each token in the sequence has to be classified in the context of each possible partial labelling up to that point. Clearly, this approach, if applied naively, gives rise to an exponential increase in possible partial labellings at each token. Therefore, cmms employ a beam search to find the eventual best labelling. With beam search, at each point in time, only a prespecified number of partial labellings – those having highest probability – are considered for expansion, all the

Table 3. Comparison of generalisation performances of the MBL classifier predicting class trigrams, and each of the probabilistic methods. The best performances per task are printed in bold.

TASK	MBL	CMM	MEMM	CRF
MED	**67.7** ±1.07	59.7 ±1.07	60.3 ±1.13	63.4 ±0.95
GENIA	60.4 ±1.12	59.9 ±1.04	56.1 ±1.11	**62.8** ±1.08

other candidates are discarded.

When tested on the two sample tasks, CMM obtains the scores listed in the third column of Table 3. In comparison with the trigram-predicting MBL classifier, it performs considerably worse on MED, but similarly on GENIA.

4.2. Maximum-entropy Markov Model

A more recent probabilistic sequence labelling method is the maximum-entropy markov model (MEMM), proposed by McCallum et al. (2000). Derived from hidden markov models, MEMMs are modelled after a probabilistic state machine, in which, in the simplest case, a state corresponds to the output label of the previous token, and for each state, a separate conditional probability distribution determines the next state, that is, the output label for the current token, given the feature vector of this token. A slight modification of the Viterbi algorithm is used to determine the optimal path through the state machine given the input sequence.

The fourth column of Table 3 shows the performance of MEMM applied to the MED and GENIA tasks. On both tasks, MEMM is outperformed by the MBL classifier. For MED, MEMM's score is quite similar to that of CMM, but on GENIA, MEMM is outperformed by CMM as well.

4.3. Conditional Random Fields

Conditional random fields (CRF) (Lafferty et al., 2001) have been designed to resolve some of the shortcomings of MEMMs. The main difference lies in the number of probabilistic models used for estimating the conditional probability of a sequence labelling: MEMMs use a separate probabilistic model for each state, whereas CRFs have a single model for estimating the likelihood of an entire label sequence. The use of a single probabilistic model leads to a more realistic distribution of the probability mass among the alternative paths. As a result, CRFs tend to be less biased towards states with few successor states than CMMs and MEMMs.

On both sample tasks, CRF attains the highest scores of all three probabilistic methods tested; the last column in Table 3 shows the scores. Compared with MBL, CRF performs considerably worse on MED, but this order is reversed on GENIA, where CRF attains the best score.

5. Conclusion

Classifiers trained on entity chunking tasks that make isolated, near-sighted decisions on output symbols and that do not optimise the resulting output sequences afterwards or internally through a feedback loop, tend to produce weaker models for sequence processing tasks than classifiers that do. The two entity chunking tasks investigated in this paper are challenging tasks; not only because they demand the classifier to be able to segment and label variable-width chunks while obeying the syntax of the chunk analysis, but also because positive examples of labelled chunks are scattered sparsely in the data.

Following Van den Bosch and Daelemans (2005), this paper used a method based on predicting overlapping class trigrams to boost the performance of an MBL classifier on the two entity chunking tasks. Unlike the original work on class trigrams, however, the current study replaced the handcrafted post-processing rules by post-processing procedures learned automatically from labelled example data.

In a series of experiments, the two automatically learned post-processing procedures have been compared with a baseline unigram class predicting classifier, and with a class trigram predicting classifier using the original handcrafted post-processing rule. A meta-learner that simply tries to map the three overlapping class trigrams to a single class unigram was able to improve upon the baseline performance, but in comparison with the handcrafted post-processing rule, its performance was considerably worse.

The other meta-learned post-processing procedure used a description language that was more fine-grained, allowing the rules to refer to component symbols of the trigrams, as well as matches between them. With this representation language, a performance is attained that matches that of the handcrafted post-processing rule. Further analysis of the learned combination rules points out that they implement a voting procedure in quite the same way as the handcrafted

post-processing rule does. However, both methods do differ in the way ties are dealt with. The handcrafted rule bases its decision on classifier confidence, whereas the learned procedure contains separate tie-breaking rules for each different class. Overall, the results show that predicting class trigrams is a useful method to improve upon a baseline classifier predicting unigram classes, and that majority voting is sound method for combining the overlapping class trigrams produced by the base classifier.

In order to evaluate the class trigram method with more competitive reference scores, a number of probabilistic sequence labelling methods have been evaluated on the same entity chunking tasks. On the two benchmark sets, the class trigram method compares rather favourably with the probabilistic methods: on MED, it outperforms all three probabilistic methods by large margin; on GENIA, conditional random fields are the best performing method, MBL with class trigrams ending second, with a performance that is roughly similar to that of the conditional markov model.

These findings suggest that not only is the class trigram method able to improve upon a baseline classifier predicting only unigram classes, an optimised MBL classifier predicting class trigrams followed by a learned combination procedure also performs rather similarly to state-of-the-art probabilistic sequence labelling techniques.

Acknowledgements

The work of the first author is funded by the Netherlands Organisation for Scientific Research (NWO) as part of the NWO IMIX Programme.

References

Cohen, W. (1995). Fast effective rule induction. *Proceedings 12th International Conference on Machine Learning* (pp. 115–123). Morgan Kaufmann.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory, 13*, 21–27.

Daelemans, W., Zavrel, J., Berck, P., & Gillis, S. (1996). MBT: A memory-based part of speech tagger generator. *Proceedings of Fourth Workshop on Very Large Corpora* (pp. 14–27).

Daelemans, W., Zavrel, J., Van der Sloot, K., & Van den Bosch, A. (2004). *TiMBL: Tilburg memory based learner, version 5.1.0, reference guide* (Technical Report ILK 04-02). ILK Research Group, Tilburg University.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the 18th International Conference on Machine Learning*. Williamstown, MA.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proceedings of the 17th International Conference on Machine Learning*. Stanford, CA.

McCallum, A. K. (2002). Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

Noreen, E. (1989). *Computer-intensive methods for testing hypotheses: an introduction*. John Wiley and sons.

Ramshaw, L., & Marcus, M. (1995). Text chunking using transformation-based learning. *Proceedings of the 3rd ACL/SIGDAT Workshop on Very Large Corpora, Cambridge, Massachusetts, USA* (pp. 82–94).

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. *Proceedings of the Conference on Empirical Methods in Natural Language Processing, May 17-18, 1996, University of Pennsylvania*.

Tateisi, Y., Mima, H., Tomoko, O., & Tsujii, J. (2002). Genia corpus: an annotated research abstract corpus in molecular biology domain. *Human Language Technology Conference (HLT 2002)* (pp. 73–77).

Van den Bosch, A. (2004). Wrapped progressive sampling search for optimizing learning algorithm parameters. *Proceedings of the 16th Belgian-Dutch Conference on Artificial Intelligence* (pp. 219–226). Groningen, The Netherlands.

Van den Bosch, A., & Daelemans, W. (2005). Improving sequence segmentation learning by predicting trigrams. *Proceedings of the Ninth Conference on Computational Natural Language Learning*. To appear.

Wolpert, D. H. (1992). Stacked Generalization. *Neural Networks, 5*, 241–259.

Using ILP to learn a domain theory in the form of a FSA

Maria LIAKATA and Stephen PULMAN
Centre for Linguistics and Philology
Walton Street
University of Oxford
U.K.,
maria.liakata@clg.ox.ac.uk, stephen.pulman@clg.ox.ac.uk

June 13, 2005

Abstract

This paper describes a method for inducing a domain theory from a corpus of parsed sentences by means of ILP techniques. A 'domain theory' in the current context stands for a collection of facts and generalisations or rules which describe entities and relations between entities within a domain of interest. As language users, we implicitly draw on such theories in various disambiguation tasks, such as anaphora resolution and prepositional phrase attachment, or to draw inferences. Formal encodings of domain theories can be used for this purpose in natural language processing but they may also be objects of interest in their own right, that is, as the output of a knowledge discovery process. The patterns learnt are represented as FSAs, providing a graphical representation and a more compact format than the original symbolic rules. The approach is generizable to different domains provided it is possible to get logical forms for the text in the domain.

1. Introduction

It is an old observation that in order to choose the correct reading of an ambiguous sentence, we need a great deal of knowledge about the world. Moreover, world knowledge is crucial to all types of linguistic disambiguation and reasoning, as reflected in work such as [Hobbs et al.1993], where hand-coded pragmatic inference rules are used for linguistic interpretation in naval operation and terrorist reports. Rather than constructing such rules manually, which is time consuming and difficult to replicate, the current paper proposes a method for learning automatically from text a set of domain rules describing associations between two or more verbs and their respective arguments. Note that this is a different goal from merely obtaining selectional preferences for verbs as we are primarily interested in detecting more long range dependencies between verbs.

2. Some background

[Pulman2000] describes a method for obtaining a theory for prepositional phrase disambiguation. This approach inverts the observation that disambiguation decisions depend on knowledge of the world and showed that it is possible to learn a simple domain theory from a disambiguated corpus by capitalising on the information tacitly contained in the disambiguation decisions. Ambiguous sentences from a subset of the ATIS (air travel information service) corpus [Doddington and Godfrey1990] were annotated so as to indicate the preferred reading, e.g.

```
[do,they,[serve,a,meal],on,
[the,flight,from,san_francisco,to,atlanta]]
```

The 'good' and the 'bad' parses were used to produce simplified first order logical forms representing the semantic content of the various readings of the sen-

Appearing in *Proceedings of the 4^{th} Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

tences. The 'good' readings were used as positive evidence, and the 'bad' readings (or more accurately, the bad parts of some of the readings) were used as negative evidence. Next the Inductive Logic Programming algorithm, Progol [Muggleton1995], was used to learn a theory of prepositional relations in this domain: i.e. what kinds of entities can be in these relations, and which cannot.

Among others generalisations like the following were obtained (all variables are implicitly universally quantified):

$$fare(A) \wedge airline(B) \quad \rightarrow on(A,B)$$
$$meal(A) \wedge flight(B) \quad \rightarrow on(A,B)$$

This domain theory was then used successfully in disambiguating a small held-out section of the corpus, by checking for consistency between logical forms and domain theories.

While the numbers of sentences involved in that experiment were too small for the results to be statistically meaningful, the experiment proved that the method works in principle. Moreover, the results of the theory induction process are perfectly comprehensible - the outcome is a theory with some logical structure, rather than a black box.

3. Current Approach

The current paper seeks to extend this approach and at the same time derive less specialised rules that can be useful in a variety of tasks ranging from different types of linguistic disambiguation to reasoning (e.g. in question-answering). The method requires a fully parsed corpus with corresponding logical forms. We have experimented with larger datasets, using the Penn Tree Bank [Marcus et al.1994] since the syntactic annotations for sentences given there are intended to be complete enough for semantic interpretation, in principle, at least.

In practice, [Liakata and Pulman2002] report, it is by no means easy to do this. It is possible to recover partial logical forms from a large proportion of the treebank, but these are not complete or accurate enough to simply replicate the ATIS experiment. In the work reported here, we selected about 40 texts containing the verb 'resign', all reporting, among other things, 'company succession' events, a scenario familiar from the Message Understanding Conference (MUC) task [Grishman and Sundheim1995]. The texts amounted to almost 4000 words in all. Then we corrected and completed some automatically produced logical forms by hand to get a fairly full representation of the meanings of these texts (as far as is possible in first order logic). We also resolved by hand some of the simpler forms of anaphoric reference to individuals to simulate a fuller discourse processing of the texts.

To give an example, a sequence of sentences like:

J.P. Bolduc, vice chairman of W.R. Grace & Co. (...) was elected a director. He succeeds Terrence D. Daniels,... who resigned.

was represented by the following sequence of literals:

```
verb(e1,elect).
funct_of('J.P._Bolduc',x1).
...
subj(e1,unspecified).
obj(e1,x1).
description(e1,x1,director,de1).
verb(e5,succeed).
subj(e5,x1).
funct_of('Terrence_D._Daniels',x6).
obj(e5,x6).
verb(e4,resign).
subj(e4,x6).
```

The representation is a little opaque, for implementation reasons relating to the input settings of the learning mechanism. The above example can be paraphrased as follows: there is an event, e1, of electing, the subject of which is unspecified, and the object of which is x1. x1 is characterised as 'J P Bolduc', and e1 assigns the description de1 of 'director' to x1. There is an event e5 of succeeding, and x1 is the subject of that event. The object of e5 is x6, which is characterised as Terrence D Daniels. There is an event e4 of resigning and the subject of that event is x6.

The reason for all this logical circumlocution is that we are trying to learn a theory of the 'verb' predicate, in particular we are interested in relations between different verbs as well as associations between verbs and their argument since these may well be indicative of causal or other regularities that should be captured in

the theory of the company succession domain. If the individual verbs were represented as predicates rather than arguments of a 'verb' predicate we would not be able to generalise over them: we are restricted to first order logic, and this would require higher order variables.

We also need to add some background knowledge. We assume a fairly simple flat ontology. It is also possible to work with hierarchical ontologies, for example WordNet. Indeed, in a version of our experiment we obtained all hypernym chains for verbs and configured the learning algorithm to add a verb concept to each iteration, from most general to specific, until the pattern would fail to exceed a certain threshold. However, this approach led to overly general, uninformative patterns. For the purpose of this experiment, it was not deemed necessary to repeat the same process with verb arguments.

Entities were assigned to classes semi-automatically, using clustering techniques described in [Liakata2004] followed by manual adjustment. The latter involved merging the 32 classes resulting from the automatic clustering into 11 principal categories, namely the following:

company, financial instrument, financial transaction, location, money, number, person, company position, product, time, and unit (of organisation).

As with the 'verb' predicate, the representation has these categories as an argument of a 'class' predicate to enable generalisation:

```
class(person,x1).
class(company,x3).
etc.
```

Ideally, to narrow down the hypothesis space for ILP, we need some negative evidence. In the Penn Tree Bank, though, only the good parse is represented. There are several possible ways of obtaining negative data: one could use a parser trained on the Tree Bank to reparse sentences and recover all the parses. However, there still remains the problem of recovering logical forms from 'bad' parses. An alternative would be to use a kind of 'closed world' assumption: take the set of predicates and arguments in the good logical forms, and assume that any combination not observed is actually impossible. One could generate artificial negative evidence this way but one risks overgenerating and rejecting otherwise plausible combinations.

Alternatively, one can try learning from positive only data. The ILP systems Progol [Muggleton1995] and Aleph [Srinivasan1999] are able to learn from positive only data, with the appropriate settings. Likewise, so-called 'descriptive' ILP systems like WARMR [Dehaspe1998] do not always need negative data: they are in effect data mining engines for first order logic, learning generalisations and correlations in some set of data.

4. Learning rules for company succession events

We found that the most successful method, given the absence of negative data, was to use WARMR to learn association rules from the positive data. WARMR is very closely related to the levelwise [Mannila and Toivonen1997] and the APRIORI algorithms [Agrawal et al.1996]. The latter assume a lattice of data, denoting relations between binary valued attributes and the algorithm performs a general-to-specific, breadth-first search, looking at one level of the data lattice at a time. The patterns are specialised at each level, by the addition of an extra attribute-value pair. The method iterates between candidate generation and candidate evaluation. The algorithm implemented in WARMR is a version of the levelwise algorithm where binary attributes have been replaced by predicate calculus literals.

The input to WARMR consists of the evidence (sentences) represented in terms of models, where each model can span over several sentences, thus increasing the possibilities of associations between verbs, which constitute our main point of interest. The language bias is made available as a set of type and mode declarations, called rmodes. The rmodes define a set of literal lists, which are interpreted by the system. WARMR rmodes are interpreted as queries and checked against the evidence at each iteration of the algorithm.

Examples of rmodes are:

```
%1
```

```
rmode(3:verb(-Sub,\Event,#1)).

%2
rmode((attr(#1,+Event,\ID), class(#1,ID))).
```

The first rmode declaration says that a likely pattern is expected to contain up to three verbs. The '\ Event' variable means that the verb introduced should correspond to a new event variable whose value is different from others of the same type. The '#1' symbol is a way of denoting that the corresponding slot is a constant, generated from the data. The second rmode says that a likely pattern should include an attribute along with its class information.

As with all types of association rule learning, WARMR produces a huge number of rules, of varying degrees of coverage. We spent some time writing filters to narrow down the output to something useful. Such filters consist of constraints ruling out patterns that are definitely not useful, for example patterns containing a verb but no arguments or attributes. An example of such a restriction is provided below:

```
pattern_constraint(Patt):-
     member(verb(_,E,_A,_,_),Patt),
     (member(attr(_,E,Attr),Patt)
      ->
      \+constraint_on_attr(Patt,Attr)).
```

If $pattern\_constraint/1$ succeeds for a pattern Patt, then Patt is discarded. Basically, this says that a rule isn't useful unless it contains a verb and one of its attributes that satisfies a certain constraint. A constraint might be of the following form:

```
constraint_on_attr(Patt, Attr) :-
      member(class(_,Attr), Patt).
```

The above states that there should be a classification of the attribute Attr present in the rule. A useful pattern Patt will satisfy such constraints.

Some of the filtered output, represented in a more readable form compatible with the examples above are as follows (note that the first argument of the verb/2 predicate refers to an event):

Companies report financial transactions:

$subj(B, C) \wedge obj(B, D) \wedge$
$class(fin\_tran, D) \wedge class(company, C) \rightarrow verb(B, report)$

Companies acquire companies:
$subj(B, C) \wedge obj(B, D) \wedge class(company, D) \wedge class(company, C) \rightarrow verb(B, acquire)$

Companies are based in locations:

$obj(A, C) \wedge class(company, C) \wedge in(A, D) \wedge class(location, D) \rightarrow verb(A, base)$

If person C succeeds person E, then someone has elected person C:

$obj(A, C) \wedge class(person, C) \wedge$
$verb(D, succeed) \wedge$subj(D,C) $\wedge obj(D, E) \wedge$
$class(person, E) \rightarrow verb(A, elect)$

If someone elects person C, and person D resigns, then C succeeds D:

$subj(G, C) \wedge verb(A, elect) \wedge obj(A, C) \wedge class(person, C) \wedge verb(E, resign) \wedge$
$subj(E, D) \wedge class(person, D) \rightarrow$
$verb(G, succeed)$

While there are many other rules learned that are less informative than this, the samples given here are true generalisations about the type of events described in these texts: unremarkable, perhaps, but characteristic of the domain. It is noteworthy that some of them at least are very reminiscent of the kind of templates constructed for Information Extraction in this domain, suggesting a possible further use for the methods of theory induction described here.

5. Representing the Output

Each of the numerous patterns resulting from WARMR consists of a list of frequently associated predicates, found in the flat quasi-logical forms of the input sentences. An example of such a pattern is provided by the following:

```
freq(6,[verb(A,B,elect,p,d),
        verb(C,D,succeed,p,d),
        attr(subj,B,unspecified),
        attr(obj,D,E),class(cperson,E),
        attr(subj,D,F),class(cperson,F),
```

```
      attr(obj,B,F)],
    0.1463).
```

The first argument of the predicate $freq/3$ shows the level of the algorithm at which the pattern/query was acquired [Dehaspe1998]. The fact that the pattern was acquired at the sixth level means it was created during the sixth iteration of the algorithm trying to satisfy the constraints input as settings to the system. This pattern satisfied four constraints, two of them twice[1]. The second argument of $freq/3$ is the query itself and the third is its frequency. What is meant by frequency of the query in this instance is the number of times it succeeds (i.e. the number of training examples it subsumes), divided by the number of training examples. To illustrate the meaning of such a pattern one needs to reconstruct the predicate-argument structures while maintaining the flat format. Thus, the above pattern is converted to the following:

```
list(529,0.1463,[elect(A,B,C),
                 cperson(C),
                 succeed(D,C,E),
                 cperson(E)]).
```

It is now easier to understand the pattern as :'A person C who is elected succeeds a person E'. However, it is still not straightforward how one can evaluate the usefulness of such patterns or indeed how one can incorporate the information they carry into a system for disambiguation or reasoning. This problem is further aggravated by the large number of patterns produced. Even after employing filters to discard patterns of little use, for example ones containing a verb but no classification of its arguments, over 26,000 of them were obtained. Experimenting with a more restrictive language bias is an option but our experience was that there is a significant trade off between complex rmodes and algorithm efficiency. We found that it was better to relax the language settings and allow WARMR to perform more iterations, while at the same time requiring more extensive filtering of the output.

The large size of the output is mainly due to the fact that many patterns are overly general: the training set consists of only 372 verb predicates and a total of 436 clauses. Such overgeneration is a well known problem of data mining algorithms and requires sound criteria for filtering and evaluation. Most of the patterns generated are in fact variants of a much smaller group of patterns. The question then arises of how it is possible to merge them so as to obtain a small number of core patterns, representative of the knowledge obtained from the training set. Representing the patterns in a more compact format also facilitates evaluation either by a human expert or through incorporation into a pre-existing system to measure improvement in performance.

[1]There are eight literals in the pattern, even though it was obtained in round 6. This is because the rmode satisfied at rounds 4 and 5 adds two literals simultaneously.

6. FSA conversion

Given the large amount of shared information in these outputs, we decided to try to represent it as a set of Finite State Automata, where each transition corresponds to a literal in the original clauses[2] Since all the literals in the raw output are simply conjoined, the interpretation of a transition is simply that if one literal is true, the next one is also likely to be true. Our aim was to be able to use standard FSA minimisation and determination algorithms [Aho et al.1986],[Aho et al.1974] to reduce the large set of overlapping clauses to something manageable and visualisable, and to be able to use the frequency information given by WARMR as the basis for the calculation of weights or probabilities on transitions.

To convert our patterns into FSAs (and in particular recognizers), we used the package FSA Utilities (version FSA6.2.6.5)[van Noord2002], which includes modules for compiling regular expressions into automata (recognizers and transducers) by implementing different versions of minimisation and determinisation algorithms. The package also allows operations for manipulating automata and regular expressions such as composition, complementation etc. As the FSA Utilities modules apply to automata or their equivalent regular expressions, the task required converting the patterns into regular expressions. To do this we treat each literal as a symbol. This means each verb and attribute predicate with its respective

[2]One possibility would have been to employ Galois lattices to merge together patterns subsuming each other. In that case single predicates or tuples would be used as descriptors-attributes of the patterns-objects. However, weighted FSAs present a more straightforward way of designating the dependencies between patterns and preserve the ordering of literals in the pattern.

arguments is taken to denote a single symbol. The literals are implicitly conjoined and thus ordering does not matter. Thus we chose to impose an ordering on patterns, whereby the main verb appears first, followed by predicates referring to its arguments. Any other verbs come next, followed by predicates describing their arguments. This ordering has the advantage over alphanumeric ordering that it allows filtering out alphabetic variants of patterns where the predicates referring to the arguments of a verb precede the verb and the variables are thus given different names which results in different literals. This ordering on patterns is useful as it allows common prefixes to be merged during minimisation. Since variable names play an important role in providing co-indexation between the argument of a verb and a property of that argument, designated by another predicate, terms such as '$elect(A, B, C)$' and '$elect(D, E, F)$' are considered to be different symbols. Thus a pattern like:

```
list(768,0.07,[elect(A,B,C),cperson(C),
               chairman(C,D),old(C,E,F),
               of(D,G),ccompany(G)]).
```

was converted to the regular expression:

```
macro(x768,['elect(A,B,C)',
            'cperson(C)',
            'chairman(C,D)',
            'old(C,E,F)',
            'of(D,G)',
            'ccompany(G)']).
```

The first argument of the $macro/2$ predicate is the name of the regular expression whereas the second argument states that the regular expression is a sequence of the symbols 'elect(A,B,C)','cperson(C)','chairman(C,D)' and so on. Finally, the entire WARMR output can be compiled into an FSA as the regular expression which is the union of all expressions named via an xnumber identifier. This is equivalent to saying that a pattern can be any of the xnumber patterns defined.

We took all the patterns containing 'elect' as the main verb and transformed them to regular expressions, all of which started with 'elect(A,B,C)'. We then applied determinisation and minimisation to the union of these regular expressions. The result was an automaton of 350 states and 839 transitions, compared to an initial 2907 patterns.

However, an automaton this size is still very hard to visualize. To circumvent this problem we made use of the properties of automata and decomposed the regular expressions into subexpressions that can then be conjoined to form the bigger picture. Patterns containing two and three verbs were written in separate files and each entry in the files was split into two or three different segments, so that each segment contained only one verb and predicates referring to its arguments. Therefore, an expression such as:

```
macro(x774,[elect(A,B,C),cperson(C),
            resign(D,E,F),cperson(E),
            succeed(G,C,E)]).
```

was transformed into:

```
macro(x774a,['elect(A,B,C)',
             'cperson(C)']).
macro(x774b,['resign(D,E,F)',
             'cperson(E)']).
macro(x774c,['succeed(G,C,E)']).
```

One can then define the automaton xpression1, consisting of the union of all first segment expressions, such as x774a, the automaton resign2, consisting of all expressions where resign is the second verb and succeed3. The previous can be combined to form the automata $[xpression1, resign2]$ or $[xpression1, resign2, succeed3]$ and so on. The automaton $[xpression1, resign2]$ which represents 292 patterns, has 32 states and 105 transitions and is much more manageable.

7. Adding weights

The FSA rules derived from the WARMR patterns would be of more interest if weights were assigned to each transition, indicating the likelihood of any specific path/pattern occurring. For this we needed to obtain weights, equivalent to probabilities for each literal/predicate-argument term. Such information was not readily available to us. The only statistics we have correspond to the frequency of each entire pattern, which is defined as:

$$Freq = \frac{\text{number of times the pattern matched the training data}}{\text{number of examples in the training set}}$$

We took this frequency measure as the probability of patterns consisting of single predicates (e.g. 'elect(A,B,C)', which is equivalent to 'B elects C')

whereas the probabilities of all other pattern constituents have to be conditioned on the probabilities of terms preceding them. Thus, the probability of 'cperson(C)', given 'elect(A,B,C)' is defined by the following:

$$P('cperson(C)'|'elect(A,B,C)') = \frac{P('elect(A,B,C)','cperson(C)')}{P('elect(A,B,C')}$$

where $P('elect(A,B,C)','cperson(C))'$ is the frequency of the pattern $['elect(A,B,C)','cperson(C)']$ and $P('elect(A,B,C)')$ is defined as:

$$P('elect(A,B,C)') = \sum_X P('elect(A,B,C)',X)$$

That is, the probability of $P('elect(A,B,C)')$ is the sum of all the probabilities of the patterns that contain 'elect(A,B,C)' followed by another predicate.

In principle the frequency ratios described above are probabilities but in practice, because of the size of the dataset, they may not approximate real probabilities. Either way they are still valid quantities for comparing the likelihood of different paths in the FSA.
Having computed the conditional probabilities/weights for all patterns and constituents, we normalized the distribution by dividing each probability in a distribution by the total sum of the probabilities. This was necessary in order to make up for discarded alphabetic variants of patterns. We then verified that the probabilities summed up to 1. To visualise some of the FSAs (weighted recognizers) we rounded the weights to the second decimal digits and performed determinization and minimization as before. Rules obtained can be found in Figures 1 and 2 (see figures on last page):

The automaton of Figure 1 incorporates the following rules:

1. 'If a person C is elected, another person E has resigned and C succeeds E'

2. 'If a person C is elected director then another person F has resigned and C succeeds F'

3. 'If a person C is elected and another person E pursues (other interests) C succeeds E'

The automaton of Figure 2 provides for rules such as: 'If a person is elected chairman of a company E then C succeeds another person G'.

At each stage, thanks to the weights, it is possible to see which permutation of the pattern is more likely.

8. Related Work

Rules such as the above express causality and interdependence between semantic predicates, which can be used to infer information for various linguistic applications. The idea of deriving inference rules from text has been pursued in [Lin and Pantel2001] as well, but that approach differs significantly from the current one in that it is aimed mainly at discovering paraphrases. In their approach text is parsed into paths, where each path corresponds to predicate argument relations and rules are derived by computing similarity between paths. A rule in this case constitutes an association between similar paths. This is quite different to the work currently presented, which provides more long range causality relations between different predicates, which may not even occur in adjacent sentences in the original texts. Other approaches such as [Collin et al.2002] also aim to learn paraphrases for improving a Question-Answering system. Our work is perhaps more closely related to the production of causal networks as in [Subramani and Cooper1999], where the goal is to learn interdependency relations of medical conditions and diseases. In their work the dependencies only involve key words, but we believe that our techniques could be applied to similar biomedical domains to discover causal theories with richer inferential structure.

9. Conclusions & Future Work

We have shown that it is possible to induce logically structured inference rules from parsed text. We have also shown that by using FSA techniques it is possible to construct a weighted automaton for the representation of rules/patterns generated via a knowledge mining process. This enables merging together permutations of the same pattern and facilitates human evaluation of the pattern. Furthermore, the fact that we have

learned what is in effect a simple probabilistic graphical model means that we can now produce representations of this knowledge suitable for more robust inference methods of the type that we can deploy to aid reasoning and disambiguation tasks. The current approach is also interesting as a means of obtaining new knowledge, through previously undetected associations (e.g. if applied to medical texts). Patterns acquired would then serve the purpose of bringing new insight to the knowledge expertise in question.

10. Acknowledgements

We would particularly like to thank Ashwin Srinivasan (IBM, New Delhi), Steve Moyle (Oxford), and James Cussens (York) for their help with Aleph and Jan Struyf, Hendrik Blockeel and Jan Ramon (K.U. Leuven), for their generous help with WARMR.

References

[Agrawal et al.1996] R. Agrawal, H. Mannil, R. Srikant, H. Toivonen, and A.I. Verkamo. 1996. Fast discovery of association rules. In P. Smyth U.M. Fayyad, G. Piatetsky-Shapiro and R. Uthurusamy, editors, *Adevances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA.

[Aho et al.1974] A.H. Aho, J.E. Hopcroft, and J.D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company.

[Aho et al.1986] A.H. Aho, R. Sethi, and J.D. Ullman. 1986. *Compilers - Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, USA.

[Collin et al.2002] O. Collin, F. Duclaye, and F. Yvon. 2002. Learning Paraphrases to Improve a Question-Answering System. *staff.science.uva.nl/ mdr/NLP4QA/10duclaye-et-al.pdf*.

[Dehaspe1998] Luc Dehaspe. 1998. *Frequent Pattern Discovery in First-Order Logic*. Ph.D. thesis, Katholieke Universiteit Leuven.

[Doddington and Godfrey1990] G. Doddington and C.H.J. Godfrey. 1990. The ATIS Spoken Language Systems Pilot Corpus. In *Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania.

[Grishman and Sundheim1995] R. Grishman and B. Sundheim. 1995. "Message Understanding Conference-6: A Brief History". *www.cs.nyu.edu/cs/projects/proteus/muc/muc6-history-coling.ps*.

[Hobbs et al.1993] J. Hobbs, M. Stickel, D. Appelt, and P. Martion. 1993. Interpretation as abduction. *Artificial Intelligence*, 63:69–142.

[Liakata and Pulman2002] M. Liakata and S. Pulman. 2002. From Trees to Predicate-Argument Structures. In *International Conference for Computational Linguistics (COLING)*, pages 563–569, Taipei, Taiwan.

[Liakata2004] M. Liakata. 2004. Inducing domain theories. D.Phil. thesis.

[Lin and Pantel2001] D. Lin and P. Pantel. 2001. Dirt-Discovery of Inference Rules from Text. In *In ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 323–328.

[Mannila and Toivonen1997] H. Mannila and H. Toivonen. 1997. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining Knowledge Discovery*, 1(3):241–258.

[Marcus et al.1994] M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*.

[Muggleton1995] Stephen Muggleton. 1995. Inverse Entailment and Progol. *New Generation Computing, special issue on Inductive Logic Programming*, 13(3-4):245–286.

[Pulman2000] Stephen Pulman. 2000. Statistical and Logical Reasoning in Disambiguation. *Philosophical Transactions of the Royal Society*, 358 number 1769:1267–1279.

[Srinivasan1999] Ashwin Srinivasan. 1999. "the Aleph Manual". *www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*.

[Subramani and Cooper1999] M. Subramani and G.F. Cooper. 1999. Causal Discovery from Medical Textual Data. *http://www.amia.org/pubs/symposia/D200558.PDF*.

[van Noord2002] Gertjan van Noord. 2002. FSA6 Reference Manual. *http://odur.let.rug.nl/ vannoord/Fsa/*.

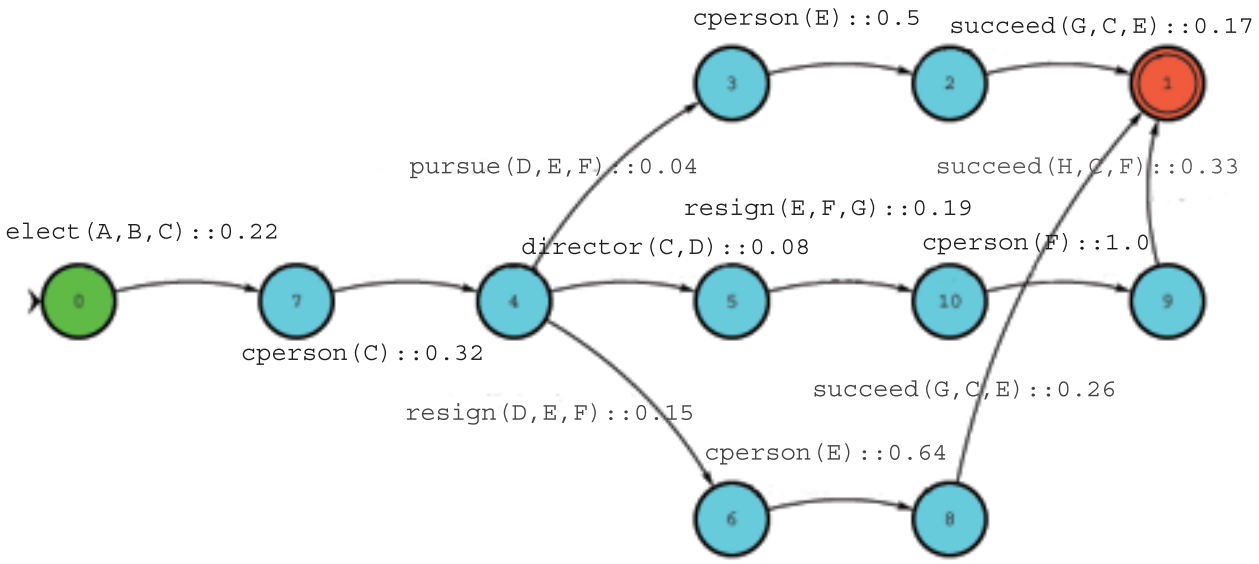


Figure 1. The more likely path in this FSA segment is given by the choice of $resign(D, E, F) : 0.15$, followed by $cperson(E) : 0.64$ and finally $succeed(G, C, E) : 0.26$. This can be interpreted as follows: 'If a person C is elected, another person E has resigned and C succeeds E'

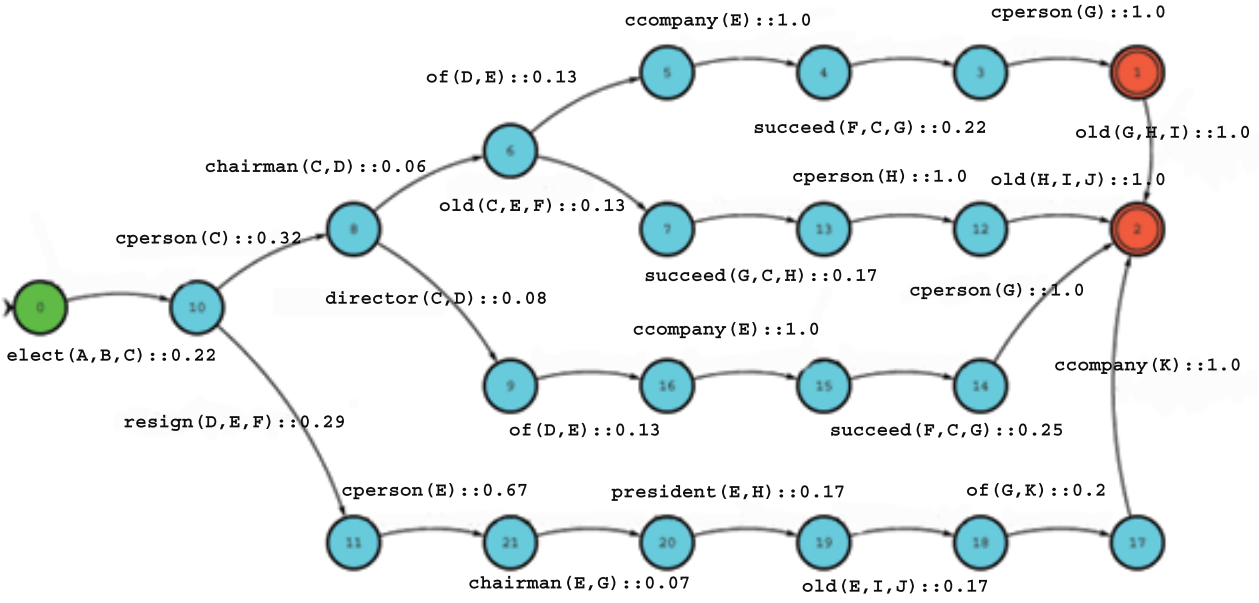


Figure 2. Here the more likely path is provided by the sequence:
$cperson(C) : 0.32, director(C, D) : 0.08, of(D, E) : 0.13, company(E) : 1, succeed(F, C, G) : 0.25, cperson(F) : 1$. This can be read as: 'If a person C is elected director of a company E then C succeeds another person G'.

Notice the above illustrate only parts of the FSAs, which justifies why the probabilites of arcs leaving a node don't add up to 1

A Generic Approach to EM learning for Symbolic-statistical Models

Taisuke Sato SATO@MI.CS.TITECH.AC.JP
Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo, Japan

Abstract

We present a generic approach to EM learning (i.e. parameter learning by ML estimation for probabilistic models using the EM algorithm). We write domain-dependent programs in a symbolic-statistical modeling language PRISM to define distributions but apply a common EM algorithm to them. Differences in EM learning for each model are subsumed by differences in programs, and hence we have only to write PRISM programs to perform EM learning even for new probabilistic models and the rest of task is automatically carried out by the PRISM's learning routine.

1. Introduction

Parameter learning by ML (maximum likelihood) estimation for probabilistic models is one of the most basic techniques for machine learning and the EM algorithm (Dempster et al., 1977) has been a primary tool for ML estimation. So far however, EM learning, i.e. parameter learning by the EM algorithm has been carried out domain-dependently. In other words one has to use different EM algorithms for different applications or when no EM algorithm is available, he or she has to invent a new EM algorithm.

In this paper we present a different approach with the following features. First we use a single EM algorithm called the *gEM (graphical EM) algorithm* (Kameya & Sato, 2000) for every application regardless of whether it belongs to a known model or not. Second we use a symbolic-statistical modeling language PRISM[1] and write a program to define a parameterized distribution for each probabilistic model. To learn parameters of the distribution, we first apply the program to data and extract a certain AND-OR graph called *explanation graph* representing the likelihood function, and then run the gEM algorithm on it. So differences in distributions are reflected on differences in programs and do not affect the EM algorithm itself. Put it differently we do not need to invent a new EM algorithm even for a new model. Third computationally speaking, a kind of dynamic programming is built-in in PRISM and our approach can be reasonably competitive.[2]

We proved in (Sato & Kameya, 2001) that PRISM programs representing popular symbolic-statistical models, i.e. singly connected BNs (Bayesian networks), HMMs (hidden Markov models) and PCFGs (probabilistic context free grammars) have the same time complexity in EM learning as their specialized counterparts, i.e. EM learning by Pearl's belief propagation (Pearl, 1988), the Baum-Welch algorithm (Rabiner & Juang, 1993) and the Inside-Outside algorithm (Manning & Schütze, 1999) respectively. In addition in the case of PCFGs, it is experimentally confirmed that the gEM algorithm runs faster than the Inside-Outside algorithm by orders of magnitude provided explanation graphs are given (Sato & Kameya, 2001).

We have tested other known and unknown EM learning by PRISM programs in addition to these well-known models to see the strength and weakness of our approach. The list includes Naive Bayes, linkage analysis (Lander-Green algorithm (Kruglyak et al., 1996), Elston-Stewart algorithm (Elston & Stewart, 1971)), profile HMMs for alignment, generalized LR(k) parsing (Inui et al., 1997), game modeling and most recently left-corner parsing (Manning, 1997) and context free graph grammars (Rozenberg, 1997). What has become clear from these modeling experiences is that our approach fits well in the development stage of new statistical models because we can explore and test them at higher level without the additional trouble of implementing EM algorithms from scratch. Also continuing efforts for efficient implementation of PRISM

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

[1] URL = `http://sato-www.cs.titech.ac.jp/prism/`

[2] However we do not claim that it is as efficient as EM learning by a specialized EM algorithm implemented in C.

(Zhou et al., 2004) made it usable in such a stage.[3] In this paper, we show how to perform EM learning, using PRISM, for a stochastic left-corner parsing model, and two classes of stochastic context free graph grammars. We remark that there is little or no literature on EM learning of these probabilistic models.

In what follows, after reviewing PRISM briefly by a simple example, we look at each of the above mentioned EM learning. The reader is assumed to be familiar with the EM algorithm (McLachlan & Krishnan, 1997), logic programming (Doets, 1994) and stochastic natural language processing (Manning & Schütze, 1999).

2. Symbolic-statistical modeling

By symbolic-statistical modeling we mean modeling of a distribution over structured objects represented by symbols. Typical symbolic-statistical models are discrete BNs, HMMs and PCFGs where BNs define distributions over symbols, HMMs over strings and PCFGs over parse trees. They have been developed as different formalisms and people have to write code for different EM algorithms. PRISM breaks this order-made approach (Sato & Kameya, 1997; Sato & Kameya, 2001). It offers a generic language for describing symbolic-statistical models together with a unified EM learning routine. As a programming language it is an augmentation of Prolog by probabilistic built-ins for calculating and learning probabilities. So the user writes just Prolog programs using some probabilistic predicates. However the semantics, *distribution semantics* (Sato, 1995), is significantly extended compared to the standard least model semantics for logic programs in which a program denotes a (σ additive) probability measure over the set of possible Herbrand interpretations. The current PRISM[4] is implemented on top of B-Prolog using its tabling mechanism (Zhou et al., 2004). It can also deal with negation (failure), though we do not discuss this new feature in this paper.

We here explain a PRISM program for the sake of self-containedness. Figure 1 is a simple PRISM program modeling the inheritance of ABO blood types. `target(btype,1)` says we observe an event represented by an atom `btype(·)`. `values(abo,[a,b,o])` introduces a discrete random variable `abo` (i.i.d.s) whose range is {`a`, `b`, `o`} and the corresponding probabilities are set by `set_sw/2` to 0.3, 0.3 and 0.4 respectively on loading the program.

```
target(btype,1).
values(abo,[a,b,o]).
:- set_sw(abo,0.3+0.3+0.4).

btype(X):- pg_table(X,[Gf,Gm]),gtype(Gf,Gm).
pg_table(X,GT):-
  ((X=a;X=b),(GT=[X,o];GT=[o,X]; GT=[X,X])
  ; X=o, GT=[o,o]
  ; X=ab, (GT=[a,b];GT=[b,a]) ).
gtype(Gf,Gm):- msw(abo,Gf),msw(abo,Gm).
```

Figure 1. ABO-blood type program

The following clauses can be read just like an ordinary Prolog program. The only difference from Prolog is the use of `msw/2` predicate in the definition of `gtype` clause. In general when a random variable named 'n' is introduced by `values(`n`,[`$v_1,\ldots,v_k$`])` and its distribution is specified by `set_sw/2` as `set_sw(n,[`p_1`+`$\cdots$`+`p_k`])` ($\sum_i p_i = 1$), `msw(`n`,Y)` probabilistically returns in `Y` one of $\{v_1,\ldots,v_k\}$ with $P(n = v_i) = p_i$. These probabilities are called *parameters*. We assume different occurrences of the same random variable name denote i.i.d.s.

The purpose of this program is to estimate parameters for the distribution of `abo` genes from observations given as a list like `[btype(a),btype(ab),`$\cdots$`]`. In view of statistical modeling, `btype(·)` is an incomplete data with `abo` as a hidden variable. So the EM algorithm applies to this parameter estimation problem. The unique feature of our approach is the existence of a search phase that logically reduces an observed atom to a disjunction of explanations[5] represented as an AND-OR graph called *explanation graph* to which a common EM algorithm, the graphical EM algorithm, is applied. The formula below shows the explanation graph for `btype(a)`.

```
btype(a) <=>
    gtype(a,o) v gtype(o,a) v gtype(a,a)
gtype(a,o) <=> msw(abo,o) & msw(abo,a)
gtype(o,a) <=> msw(abo,a) & msw(abo,o)
gtype(a,a) <=> msw(abo,a) & msw(abo,a)
```

As the search phase is carried out using tabling (memoizing), the generated explanation graph generally

[3] For the record in the case of PCFGs, EM learning for a PCFG of moderate size (860 production rules) using a real corpus, ATR corpus(Uratani et al., 1994), containing 11000 sentences can now be completed in less than 10 minutes on a PC with 2MHz CPU.

[4] URL = `http://sato-www.cs.titech.ac.jp/prism/`

[5] An explanation is a conjunction of `msw` atoms.

shares subgraphs hierarchically, and this hierarchical structure sharing makes it possible to compute probabilities in a dynamic programming manner.

3. EM Learning for Probabilistic Left Corner Grammars

3.1. Probabilistic LC grammars

In this section, we deal with EM learning for *PLCGs (probabilistic left-corner grammars)*. PLCGs construct parse trees in a bottom-up manner using CFG rules but with different parameterization from PCFGs. In a PCFG probabilities $P(A \to \alpha \mid A)$ in top-down rule selection of $A \to \alpha$ given a non-terminal A are taken as parameters while in a PLCG projection probabilities $P(A \to B\beta \mid B, G)$ given a category B of a completed subtree and a goal category G, for instance are included in the parameter set for a PLCG. As rule selection is affected by the completed tree B, PLCGs are expected to be more context sensitive than PCFGs.

Although PCFGs and PLCGs use the same set of CFG rules, implementing EM learning for PLCGs seems much harder than PCFGs. In fact although there is some literature on PLCGs (Manning, 1997; Roark & Johnson, 1999), parameters there are obtained by counting using a tree bank, not by EM learning. The only literature we found that uses EM learning is (Van Uytsel et al., 2001) which describes a specialized EM algorithm derived for lexicalized PLCGs.

By contrast we do not need such a derivation. All one has to do is to write an efficient program while observing the *principle of generative modeling* which states that a model should describe a sequential process of generating an observable output and once a probabilistic choice is made, there must be no failure in the subsequent process (*failure-free condition*). This principle looks rigid but ensures the mathematical correctness of our statistical model.[6]

3.2. Three operations for PLCG parsing

We encode a PLC grammar as a PRISM program in Figure 2 which is a probabilistic extension of a non-probabilistic left-corner parser described in (Manning, 1997). The behavior of this program as a parser exactly follows the non-probabilistic LC parser except the use of `msw/2`. It parses a sentence of length N in $O(N^3)$ time thanks to the tabling (caching) mechanism of PRISM.

[6]Recently we succeeded in relaxing the failure-free condition to explore a wider class of distributions(Sato & Kameya, 2004).

So let us check how it works as a sentence generator. The top-goal for this purpose is `:-plc(Ws)` which, after invoking `g_call/3` and `lc_call/4` recursively, returns a list `Ws` = $[w_1, \ldots, w_k]$ of terminals as a grammatical sentence.

```
plc(Ws):-
   start_symbol(C), g_call([C],Ws,[]).

g_call([],L,L).     % shift
g_call([G|R],[Wd|L],L2):-
   ( terminal(G), G=Wd, L1=L
   ; \+ terminal(G),
       msw(first(G),Wd), lc_call(G,Wd,L,L1) ),
   g_call(R,L1,L2).

lc_call(G,B,L,L2):- % project or attach
   msw(lc(G,B),rule(A,[B|RHS2])),
   g_call(RHS2,L,L1),
   ( G == A, attach_or_project(A,Op),
       ( Op == attach, L2=L1
       ; Op == project, lc_call(G,A,L1,L2) )
   ; G \== A, lc_call(G,A,L1,L2) ).

attach_or_project(A,Op):-
   ( values(lc(A,A),_), msw(attach(A),Op)
   ; \+ values(lc(A,A),_), Op=attach ).
```

Figure 2. A probabilistic LC parser

`g_call(Gs,L1,L2)` says a difference list `L1-L2` is derived from a list `Gs` of terminals and nonterminals. The shift operation in LC parsing is carried out by the second `g_call` clause in such a way that in the generation process, if `G` is not a terminal, `Wd` is probabilistically chosen by `msw(first(G),Wd)` from the first set of `G`. Once `Wd` is chosen, `L-L1`, the rest of sentence, is generated by calling `lc_call(G,Wd,L,L1)`.

`lc_call(G,B,L,L2)` means there is a completed subtree whose category is `B`, the goal category `G` and `B` stand in the left-corner relation[7], and `G` derives `[B|L]-L2` (see Figure 3). Given `G` and `B` in the generation process, a rule `A` $\to$ `[B|RHS2]` is probabilistically chosen by `msw(lc(G,B),·)` depending on the pair `(G,B)`. Then `g_call(RHS2,L,L1)` is called to complete the derivation of `L-L1` from `RHS2`. We have two cases then. If `G` = `A` and both project operation and attach

[7]If there is a chain of production rules $G \to A_1\alpha_1, A_1 \to A_2\alpha_2, \ldots, A_n \to B\alpha_n$ $(n \geq 1)$ G and B are said to be in the left-corner relation.

operation are possible, we probabilistically choose one of them by `msw(attach(A),·)`. Otherwise only attach operation is possible. Else `G` $\neq$ `A` and only project operation is possible. So we call `lc_call(G,A,L1,L2)`.

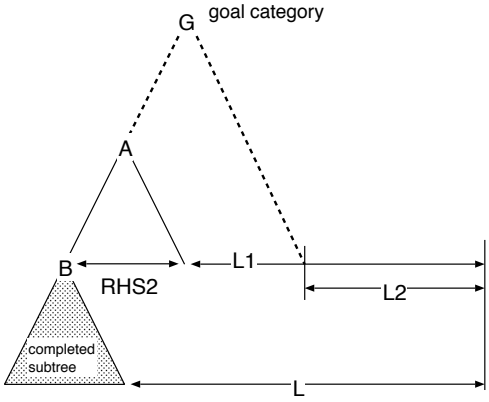


Figure 3. `lc_call(G,B,L,L2)` in LC parsing

By inspection, we know that the generation process started by `:-plc(Ws)` never fails and probabilistic choices are exclusively made by `msw/2`. Hence we may conclude according to (Sato & Kameya, 2001) that EM learning performed by the program is mathematically correct[8].

3.3. EM learning with ATR corpus

We conducted an experiment of EM learning with a real corpus, ATR corpus which is a Japanese corpus containing more than 11000 parses (Uratani et al., 1994). The backbone CFG grammar consists of 860 rules. Prior to the experiment, we specialized (unfolded) the program in Figure 2 by CFG rules for speed up. The specialization yielded about 2800 clauses such as

```
g_call(adv,[Wd|L],L2):-
    msw(first(adv),Wd),lc_call(Wd,adv,L,L2).
```

We also obtained 21000 `values` declarations for `msw` atoms. In the experiment, a PC having 3.4Ghz CPU and 2GB memory was used with a PRISM program containing these clauses and declarations. The parsing phase finished in 64 seconds and in the subsequent EM learning phase, the gEM algorithm converged after 367 iterations[9], taking 66 minutes as total learning time.

[8]Actually we need another condition "if there is no loss of probability mass to infinite generation process," which we assume to hold for simplicity here.

[9]The convergence is judged if an increase in the log-likelihood is less than 10^{-4}.

It is hard to say whether our approach is unreasonably slow or not, because (Van Uytsel et al., 2001) which is the only literature available to us on similar EM learning, does not mention the total learning time for the EM learning of a PLCG and hence comparison is difficult to make. Also we expect a new implementation of PRISM underway will speed up learning further. Putting learning time aside however, one thing seems clear: this experiment shows that our generic approach works even in the relatively unexplored field, i.e. EM learning of PLCGs, straightforwardly without much difficulty.

4. EM Learning for Probabilistic Context Free Graph Grammars

In this section, we tackle the problem of parameter learning of *PCFGGs (probabilistic context free graph grammars)*, an important yet almost unknown area. *CFGGs (context free graph grammars)* are a natural generalization of CFGs. They define a set of graphs by repeatedly replacing subgraphs using production rules whose right-hand side is a graph, not a string. PCFGGs are CFGGs with probabilities associated with production rules just like PCFGs. They define distributions over the set of producible graphs. We here focus on two classes of CFGGs and their probabilistic versions. One is *HRGs (hyper edge replacement grammars)* which replace edges and the other is *NLCGs (node label controlled grammars)* which replace nodes (Rozenberg, 1997).

As graphs are much richer structure than strings and abundantly used to describe control diagrams, chemical compounds, gene networks, WWW and so on, developing a method of parameter learning for PCFGGs would contribute greatly to the statistical analysis of complex structure represented by graphs. However in reality there is almost no literature on parameter learning of PCFGGs except (Oates et al., 2003) in which the authors state that "To the best of our knowledge, our paper is the first to present a formally sound algorithm for computing maximum likelihood parameter estimates for a large class of HR grammars."

In the following we show that PRISM enables us to learn parameters not only for *PHRGs (probabilistic HR grammars)* like (Oates et al., 2003) but for an untried class, *PNLCGs (probabilistic NLC grammars)*[10] as well.

[10]There is no literature on EM learning of PNLCGs as far as we know.

4.1. Probabilistic HR grammars

In this subsection, we attempt EM learning of PHRGs using PRISM.

4.1.1. Hyper graphs

We first introduce terminology, mostly following (Rozenberg, 1997). A *hyper graph* $\mathcal{H}$ is a triple (V, E, X) where V is a finite set of nodes, E a finite set of *hyper edges* and X a sequence of nodes in V called *external nodes*. The type of $\mathcal{H}$ is defined to be the length of X and denoted as type($\mathcal{H}$).

A *hyper edge* is an ordered pair (Cat, AN) such that Cat is a label of the hyper edge and AN is a sequence of pair-wise distinct (this is our simplifying assumption) nodes in V called *attachment nodes*. The label can be null. The length of AN is denoted by type(Cat). In Figure 4, the left graph represents a hyper edge $(S, (2, 5, 4))$ whereas the right graph represents $(S, (1, 2))$ respectively. In drawing graphs, a label is boxed and an underlined number $\underline{i}$ marks an edge connected to the i-th attachment node in AN. When the length of AN is 2, we omit underlined numbers and instead use an arrow assuming the source node of the arrow is the first attachment node and the sink node is the second one.

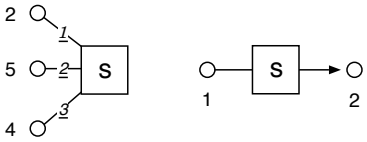


Figure 4. Hyper edges

An *HRG (hyper edge replacement grammar)* is a quadruple (N, T, P, S) where N, a set of labels called *nonterminals* and T, a set of labels called *terminals* are disjoint. P is a set of *rules* and S a *start symbol*.

A *rule* is an ordered pair (Cat, HG) where Cat is a nonterminal which is a label of a hyper edge HE whereas HG is a hyper graph such that type(Cat) = type(HG). So the attachment nodes AN in HE and the external nodes X' in HG have the same length. We understand that the i-th attachment node of AN corresponds to the i-th external node of X'. Using this correspondence, we replace (an isomorphic copy of) the hyper edge HE in a host graph $\mathcal{H}$ by HG as follows. First after matching HE against some hyper edge in $\mathcal{H}$, we remove the matched edge and add HG. Second we glue each external node of X' of HG to the corresponding attachment node of AN in $\mathcal{H}$ (*hyper edge replacement*). A *start graph* is a hyper graph such that nodes are all external nodes and it contains a single hyper edge whose label is S, the start symbol.

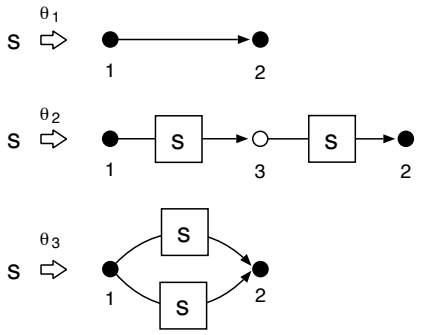


Figure 5. PHRG $\mathcal{G}_{hr}$

A *PHRG (probabilistic hyper edge replacement grammar)* is an HRG such that rules for the same label are assigned probabilities (*parameters*) which sum to unity. Just like PCFGs, we start from the start graph and repeat one step derivation, i.e. hyper edge replacement using a probabilistically chosen rule until a *terminal graph*, i.e. a hyper graph TG in which all labels are terminals, is generated. TG is a member of the set of graphs specified by the PHRG, and its probability is computed as the product of probabilities associated with rules used in its derivation. This way a PHRG defines a distribution over the set of terminal graphs. A PHRG in Figure 5 is based on an HRG borrowed from (Rozenberg, 1997). Each rule has a parameter θ_i $(i = 1, 2, 3)$. There black circles are external nodes. Figure 6 below illustrates a derivation process from the start graph in the upper-left corner.

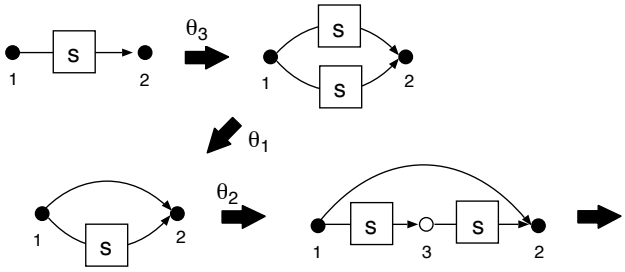


Figure 6. A derivation sequence using $\mathcal{G}_{hr}$

4.1.2. EM learning of PHRGs

We conducted EM learning of $\mathcal{G}_{hr}$. The first step is to write a PRISM program for $\mathcal{G}_{hr}$ that causes no failure, following the principle of generative modeling (Sato & Kameya, 2001). So we wrote a program DB_{hr} in Figure 7 that generatively defines a distribution over terminal graphs specified by $\mathcal{G}_{hr}$. We encoded a hyper graph as a list (set) of hyper edges represented

by atoms of the form `edge(Cat,AN)`. The three rules in Figure 5 are straightforwardly encoded as a value declaration below. For example, the second line of the declaration encodes the second rule of $\mathcal{G}_{hr}$.

```
values(s,[
 [[1,2],[], [edge([],[1,2])]],
 [[1,2],[3],[edge(s,[1,3]),edge(s,[3,2])]],
 [[1,2],[], [edge(s,[1,2]),edge(s,[1,2])]] ]).

phrg(edge(Cat,AN),L):-
  get_max_node(AN,Max),
  phrg(edge(Cat,AN),Max,L,[],_).
phrg(edge(Cat,AN),Max,L1,L3,Max3):-
  ( values(Cat,_),    % replacement possible
      msw(Cat,RHS),   % choose a rule
      glue_rhs(RHS,AN,HG),
      phrg2(HG,Max2,L1,L3,Max3)
  ; \+ values(Cat,_), % no replacement
     L1 = [edge(Cat,AN)|L3],
     Max3 is Max ).
phrg2([edge(Cat,AN)|X],Max,L1,L3,Max3):-
  phrg(edge(Cat,AN),Max,L1,L2,Max2),
  phrg2(X,Max2,L2,L3,Max3).
phrg2([],Max,L1,L1,Max).
```

Figure 7. PRISM program DB_{hr} for $\mathcal{G}_{hr}$ (part)

This program works similarly to a PRISM program for a PCFG. `:-phrg(edge(s,[1,2]),L)` for example, probabilistically generates a terminal graph from `edge(s,[1,2])`, and returns it in `L`. The primary computation is done by `phrg(edge(Cat,AN),Max,L1,L3,Max3)`. When called with a ground `edge(Cat,AN)`, `L1` which stores the current hyper graph and `Max`, the current maximum number of nodes, it adds to `L1` a new terminal graph derived from `edge(Cat,AN)` and returns the enlarged terminal graph as a difference list `L1-L3` together with `Max3`, the renewed maximum node number. `glue_rhs(RHS,AN,HG)` carries out hyper edge replacement using the right-hand side of a rule and `AN`, the list of attachment nodes in the graph being constructed. It returns a terminal graph `HG` to be added to `L1`.

Using DB_{hr}, we randomly generated a sample of size 100 using parameters set to $(\theta_1,\theta_2,\theta_3) = (0.6, 0.2, 0.2)$. To cut down on processing time, the size of generated graphs was restricted to 15. To perform parameter estimation, we used a very naive parser written in PRISM which resembles DB_{hr}, and extracted explanation graphs from the sampled graphs, on which the gEM algorithm was run to estimate parameters[11]. We repeated this experiment 20 times and took averages of estimated parameters. The table in Figure 8 summarizes results where σ is a standard deviation. Looking at these figures, it might be safely said that parameters are reasonably estimated though how the size restriction on generated graphs affected estimation is unclear.

	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$
Original	0.6	0.2	0.2
Ave.	0.65082	0.18220	0.16697
σ	0.01949	0.01672	0.01692

Figure 8. Learned parameters for $\mathcal{G}_{hr}$

We conducted EM learning of PHRGs for other probabilistic models as well. They include an HMM, a PCFG, a probabilistic context sensitive grammar and so on, and in all cases we could successfully estimate parameters.

4.2. Probabilistic NLC grammars

Finally we attempt EM learning of probabilistic *NLC (node label controlled) grammars*. Due to the lack of space, we have to skip details and programs.

Suppose we have NT, a set of nonterminal node labels, and T, a set of terminal node labels. NT and T are disjoint. As convention we use upper case letters for labels in NT and lower case letters for those in T. Next an *NLC rule* is a replacement rule $X \to R : C$ where X is a nonterminal node label, R is an undirected graph called *replacing graph* such that nodes are labeled by $NT \cup T$. In the following, we add a simplifying assumption that *labels in R are distinct from one another*. So R can be represented by a set of pairs of labels. C denotes a set of connection instructions (α, β) where α and β are from $NT \cup T$. Let us take an example to get the idea of node replacement using C.

$$A \to \{(a, A)\} : \{(a, a), (a, A)\}$$

This rule replaces a node n labeled A in a host graph H with a new graph in three steps. First we remove n and every edge incidental to n from H. We use H^- to refer to the remaining graph. Second we add (a copy of) the replacing graph $R = \{(a, A)\}$ to H^-. Here (a, A) is an edge connecting a node labeled a and

[11] We did not attempt to develop a sophisticated parser, as our primary purpose is not efficient parsing but the verification of the possibility of EM learning.

a node labeled A. Finally according to a connection instruction (a, a) in the right component of the rule, we connect every node labeled a in H^- to the node labeled a in the added R. Likewise we process (a, A) by connecting every node labeled a in H^- to every node labeled A in R.

An *NLC grammar* is a tuple (NT, T, P, S) where NT is a set of nonterminal node labels, T a set of terminal node labels, P a set of NLC rules and S a start graph (we assume a single symbol here). By associating probabilities with rules as usual, we can define a *PNLCG (probabilistic NLC grammar)*. Figure 9 is an example of probabilistic NLC grammar $\mathcal{G}_{nlc}$.

$$\left\{\begin{array}{lcl:l:l} A & \to & \{(a,A)\} & \{(a,a),(a,A)\} & \theta_1 \\ A & \to & \{(a,B)\} & \{(a,a),(a,B)\} & \theta_2 \\ A & \to & \{a\} & \{(a,a)\} & \theta_3 \\ B & \to & \{(a,A)\} & \{(a,a),(a,A)\} & \theta_4 \\ B & \to & \{(a,B)\} & \{(a,a),(a,B)\} & \theta_5 \\ B & \to & \{a\} & \{(a,a)\} & \theta_6 \end{array}\right.$$

Figure 9. $\mathcal{G}_{nlc}$

Figure 10 depicts a derivation of a terminal graph which has no nonterminals. Nonterminal labels are boxed while terminal labels are put in a circle. In this case the probability of the derived graph is calculated as $\theta_1\theta_2\theta_4\theta_3$.

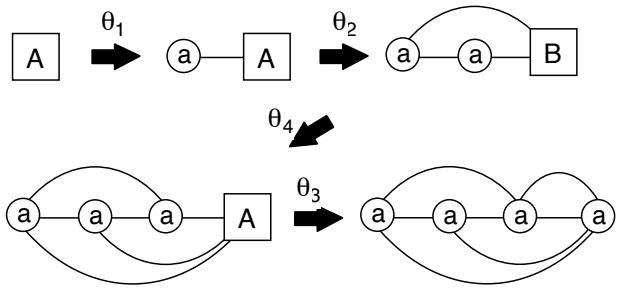


Figure 10. A derivation sequence by $\mathcal{G}_{nlc}$

We conducted EM learninng of $\mathcal{G}_{nlc}$. We first wrote a PRISM program DB_{nlc} not shown here which faithfully simulates a derivation process of $\mathcal{G}_{nlc}$. After setting parameters to $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = (0.2, 0.2, 0.6, 0.6, 0.2, 0.2)$, we randomly generated 1000 samples by DB_{nlc} with A being a start symbol. We also prepared a simple (naive) parser written in PRISM by slightly modifying DB_{nlc} and let it run on the sampled graphs to learn parameters by the gEM algorithm. the gEM algorithm stopped after 63 iterations (threshold is 10^{-4}). The estimated parameters are shown in Table 11. They appear to be close to the original values[12].

	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$
Original	0.2	0.2	0.6
Estimate	0.22134	0.21907	0.5595
	$\hat{\theta}_4$	$\hat{\theta}_5$	$\hat{\theta}_6$
Original	0.6	0.2	0.2
Estimate	0.60468	0.15049	0.24481

Figure 11. Learned parameters for $\mathcal{G}_{hr}$

5. Discussion and related work

We have proposed a generic approach to EM learning for symbolic-statistical models based on a logic-based probabilistic language PRISM (Sato & Kameya, 1997). It saves us the trouble of deriving a specialized EM algorithm for each application. We have only to write, obeying the principle of generative modeling (Sato & Kameya, 2001), a PRISM program tuned for each probabilistic model to define a desired distribution. Then parameters are automatically estimated from data by a built-in EM algorithm.

We have demonstrated effectiveness of our approach by tackling problems in relatively unknown areas, i.e. EM learning for a probabilistic LC parser, one for a probabilistic HR graph grammar and one for a probabilistic NLC graph grammar, where very few or no literature is available. We developed appropriate PRISM programs and EM learning was successfully done in every case using them, which seems to imply that PRISM is an appropriate tool for rapid prototyping of new probabilistic models.

Recently there are many proposals for probabilistic language for probabilistic models. IBAL (Pfeffer, 2001) for example is a probabilistic functional language which can define distributions and perform parameter estimation. It also can deal with decision theoretic programs. Dyna (Eisner et al., 2004) is a programming language designed mainly for statistical natural language processing. It is based on equations and dynamic programming. For other proposals related to statistical relational learning, (De Raedt & Kersting, 2003) provides a comprehensive survey.

References

De Raedt, L., & Kersting, K. (2003). Probabilistic logic learning. *ACM-SIGKDD Explorations, special*

[12] We could not repeat this experiment due to a shortage of time.

issue on Multi-Relational Data Mining, 5, 31–48.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Royal Statistical Society, B39*, 1–38.

Doets, K. (1994). *From logic to logic programming.* The MIT Press.

Eisner, J., Goldlust, E., & Smith, N. (2004). Dyna: A declarative language for implementing dynamic programs. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL'04), Companion Volume* (pp. 218–221).

Elston, R., & Stewart, J. (1971). A general model for the genetic analysis of pedigree data. *Human Heredity, 21*, 523–542.

Inui, K., Sornlertlamvanich, V., Tanaka, H., & Tokunaga, T. (1997). *A new probabilistic LR language model for statistical parsing*Technical Report TR97-0004 Dept. of CS). Tokyo Institute of Technology.

Kameya, Y., & Sato, T. (2000). Efficient EM learning for parameterized logic programs. *Proceedings of the 1st Conference on Computational Logic (CL2000)* (pp. 269–294). Springer.

Kruglyak, L., Daly, M., Reeve-Daly, M., & Lander, E. (1996). Parametric and nonparametric linkage analysis: A unified multipoint approach. *American Journal of Human Genetics, 58*, 1347–1363.

Manning, C. (1997). Probabilistic parsing using left corner language models. *Proceedings of the Fifth International Conference on Parsing Technologies (IWPT-97)* (pp. 147–158). MIT Press.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing.* The MIT Press.

McLachlan, G. J., & Krishnan, T. (1997). *The EM algorithm and extensions.* Wiley Interscience.

Oates, T., Doshi, S., & Huang, F. (2003). Estimating maximum likelihood parameters for stochastic context-free graph grammars. *Proceedigs of the 13th International Conference on Inductive Logic Programming (ILP 2003)* (pp. 281–298).

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems.* Morgan Kaufmann.

Pfeffer, A. (2001). IBAL: A probabilistic rational programming language. *Proceedings of the 17th International Conference on Artificial Intelligence (IJCAI'01)* (pp. 733–740).

Rabiner, L. R., & Juang, B. (1993). *Foundations of speech recognition.* Prentice-Hall.

Roark, B., & Johnson, M. (1999). Efficient probabilistic top-down and left-corner parsing. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (pp. 421–428).

Rozenberg, G. (Ed.). (1997). *Handbook of graph grammars and computing by graph transformations, volume 1: Foundations.* World Scientific.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)* (pp. 715–729).

Sato, T., & Kameya, Y. (1997). PRISM: a language for symbolic-statistical modeling. *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)* (pp. 1330–1335).

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research, 15*, 391–454.

Sato, T., & Kameya, Y. (2004). A dynamic programming approach to parameter learning of generative models with failure. *Proceedings of ICML 2004 workshop on Learning Statistical Models from Relational Data (SRL2004).*

Uratani, N., Takezawa, T., Matsuo, H., & Morita, C. (1994). *ATR integrated speech and language database*Technical Report TR-IT-0056). ATR Interpreting Telecommunications Research Laboratories. In Japanese.

Van Uytsel, D., Van Compernolle, D., & Wambacq, P. (2001). Maximum-likelihood training of the PLCG-based language model. *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop 2001 (ASRU'01).*

Zhou, N.-F., Shen, Y., & Sato, T. (2004). Semi-naive Evaluation in Linear Tabling . *Proceedings of the Sixth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP2004)* (pp. 90–97).

Part II

Genic Interaction Extraction Challenge Papers

Learning Language in Logic - Genic Interaction Extraction Challenge

C. Nédellec CLAIRE.NEDELLEC@JOUY.INRA.FR
Laboratoire Mathématique, Informatique et Génome (MIG), INRA,
Domaine de Vilvert, 78352 F- Jouy-en-Josas cedex.

Abstract

We describe here the context of the LLL challenge of Genic Interaction extraction, the background of its organization and the data sets. We discuss then the results of the participating systems.

1. Introduction

The Learning Language in Logic **(**LLL05) challenge is part of the 2005 LLL workshop. The LLL05 challenge task is to learn rules to extract protein/gene interactions in the form of relations from biology abstracts from the Medline bibliography database. The goal of the challenge is to test the ability of the participating ML systems to learn rules for identifying the gene/proteins that interact and their roles, agent or target. The training data contains the following information:

- The Agent and Target of the genic interactions.
- A dictionary of named entities (including typographic variants and synonyms)
- Linguistic information: word segmentation, lemmatization and syntactic dependencies.

The participants have tested their Information Extraction (IE) rules on a separate test set in a limited amount of time. The challenge organizers have provided the facilities for computing the scores of the results. Six different teams have participated and reported their results in the papers in this volume. This paper aims at summarizing the motivation for the challenge, the presentation of the training and test data and comparing the participant results.

2. Motivation

2.1 Biological motivation

Developments in biology and biomedicine are reported in large bibliographical databases either focused on a specific species (*e.g.* Flybase, specialized on *Drosophila Melanogaster*) or not (*e.g.* Medline). These types of information sources are crucial for biologists, but there is a lack of tools to explore them and extract relevant information.

While recent named entity recognition tools have gained a certain success on these domains, event-based Information Extraction (IE) is still challenging. Biologists can search bibliographic databases via the Internet, using keyword queries that retrieve a large set of relevant papers. To extract the requisite knowledge from the retrieved papers, they must identify the relevant paragraphs or sentences. Such manual processing is time consuming and repetitive, because of the bibliography size, the relevant data sparseness, and because the database is continually updated. For example, from the Medline database, the focused query "*Bacillus subtilis* and transcription", which returned 2,209 abstracts in 2002 retrieves more than 2,693 today. We chose this example because *Bacillus subtilis* is a model bacterium and because transcription is both a central phenomenon in functional genomics involved in gene interaction and a popular IE problem.

Example:
GerE *stimulates* ***cotD*** *transcription and inhibits* ***cotA*** *transcription in vitro by* ***sigma K*** *RNA polymerase, as expected from in vivo studies, and, unexpectedly, profoundly inhibits in vitro transcription of the gene (****sigK****) that encode* ***sigma K****.*

In this example, there are 6 genes and proteins mentioned and among the 30 potential ordered couples, 5 couples actually interact: (GerE, cotD), (GerE, cotA), (sigma K, cotA), (GerE, SigK) and (sigK, sigma K). The precision of the baseline method that extracts gene/protein cocitations is then 20 % for 100 % recall. In gene interactions, the agent is distinguished from the target of the interaction. Such interactions are central in functional genomics because they form regulation networks that are very useful for determining the function of the genes. The description of such gene interactions is not available in structured databases but only in scientific papers. Figure 1 gives an example of such a regulation network.

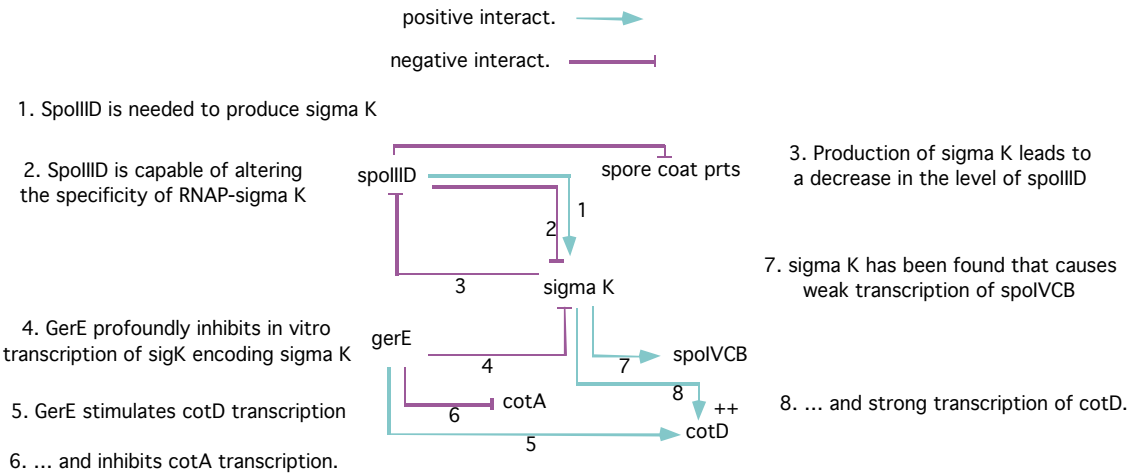


Figure 1. Example of a regulation network

The arrows in Figure 1. represent the interactions between proteins and genes of *Bacillus subtilis* involved into the sporulation process. The numbered textual annotations around represent the fragments of MedLine abstracts the interactions have been extracted from.

2.2 Learning Language in Logic motivation

Applying IE to genomics and more generally to biology is not an easy task because IE systems require deep analysis methods to extract the relevant pieces of information. As shown in the example, retrieving that GerE is the agent of the inhibition of the transcription of the gene sigK requires at least coordination processing and syntactic dependency analysis *(e.g.* GerE is the subject of inhibits and cotA transcription is the object of inhibits). Such a relational representation of the text motivates relational learning to be applied to automatically acquire the information extraction rules.

For instance:

genic_interaction(X,Z):-
is-a(**X**,protein), **subject(X,Y)**, verb(**Y**), is-a(**Y**,interaction_action), **Obj**(**Z**,**Y**), is-a(**Z**,gene-expression).

Interpretation of the rule

If the **subject X** of an interaction action **verb Y**, is a protein name, and the **object Z** is a gene name or a gene expression, **then**, X is the *agent* and Z is the *target* of the interaction.

2.3 Expected impact on Machine Learning research and field of interest

Information Extraction has been a ML application area since the beginning of the nineties. However, most of the work focuses on the named-entity recognition problem with mainly statistics-based methods applied on shallow text representations. There were few attempts to develop ML methods for extracting relations from text although the development of relational methods and inductive learning yield excellent results in other application areas. The main reason for the lack of relational learning development in IE is due to the lack of dataset in IE that ML researchers could use without any investment in natural language processing (NLP). Indeed, relational event extraction requires that the text is deeply processed by syntactic parsing including syntactic dependencies. Most of the ML research groups do not have the NLP competencies and tools for performing this processing in specific domains with a good quality level. As a consequence, the training data set has been prepared so that ML researchers only could perform basic format change to be able to apply their methods.

The LLL challenge data set meets this requirement. Its use does not need any investment in biology neither in NLP. All the needed information is provided at a good quality level. The syntactic dependencies, which are critical here, have been automatically produced by LinkParser (Sleator and Temperley, 1993) and manually crosschecked by specialists of syntactic analysis of MIG and LIPN laboratories.

The expected impact on ML is a growing interest for IE and more generally for semantic knowledge learning from textual data. It is a great opportunity for ILP to evaluate, compare, adapt and develop methods on a large application domain that is critical from both a research and economic point of view. For instance, automatically producing meta data for the semantic Web from textual Web pages is strongly related to this ML and IE domain.

Moreover, the biologist expectations are very high and the particular task proposed here is not artificial but is critical in functional genomics. Even a partial automatization of the information extraction would be a considerable progress. We also expect a high impact of the availability of this data on the development of ML in bioinformatics for the access to textual content.

3. Description of the data

The challenge focuses on information extraction of gene interactions in *Bacillus subtilis*. Extracting gene interaction is the most popular *event* extraction task in biology. *Bacillus subtilis* (*Bs*) is a model bacterium and many papers have been published on direct gene interactions involved in sporulation, as opposed to what happens for eukaryotes. The gene interactions are generally mentioned in the abstract and the full text of the paper is not needed here. The relevant abstracts have been selected by querying MedLine on Bacillus subtilis transcription and sporulation. The relevant information is mostly local to single sentences (Ding *et al.*, 2002). The main exception comes from coreferences. For instance, the gene/protein name is mentioned in a sentence and referred to in the form of a pronoun or an hyperonym in the next sentence. We do not consider this case here. The abstracts have been segmented into sentences. Sentences have been automatically filtered by the STFilter system in order to retain those that contain at least two gene/protein names and are most probable to denote interactions (Nedellec *et al.*, 2000). MIG-INRA expert biologists have annotated with the XML editor CADIXE[1] hundreds of the interactions and the experimental conditions. For this challenge, a simple subset of them is provided as training and test data. The protein/gene names that can play the roles of agent and target of the gene interaction in the data sets are also recorded in a named-entity dictionary in the form of lists of canonical forms and variants. There could be more than one interaction per sentence and a given protein / gene may be involved in several interactions in different roles, agent or target.

3.1 Biological typology

The data has been selected on the following basis, the gene interaction is expressed,

- By an **explicit action** such as, ***GerE*** *stimulates* ***cotD*** *transcription*
- Or by a **binding of the protein** on the promoter of the target gene, *Therefore,* ***ftsY*** *is solely expressed during sporulation from a* ***sigma(K)****- and* ***GerE****-controlled promoter that is located immediately upstream of* ***ftsY*** *inside the smc gene*.
- Or by **membership to a regulon** family, ***yvyD*** *gene product, being a member of the* ***sigmaB*** *regulon* [..]

The sentences relying on other biological models have not been considered. For instance, a very frequent case involves gene mutants where the role of the genes in the interactions can be derived from the comparison with the normal experimental conditions. Other biological models are less represented. Then, the three selected categories are well representative of the interaction distribution excluding the mutant category.

3.2 Linguistic typology

The data set is decomposed into two subsets of increasing difficulties. The first subset does not include coreferences neither ellipsis, as opposed to the second subset. The coreferences selected are kept very simple. Most of them are just appositions.

For example,

Transcription of the ***cotD*** *gene is activated by a protein called* ***GerE****, [..]*

GerE *binds to a site on one of this promoter,* ***cotX*** *[..]*

Notice that when the absence of interaction between two genes is explicitly stated, it is represented as interaction information.

For example,

There likely exists another ***comK****-independent mechanism of* ***hag*** *transcription.*

3.3 Linguistic information

These two subsets are available with two kinds of linguistic information,

1. **The Basic data set** includes sentences, word segmentation and biological target information: agents, targets and genic interactions
2. **The Enriched data set** includes also lemmas and syntactic dependencies manually checked.

The corpora and the information extraction task are the same in both cases. The two sets differ only by the nature of the linguistic information available. The participants to the challenge were free to use or not this linguistic information or to apply their own linguistic tools. When publishing their results, the participants had to be clear about the kind of information that has been used for training the learning methods.

3.4 Data representation

The data representation is detailed on the Web site: http://genome.jouy.inra.fr/texte/LLLchallenge/ The training data includes the target information to be extracted, the agent and target of the interaction.

[1] It has been developed by the National inter-EPST Caderige project and mainly involves LEIBNIZ-IMAG, MIG-INRA, LIPN-CNRS and ENSAR-INRA. It is available on demand.

Example from the Basic data set:

```
ID      11011148-1

sentence      ykuD was transcribed by SigK RNA polymerase from T4 of sporulation.

words  word(0,'ykuD',0,3)
       word(1,'was',5,7)
       word(2,'transcribed',9,19)
       word(3,'by',21,22)
       word(4,'SigK',24,27)
       word(5,'RNA',29,31)
       word(6,'polymerase',33,42)
       word(7,'from',44,47)
       word(8,'T4',49,50)
       word(9,'of',52,53)
       word(10,'sporulation',55,65)

agents        agent(4)

targets       target(0)

genic_interactions
       genic_interaction(4,0)
```

There is one genic interaction involving one agent and target here. The arguments of the agent, target and genic-interaction literals refer to the unique identifier of the word.

Example from the enriched data set:

```
ID      10747015-5

sentence      Localization of SpoIIE was shown to be dependent on the essential cell division protein FtsZ.

words  word(0,'Localization',0,11)
       word(1,'of',13,14)
word(2,'SpoIIE',16,21)

lemmas lemma(0,'localization')
       lemma(1,'of')lemma(2,'SpoIIE')

syntactic_relations
       relation('comp_of:N-N',0,2)
       relation('mod_att:NADJ',13,10)
       relation('mod_pred:N-ADJ',0,7)
       relation('mod_att:N-N',14,13)

agents        agent(14)

targets       target(2)

genic_interactions
       genic_interaction(14,2)
```

The lemma of named-entities is the canonical form as defined in the associated named-entity dictionary. For instance, the canonical form of kinD is ykvD according to the dictionary. The syntactic relations are defined in the Syntactic Analysis Guidelines document. For instance, `relation('comp_of:N-N',0,2)` means that word 0 and 2, namely, 'Localization' and 'SpoIIE' are two nouns and SpoIIE is a modifier of Localization which is the head of the relation introduced by the preposition 'of'.

Participants were free to use all external information that they find useful, annotated Medline abstracts included. However, for this latter resource, they had to select abstracts later than year 2000 in order to avoid overlapping with the test data.

3.5 Training data set

The training set without coreferences includes 57 sentences describing *106 positive examples* of genic interactions:

- 70 examples of action
- 30 examples of binding and promoter
- 6 examples of regulon

The training set with coreferences includes 23 sentences describing *165 positive examples* of interactions with coreferences

- 42 examples of action
- 10 examples of binding and promoter
- 7 examples of regulon

There are then *271 training examples* in 80 sentences. The training data does not explicitly describe negative examples. A straightforward way for generating negative examples is to use the Closed-World Assumption: if no interaction is specified between two given biological objects A and B, then they do not interact and form a negative example. This way, they could be easily derived from the training data and the dictionary as near-miss examples.

3.6 Test set

The test data are examples from sentences following the same biological typology as the training data. The distribution of the positive examples among the biological categories (action, binding, promoter and regulon) and with / without coreferences is the same as in the training data. The test set also includes negative examples, namely sentences without any genic interaction. This set follows the same distribution as in the initial corpus selected by MedLine query and containing at least two gene names, i.e. 50 % of the sentences are negative. The test set includes 87 sentences describing *106 positive examples* of genic interactions:

- 55 examples of action
- 23 examples of binding and promoter
- 5 examples of regulon

There is no sentence in the test data with no clear separation between the agent and the target (*e.g.*, "gene products x and y are known to interact").

The distinction between the sentences, with and without coreferences is not done in the test set and

is not known by the participants because the test data set also contains sentences without any interaction. Marking "coreferences" sentences in the test set would bias the test task by giving hints for identifying the sentences without any interaction. However, the distinction is taken into account by the score computation.

4. Information extraction task

Given the description of the test examples and the named-entity dictionary, the task consists in automatically extracting the agent and the target of all genic interactions.

In order to avoid ambiguous interpretations, the agents and targets have to be identified by the canonical forms of their names as they are defined in the dictionary and by lemmas in the enriched version of the data. Thus there are two ways of retrieving the canonical name, given the actual name.

The agent and target roles should not be exchanged. If the sentence mentions different occurrences of an interaction between a given agent and target, then the answer should include all of them. For instance, in *A low level of GerE activated transcription of cotD by sigmaK RNA polymerase in vitro, but a higher level of GerE repressed cotD transcription*. There are two interactions to extract between GerE and cotD.

5. Computation of the score

The evaluation is based on the usual counting of false positive and false negative examples and on recall and precision. Partially correct answers will be considered as wrong answers. By partially correct answer we mean answers where the roles are exchanged, or only one of the two arguments (agent or target) of the genic interaction is correct. The score computation has been measured by the organizers on the results provided by the participants by applying the score computation program available to download as well as the check format program. These official scores are compared in section 6. The details on how scores are computed can be found on line in the user's manual of the score computation program.

The learning methods have been trained either on the file without coreferences or with coreferences, or on both of them (union). The participants have to specify which data set they compete for, so that the score computation program takes it into account for computing the scores.

The organizers also provide Web facilities to the participants for automatically uploading result files and compute the scores on the test data after the result submission deadline. These results have been further improved by the participants after the deadline. These "non official" results are not considered here for comparison because of the risk of over-fitting on the test data. However, they are interpreted and analyzed in the participant papers in this volume.

6. Result interpretation and comparison

Six research groups have participated in the challenge by submitting the results of the test set. The papers reporting their method and results are included in this volume. This section compares the official results among the participants.

6.1 Participating systems

Group 1 (KMB, Univ. Berlin and EBI) has applied alignment and finite-state automata technology for generating IE patterns from the LLL data set and an additional corpus of 256 positive examples manually annotated. The corpus has been enriched by POS tags and a list of words denoting interactions.

Group 2 (CS, Univ. Sheffield) method generates candidate patterns from examples parsed by MiniPar and semantically tagged by WordNet and PASBio. The candidates are manually filtered and then generalized with respect to a similarity criterion with already learned patterns. The training set has been augmented by weakly labeled training examples (cocitations of genes and proteins from positive examples, occurring in new sentences).

Group 3 (HCS Lab, Univ. Amsterdam) has applied the rule induction method Ripper to lexical-semantic-syntactic subtrees obtained by unification of the enriched form of the training examples. The semantics is given by an *ad'hoc* ontology designed for the challenge purpose.

Group 4 (KDLab, Univ. Brno) has applied the ILP method Aleph on the enriched data set without coreferences. Two features have been added, POS tags by the Brill tagger and WordNet hyperonyms.

Group 5 (Biostats and CS, Univ. Madison) has applied the ILP method Aleph on the enriched data set with and without coreferences wrapped into Gleaner that selects the best point on recall-precision curves. The data sets have been preprocessed and enriched by 215 new predicates including position, neighborhood, typographic, syntactic, semantic (belonging to MesH) and counting features.

Group 6 (ICCS, Univ. Edinburgh) has applied ILP and Markov Logic methods on the data parsed by the CCG and CCG2sem parsers that build syntactic and semantic paths. The best results are obtained without such preprocessing.

6.2 Results

Most of the results were obtained from the test set without coreferences (Table 1). The ML method of

Group 1. and 6. have achieved the best F-measures with balanced recall and precision around 50 %, which is high compared to other challenges on event or relation extraction such as the Succession Management MUC competition. Both systems are based on the representation of the examples as sequences. It would be interesting to study the role of the semantic tagging of word denoting interaction as done by Group 1. The other methods achieved a high recall but a poor precision. The reasons for such an overgeneralization could be explained by the fact that the training data did not include sentences without any interaction, as opposed to test data. The systems trained without such sentences or on weakly labeled additional data could have been thus handicapped. The results obtained with and without linguistic information cannot be easily compared here, since only Group 5. has provided results on both data sets. The role played by the syntactic dependencies cannot then be analyzed.

Table 1. Results on the test set without coreferences

Gr. #	Basic test set			Enriched test set		
	prec.	rec.	F	prec.	rec.	F
1.	50,0	53,8	51,8			
2.	10,6	98,1	19,1			
4.				37,9	55,5	45,1
5.	25,0	81,4	38,2	20,5	90,7	33,4
6.				60,9	46,2	52,6

Table 2. Results on the test set with coreferences

Gr. #	Basic test set			Enriched test set		
	prec.	rec.	F	prec.	rec.	F
5.	14,0	82,7	24,0	14,0	93,1	24,4

Table 3. Results on the test set with and without coreferences

Gr. #	Enriched test set		
	prec.	rec.	F
3.	51,8	16,8	25,4
6.	55,6	53,0	54,3

Table 2 presents the results as obtained on the test data with coreferences while Table 3 presents the results as obtained on the union of the test data with, and without coreferences. As shown by Table 2, the F-measure of Group 5 on the basic and linguistically enriched data set is not significantly different, as it is the case in Table 1. In all cases, the precision is poor, the recall high and the recall improved by the linguistic information.

Only the two groups 3. and 6. have provided results on the union of both test sets with and without coreferences. In both cases, the linguistic information has been exploited. Surprisingly, despite the difficulty of dealing with coreferences, the scores obtained on the set without coreferences (Table 1.) are similar: 52,6 against 54,3. Note that most of the coreferences in the test set were denoted by simple appositions and represented by explicit syntactic dependencies.

7. Conclusion

The high scores (more than 50 %) yields by the best system as well as by further experiments done by the other participants are very encouraging. As described in section 3., the data have been carefully selected in order to keep the underlying biological models simple. The parsing results as computed by LinkParser have been corrected by hand. The next challenges now consist in extending the data sets so that it becomes more representative of the real data as it can be found in MedLine abstracts and leave the syntactic parsing partially incorrect as it is when produced by automatic methods. The influence of the domain knowledge such as for instance, semantic classes of actions and their role in interactions has not been fully explored here but only through *ad'hoc* lists or patterns. It would certainly worthwhile to explore this direction.

Acknowledgements

MIG-INRA laboratory and LIPN-RCLN group have been deeply involved in the data preparation, biological annotation and syntactic dependency checking. The research and software development have been funded by the Caderige Project, Inter-EPST bioinformatics (1999-2003), ExtraPloDocs, RNTL (2000-2005) and Alvis, FP6-IST-STREP (2004-2007). The data preparation has been partially founded by Pascal FP6-IST-NoE (2004-2007).

References

Alphonse E., Aubin S., Bessières P., Bisson G., Hamon T., Lagarrigue S., Nazarenko A, Manine A.-P., Nédellec C., Ould Abdel Vetah M., Poibeau T. and Weissenbacher D. (2004). Event-Based Information Extraction for the biomedical domain: the Caderige project. Proceedings of the *Workshop BioNLP (Biology and Natural language Processing), Conférence Computational Linguistics* (Coling 2004).

Ciravegna F. (2000). Learning to Tag for Information Extraction from Text. Proceedings of the ECAI-2000 Workshop on Machine Learning for Information Extraction, F. Ciravegna *et al.* (eds), Berlin.

Cohen A. M, Hersh W. R. (2005). A survey of current work in biomedical text mining. Brief Bioinform. Mar;6(1):57-71..

Collier N., Ruch P. and Nazarenko A. (2004). Proceedings of the Joint Coling workshop on *Natural Language Processing in Biomedicine and its Applications*.

Daraselia N., Yuryev A., Egorov S., Novichkova S., Nikitin A., Mazo I. (2004). Extracting human protein interactions from MEDLINE using a full-sentence parser. *Bioinformatics*. 22;20(5):604-11.

Ding J., Berleant D., Nettleton D., Wurtele E. (2002). Mining MEDLINE: abstracts, sentences, or phrases? *Pac Symp Biocomputing* pp. 326-37.

Ding J., Berleant D., Xu J., and Fulmer A. W. (2003). Extracting Biochemical Interactions from MEDLINE Using a Link Grammar Parser. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*.

Freitag D. (1998). Toward General-Purpose Learning for Information Extraction. *Proceedings of COLING-ACL-98*.

Grover C., Lapata M., and Lascarides A. (2004). A Comparison of Parsing Technologies for the Biomedical Domain. *Journal of Natural Language Engineering*.

Hishiki T., Collier N., Nobata C., Ohta T., Ogata N., Sekimizu T., Steiner R., Park H. S., Tsujii J. (1998). Developping NLP tools for Genome Informatics: An Information Extraction Perspective. *Genome Informatics*. Universal Academy Press Inc., Tokyo, Japan.

Huang M., Zhu X., Hao Y., Payan D. G., Qu K., Li M. (2004). Discovering patterns to extract protein-protein interactions from full texts. *Bioinformatics*. 12;20(18):3604-12.

Leroy G., Chen H., Martinez J. D. (2003). A shallow parser based on closed-class words to capture relations in biomedical text. *J Biomed Inform*. Jun;36(3):145-58.

McDonald D. M., Chen H., Su H., Marshall B. B. (2004). Extracting gene pathway relations using a hybrid grammar: the Arizona Relation Parser., *Bioinformatics*. 12;20(18):3370-8..

Nédellec C. (2004). Machine Learning for Information Extraction in Genomics - State of the Art and Perspectives, *Text Mining and its Applications: Results of the NEMIS Launch Conference Series: Studies in Fuzziness and Soft Computing,* Sirmakessis, Spiros (Ed.), Springer Verlag.

Nédellec C., Ould Abdel Vetah M. and Bessières P. (2001). Sentence Filtering for Information Extraction in Genomics: A Classification Problem. In *Proceedings of the International Conference on Practical Knowledge Discovery in Databases* (PKDD'2001), pp. 326–338. Springer Verlag, LNAI 2167, Freiburg.

Ng S., Wong M. (2004). Toward routine automatic pathway discovery from on-line scientific text abstracts. *Genome Informatics*. 10:104-112.

Ono T., Hishigaki H., Tanigami A., Takagi T. (2001). Automated extraction of information on protein-protein interactions from the biological literature. *Bioinformatics*. 17(2): 155-161.

Park J. C., Kim H. S., Kim J. J. (2001). Bidirectional incremental parsing for automatic pathway identification with combinatory categorial grammar. In proceedings of *PSB'2001*.

Pyysalo S., Ginter F., Pahikkala T., Boberg J., Järvinen J., Salakoski T. and Koivula J. (2004). Analysis of Link grammar on Biomedical Dependency Corpus Targeted at Protein-Protein Interactions. Proceedings of the *Workshop BioNLP (Biology and Natural language Processing), Conférence Computational Linguistics* (Coling 2004).

Rindflesch T. C., Tanabe L., Weinstein J. N., Hunter L. (2000). EDGAR: Extraction of Drugs, Genes and Relations from the Biomedical Literature. *Proceedings of PSB'2000*, vol 5:514-525.

Roux C., Proux D., Rechenmann F., Julliard L. (2000). An Ontology Enrichment Method for a Pragmatic Information Extraction System gathering Data on Genetic Interactions. Proceedings of the *ECAI'2000 Ontology Learning Workshop*, S. Staab *et al.* (eds.).

Sasaki Y., Matsuo Y. (2000). Learning Semantic-Level Information Extraction Rules by Type-Oriented ILP. *Proceedings of COLING-2000*, Kay M. (ed), Saarbrücken.

Sleator D. and Temperley D. (1993). Parsing English with a Link Grammar. In *Third International Workshop on Parsing Technologies*. Tilburg. Netherlands.

Soderland S. (1999). Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning Journal*, vol 34.

Temkin J. M., Gilder M. R. (2003). Extraction of protein interaction information from unstructured text using a context-free grammar. *Bioinformatics*. Nov 1;19(16):2046-53.

Thomas J., Milward D., Ouzounis C., Pulman S., Carroll M. (2000). Automatic extraction of protein interactions from scientific abstracts. *PSB'2000*, pp 541-52.

Valencia A. and Blaschke C., (2004). Proceedings of the workshop *A critical assessment of text mining methods in molecular biology*, Spain.

Yakushiji A., Tateisi Y., Miyao Y., Tsujii J.-I., (2001). Extraction from biomedical papers using a full parser. *Proceedings of PSB'2001*.

LLL'05 Challenge: Genic Interaction Extraction - Identification of Language Patterns Based on Alignment and Finite State Automata

Jörg Hakenberg HAKENBERG@INFORMATIK.HU-BERLIN.DE
Conrad Plake PLAKE@INFORMATIK.HU-BERLIN.DE
Ulf Leser LESER@INFORMATIK.HU-BERLIN.DE
Knowledge Management in Bioinformatics, Humboldt-Universität zu Berlin, Germany

Harald Kirsch KIRSCH@EBI.AC.UK
Dietrich Rebholz-Schuhmann REBHOLZ@EBI.AC.UK
Rebholz-Group, European Bioinformatics Institute, Hinxton, United Kingdom

Abstract

We present a system for the identification of syntax patterns describing interactions between genes and proteins in scientific text. The system uses sequence alignments applied to sentences annotated with interactions and syntactical information (part-of-speech), as well as finite state automata optimized with a genetic algorithm. Both methods identified syntactical patterns that are generalizations of textual representations of agent-target relations. We match the generated patterns against arbitrary text to extract interactions and their respective partners. Our best system uses finite state automata optimized with a genetic algorithm, and scored an F1-measure of 51.8% on the LLL'05 evaluation set.

1. Introduction

The task for the *Learning Language in Logic (LLL'05)* challenge was to build systems that extract interactions between genes and/or proteins from biological literature. From sentences annotated with agent-target relations and other linguistic information, rules or models had to be learned and were evaluated afterwards (Genic Interaction Extraction Challenge, 2005). For this benchmark, not only the interacting partners had to be extracted, but also the agent-target dependencies had to be resolved correctly.

The task of relation mining in the biomedical domain has been studied widely over the last years. Current research covers protein-protein interactions (Huang et al., 2004; Daraselia et al., 2004), subcellular locations (Stapley et al., 2002), disease-treatment-relations (Rosario & Hearst, 2004), and certain other types. Early systems relied on simple co-occurrence analyses, and such techniques still provide very good recall performance for obvious reasons. Including syntactical information for shallow parsing leads to better results in the identification of co-occurring terms in conjunction with a verbal phrase (Chen & Sharp, 2004), *i.e.*, yields more precise predictions. Currently, systems based on sequence modeling, and pattern- or rule-based extraction provide the best results for detecting protein-protein interactions (Xiao et al., 2005; Huang et al., 2004; Saric et al., 2005).

The relation extraction system we used for the LLL'05 challenge consisted of two major components. The first extracted *syntax patterns* typical for textual descriptions of interactions from labeled examples. These patterns reside on part-of-speech information and markup of genes and selected nouns and verbs. We followed two strategies to learn such patterns from a given training sample. One was to apply a pattern generating algorithm comparable to the one described in Huang et al. (2004) to annotated examples. Subsequent *pairwise alignment* of sentences ultimately yielded patterns with as much support in the training data as possible. The other strategy was to generate *finite state automata* representing patterns, and to optimize these automata on the training sample using a *genetic algorithm* to achieve optimal results.

Appearing in *Proceedings of the 4^{th} Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

After patterns had been extracted, the second component matched these against arbitrary text to detect interactions.Matching was either based on aligning new sentences with patterns, or on mapping new sentences into FSAs, respectively.

All patterns we extracted from the given examples resembled sentences often used for describing interactions. Typical examples often found in the literature are, for instance,

ykuD is transcribed by SigK RNA polymerase	(1)
yfhS is transcribed by E sigma E	(2)
ComK regulates the expression of degR	(3)
GerE inhibits the transcription of sigK	(4)

It can easily be seen that (1) & (2), and (3) & (4) share a similar syntax, respectively. Replacing words with their respective part-of-speech tags, phrases (1) and (2) could be subsumed with the pattern

gene verb:p3s verb:pp preposition protein

This pattern describes every (partial) sentence that contains a gene, followed by a verb (present, third person singular), another verb (past participle), a preposition, and finally a protein[1]. The second verb could be narrowed down even further, by either accepting only verbs seen in the training data, or by compiling a list of possible verbs used for describing interactions. We followed the second idea, and had lists for verbs and nouns, which we refer to as *interaction verbs* and - *nouns* or simply *types* in the following (see Section 2.4.1 and supplementary information).

In this paper, we shall give an overview of all methods and algorithms used, present the preprocessing of the data set, and conclude with a discussion of our results and findings.

2. Methods and Algorithms

There were two possibilities to extract and represent patterns from a set of labeled examples. The first generated a set of patterns using pairwise alignments of sentences from the training sample. These alignments were transformed into a pattern. The second method was learning finite state automata from the training sample.

For both approaches, we represented each sentence as a sequence of POS tags instead of tokens. Additionally, we did not take the full sentences, but only the immediate phrases relevant to the description of the interaction. This included a certain boundary around the phrases (we experimented with up to three words to the left and right).

[1]In the following, we do not distinguish between genes and proteins.

Table 1. POS tags and costs for matches/mismatches. Costs shown: Gap - aligning POS tag with gap; Match - exact match; Group - match within group (same first letter); Other - mismatch.

POS Tag	Gap	Match	Group	Other
PTN	-10	+4		-3
IVERB	-10	+3		-3
INOUN	-8	+3		-3
NN/NNP	-8	+2	+1	-1
NNS/NNPS	-7	+2	+1	-1
VB/VBP/VBZ/ VBD/VBN/VBG	-7	+2	+1	-1
IN, CC, TO	-6	+2		-1
'(', ')', ','	-6	+2		-1
RB, RBR, RBS	-1	+2	+1	-1
JJ, JJR, JJS	-1	+2	+1	-1
DT, WDT	-1	+2		-1

2.1. Pairwise Alignment to Generate Patterns

2.1.1. Alignments

Quite a few methods are available to measure the similarity of two (or multiple) sentences. Our method was sequence alignment, which calculates a consensus sequence in addition to the similarity score. This consensus sequence represented all parts (POS tags and their positions) the aligned sequences had in common, and could be used directly to form a pattern. Figure 1 shows an example for two aligned sentences. For the alignment and scoring of sentence pairs, we implemented the local and end-space free alignments (Smith & Waterman, 1981; Needleman & Wunsch, 1970). With local alignment, the aligned sentences could be quite dissimilar overall, but had to contain regions that were highly similar (*e.g.*, a verb phrase or main clause). All other parts could have completely different syntax and semantics. Using the end-space free variation of global alignment, gaps at the beginning or end of a sentence had costs of zero, regardless of length. For calculating the costs for matches, mismatches, and gaps in the alignment, we used a substitution matrix covering the whole alphabet (19 part-of-speech tags). These costs were derived from previous experiments, and are are shown in Table 1. We found that while the exact values were less influential, the overall tendency of rewards and penalties for different sorts of replacements was important. A substitution matrix for an extended alphabet (full Penn Treebank tag-set) can be found in the supplementary information.

```
the oxidized cytochrome c binds    phosvitin and   ->   DT JJ PTN IVERB -- PTN CC
the           profilin    binds to  G-actin   ,    ->   DT -- PTN IVERB IN PTN ,
                                                      --------------------------
consensus sequence of both (partial) sentences     -    DT    PTN IVERB    PTN
```

Figure 1. Example for an alignment and consensus sequence of two POS tags sequences. Sentences are represented using POS tags. CC, conjunction; DT, determiner; IN: preposition; JJ: adjective; PTN, protein/gene; IVERB: verb describing an interaction.

2.1.2. Patterns

Each pattern consisted of two types of information: the pattern itself, *i.e* the sequence of POS tags, and the positions of agent, target, and interaction type in this pattern. A simple example for a pattern would be

$$\begin{array}{ll} \text{pattern} & = \text{PTN IVERB PTN} \\ \text{relation} & = \{(\text{a=1, t=3, i=2})\}, \end{array}$$

matching sentences with the basic structure "protein interaction-verb protein", for instance, "ComK regulates degR". The agent-target dependency is given via the tuple (a,t,i), where the first two digits refer to the positions of agent and target, and the third digit to the position of the interaction type, respectively. Please note that we did not distinguish between genes and proteins, but mapped both entities to the same tag, 'PTN'.

An example is shown in Figure 2. Each pattern p consisted of a POS sequence s, and a (set of) relations, K. Each element of K depicted a single interaction, and the triple (a, t, i) referred to the positions of agent, target, and interaction type within the sequence, respectively. Note that for the positions, we took into account only proteins and interaction nouns or verbs. $(1, 3, 2)$ thus stood for a relation between the agent protein at position 1 and the target protein at position 3, described by the verb at position 2.

2.1.3. Generating Patterns with Alignments

The pattern generating algorithm iterated over all sentence pairs and calculated the best alignment for each pair (see Section 2.1.1). Each respective consensus sequence from the optimal alignment of these two sentences formed a pattern. We counted the occurrences of all such patterns to calculate the support for each pattern in the training data. The maximum number of patterns for n sentences is $n(n-1)/2$, but in practice not all were generated, since a set of different alignments can lead to the same consensus sequence. On the other hand, in case there were multiple optimal alignments, all alignments were taken to form (different and/or identical) patterns. The algorithm in Figure 3 shows the pattern generating algorithm as pseudo-code.

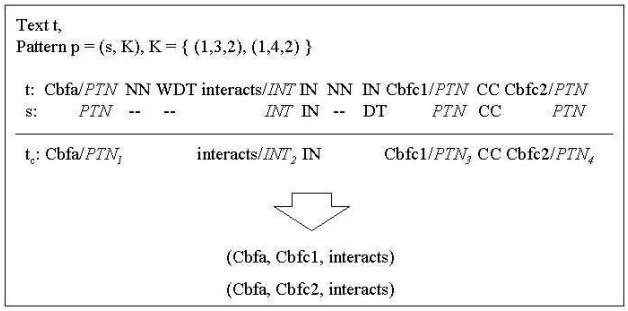


Figure 2. Text t aligned with pattern p. Consensus sequence is t_c, forming two interactions: agent_1 with target_3 via type_2, and agent_1 with target_4 via verb type_2, respectively.

Our pattern generating algorithm iteratively searched for the most similar sentences, and tried to subsume these with a more general expression, if necessary. As we took alignments for this step, we used the respective consensus sequences for the generalized sequences. This general expression, which we call pattern, was then added to a set. This set functioned as a model to explain the training data. Only patterns with at least two proteins and one interactor (specific verb or noun) were included in the final set. From the final set, all patterns below a minimum support (*i.e.* the number of aligned sentence pairs producing this pattern) were removed.

2.1.4. Applying Patterns to Arbitrary Text

We studied different methods to apply the generated patterns to arbitrary text: alignments, finite state automata, hidden Markov modeling, and hand-crafted rules (Plake et al., 2005a; Plake et al., 2005b). For the LLL'05 challenge, we used local and end-space-free alignment.

The alignment worked just like described in Section 2.1.1. Figure 2 shows an example for a pattern, its information, its alignment with a text and the resulting interactions. The positions for agent, target, and interaction type (the latter was not necessary for the

Input: set of annotated sequences, S;
threshold d
for all $s_i, s_j \in S$, $i \neq j$ **do**
$c_{i,j}$ = consensus(s_i, s_j)
$K = (pos_a, pos_b, pos_i)$ = positions of partners a, b, and interactor i in $c_{i,j}$
$p = (c_{i,j}, K)$
if $p \notin P$ **then**
add p to P; $occ_p = 1$
else
occ_p++
end if
end for
remove all $p \in P$ with $occ_p < d$
Output: set of patterns, P

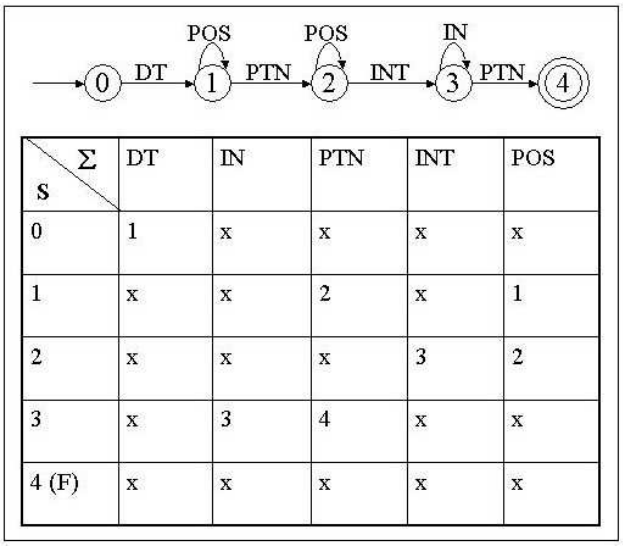
Figure 3. Pseudocode of the pattern generating algorithm.

LLL'05) in the new sentence were deduced from the information stored with each pattern. This was needed to solve the dependency between two proteins and for cases with multiple proteins in one sentence.

2.2. Learning Patterns with Finite State Automata

For the second approach to generate and represent patterns describing protein-protein interactions, we used *Mealy finite state automata* (FSAs). In these automata, each position in a sentence (*i.e.* part-of-speech tag) was represented by a transition from one state to another. Transitions thus modeled the sequence of POS tags in a sentence. The FSA has a dedicated *start state*, depicting the position before the first tag in a phrase. We stored each transition possible from each state to another in a matrix. For each state in the FSA, such a matrix contained all transitions possible to a new state using the next POS tag in the sequence. Figure 4 shows an example, where columns represent the alphabet (=POS tags), and rows the states, respectively. Each cell denoted whether a transition from the current using a particular POS tag was possible or not, and to which new state this transition would lead.

It was possible to further generalize FSAs. By introducing word gaps of variable length between states (*i.e.* between positions in a sentence), the FSA contained not only sentences fitting phrases exactly, but allowed for insertions (for example, short subordinate clauses or expressions in brackets). See Figure 5 for examples. Intuitively, a more "strict" FSA (no word gaps) would yield more precise results, while a "loose" FSA (word gaps of infinite length) would gain a higher recall, by fitting more sentences.



S \ Σ	DT	IN	PTN	INT	POS
0	1	x	x	x	x
1	x	x	2	x	1
2	x	x	x	3	2
3	x	3	4	x	x
4 (F)	x	x	x	x	x

Figure 4. Mealy-FSA and its transition matrix; s: number of states, Σ: alphabet of POS tags. For example, from state 1 a transition is possible either to state 2 using 'PTN', or back to state 1 using any other, non-listed POS tag.

We encoded the transition matrix described above in a single array. The value in each cell was transformed into a binary representation, depicting the index of a new state, when the transition using a particular POS tag was possible, or zero otherwise. In addition, we added a bit to this cell value. Whenever this bit was set to one, the transition led not only to a next state, but in addition, this state was an end-state. Table 2 shows an example for the binary representation of state 3 from Figure 4. In addition, the positions for agent(s), target(s), and interaction type(s) in the sentences had to be encoded. Taken together, this led to a binary representation of FSAs, which we refer to as a *genome*. In such a genome, all positions for the states, and all positions for transitions and end-states were fixed. Translations from a matrix to a genome and back thus were unambiguous.

The task now was to find one or more good FSAs encoded by such genomes. We first started with a set of random genomes, representing a *population*, where each individual encoded an FSA with six states[2]. Initially, most of the individual members of a population would encode a valid, but certainly a "bad performing" matrix. We optimized this population on the training sample using a genetic algorithm, see (Plake et al., 2005b). For this approach, we could use either preci-

[2]The number of states was evaluated in further experiments; data not shown.

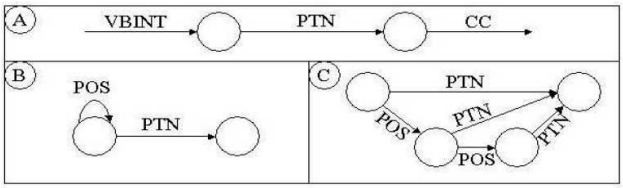


Figure 5. FSAs with word gaps. (A) shows the original transitions between two states, with a protein tag in the middle. In (B), the FSA was extended with an additional transition from the left state, using any POS tag, to the same state. This transition could be used multiple times. (C) shows an extension for a maximum of two, instead of multiple, optional POS tags.

Table 2. Binary representation of state 3 from the matrix in Figure 4. The table shows all possible transitions, and to which new state each transition leads. Using 'PTN', the transition from 3 to 4 would lead to an end-state.

POS tag	new state	end-state?	possible transition
DT	000	0	"no transition possible"
IN	011	0	"possible to state 3"
PTN	100	1	"to state 4 ⇒ end-state"
INT	000	0	"no transition possible"
POS	000	0	"no transition possible"

sion, recall, or f-measure as a *fitness function* to measure the performance of every individual FSA in the population. Choosing the best 25% of a population to form the basis for a new generation, we filled the missing 75% using *recombinations*, *cross-overs*, and *mutations* of this *parents*. Ultimately, this led to an FSA (the "best" genome in the population) covering at least some positive examples[3] from the training sample. As soon as no further improvements could be achieved, we removed all training sentences covered by this FSA. On the remaining sample, we started over with a completely new random population. We followed this *separate-and-conquer strategy*, until no positive sentences were left in the training sample, or a certain number of FSAs was found.

2.2.1. Applying Finite State Automata

In order to match interactions and to learn patterns based on FSAs, we had to foresee in our patterns that they match variability in the text. This meant that starting at the first tag in each phrase, a sequence of states including repetition of transitions had to be foreseen in our FSAs, ultimately leading to an end-state with the last tag. To parse complete sentences, we tested all sub-phrases starting at the first word, the second word, the third word, and so on. If the language of an FSA *covered* a phrase, then index positions similar to the ones described in 2.1.1 defined the interaction's partners and type.

[3] A positive example contains at least one interaction.

2.3. Data Sets

For generating patterns, we used two corpora in two separate runs. All contained sentences found in MedLine citations, together with markup for all gene/protein names, all interactions and their respective types (*i.e.* agent-target relations). The first corpus was the training data set provided with the LLL'05 challenge, consisting of 55 sentences with 103 genic interactions. No negative examples were included in this corpus. The other corpus had annotated protein-protein interactions, and consisted of 1000 sentences with 256 interactions in 174 sentences. This second corpus was derived from the BioCreAtIvE task 1A data set (Hirschman et al., 2005), in which gene and protein names were marked. Further annotations for protein-protein interactions based on this markup were added manually.

For the evaluation of strategies and methods, we generated patterns from the 1000 sentences, and applied them to the 55 sentences, on which we could measure the performance. The final test set consisted of 86 sentences containing genic interactions, and was provided with the LLL'05 challenge.

We tested different combinations of data sets for training and testing, and different methods for applying patterns to the respective test set. For the final solution, we learned patterns from our own corpus of 1000 sentences and combined them with patterns learned from the 55 training sentences.

2.4. Preprocessing

In order to apply our system to the training and test data, we had to perform several pre-processing steps. First of all, we transformed the data into XML format, including a proper tokenization. The indices of words in the word-list provided differed slightly from the corresponding token's index. Our tokenization included punctuation marks, brackets, and hyphens. We split expressions like *'sigma(K)-dependent'* into three tokens, *'sigma(K) - dependent'*.

2.4.1. Named Entity Recognition

A full list of all gene and protein names and synonyms or spelling variants appearing in the corpus was provided with the data. Named entity recognition could

Table 3. Slight modifications of the dictionaries and data sets.

- removed blanks	E sigma 27 (29, 43); E sigma A (D,E,F,G,H); sigma 29 (32, 70)
- removed symbols	sigma-43; Spo0A-P, Spo0A~P; SpoIIAA-P; alpha-amylase; PBP4*; PhoP~P
- removed symbols & blanks	pro-sigma E (K)
- added to dictionary	lacZ (ID 8169223-5,10767540-2); orf10 (10481082-2)
- altered in dictionary	sigma 27 = sigK according to 2492118-2
- added '.' at end of sentence	ID 10400595-1

thus be reduced to an exact matching of sentences against the dictionary. However, we introduced some minor modifications to cope with blanks, symbols, and overlapping gene names (*'sigma E'* versus *'E sigma E'* and *'pro-sigma E'*). We normally tackle the NER task using machine learning approaches based on support vector machines (Hakenberg et al., 2005). We did not distinguish between genes and proteins, neither in NER nor in patterns. In the literature, names referring to genes or proteins often are quite similar or even used as synonyms, and the exact meaning most times is hard to resolve.

Our pattern composition is based on *part-of-speech tags* annotated for each token. For POS-tagging, we used the TnT-Tagger, trained on the Wall-Street-Journal corpus (Brants, 2000), generating tags from the Penn Treebank tag-set (Santorini, 1990). In addition, we needed markup for tokens describing interactions, referred to as either *interaction nouns* or *verbs* (such as *'activation'*, or *'inhibits'*). We performed this step by combining POS-tag and word stem (Porter, 1980) to find an entry in fixed term-lists for nouns and verbs. The tokens in these lists were selected by experience gathered in previous studies, and additionally included suggestions from Temkin and Gilder (2003) as well (see supplementary information). We restricted the tags used in this approach to the ones shown in Table 1.

2.4.2. Dictionary and Canonical Forms

For the LLL data sets, we transformed every protein/gene name to its respective canonical form, as provided with the data. We subsequently matched every spelling variant of protein/gene names occurring in the corpus to its corresponding canonical form, starting with the longest synonyms to avoid errors in overlapping names. In addition, we altered some entries in the dictionary to deal with these problems. In general, in the refined dictionary, canonical forms did not have blanks or symbols (see Table 3). We added two names (lacZ and orf10), because they occurred in the corpus, but not in the dictionary.

Table 4. Patterns extracted from our corpus; table shows only patterns with a support of ≥30. Dependencies refer to different possibilities for agent/target relations; A: first protein in the sentences, B: second, C:third.

Pattern	Support	Dependencies
PTN IVERB PTN	1405	A→B, B→A, A↔B
PTN IVERB DT PTN	258	A→B, A↔B
PTN IVERB IN PTN	173	A→B, B→A
PTN INOUN PTN	138	A→B, B→A
PTN INOUN IN PTN	116	A→B, B→A
PTN IVERB IN DT PTN	46	A→B
PTN RB IVERB PTN	45	A→B, A↔B
INOUN IN PTN IN PTN	35	A→B, B→A
PTN IVERB NN PTN	35	A→B
PTN IVERB PTN PTN	30	A→B, A→C

Table 5. Performance on different types of interactions; all without linguistic information, without co-refs. Numbers in brackets give the total number of existing interactions (FN), and predicted interactions (FP), respectively.

	action	bind	regulon	nothing	all
TP	19	7	2	0	28
FN	17 (36)	5 (12)	2 (4)	0 (0)	24 (52)
FP	10 (29)	4 (11)	1 (3)	13 (13)	28 (56)

3. Results and Discussion

From our corpus of 1000 sentences, we were able to extract 148 patterns (see Table 4). The pattern with the highest support was "PTN IVERB PTN" – 1405 different alignments produced this sequence. The second best pattern, "PTN IVERB DT PTN" had a support of 258 only. These numbers included multiple optimal alignments for pairing two sentences. Table 4 shows the ten patterns with the highest support in the training data.

We decided to send the prediction (supposedly) having the highest F1-measure. This solution scored an F1 of 51.8% (precision of 50.0% at 53.8% recall; see Tables 5 and 6).

For the challenge, we did not use the co-references and linguistic information as provided with the corpus. Error analysis (for predictions on the training data) revealed that our system often predicted the in-

Table 6. Results from the evaluation for different methods and corpora. Upper corpus for training, lower for test; C_{1000}: corpus of 1000 sentences; C_{55}: genic_interaction_data; C_{86}: basic_test_data.

Method	Corpus	Pre	Rec	F1
Alignment	C_{1000} C_{55}	57.1	7.8	13.7
Alignment	C_{55} C_{86}	63.6	8.1	14.4
Mealy (2 FSAs)	C_{1000} C_{55}	64.3	17.5	27.5
Mealy (2 FSAs)	C_{1000} C_{86}	42.9	9.2	15.2
Mealy (4 FSAs)	C_{55} C_{86}	28.1	31.4	29.6
Mealy (5 FSAs)	$C_{1000+55}$ C_{86}	**50.0**	**53.8**	**51.8**

teracting partners correctly, but erred on the direction of the interaction, *i.e.* it interchanges agent and target. Using linguistic information, it might be possible to resolve some of these dependencies. Syntactic relations could help to generate more specific patterns containing whole phrases instead of single tags ("expression of rsfA"). For this specific phrases, certain positions could even be restricted to particular tokens, instead of general POS tags. Combinations of verbs and prepositions, for instance, contain linguistic information clearly stating the exact role of a gene/protein appearing before or after this verb phrase ("was phosphorylated by"). Grouping particular interaction types together (*e.g.*, "phosphorylation" and "methylation" refer to the creation of bonds, while "reduction" and "repression" imply an inactivation) and restricting patterns to these particular types might further help to exploit nomenclature usages.

We saw that the performance of our approach is clearly dependent on the size of the training set. Our system was able to detect only interactions for which it encountered a quite similar example in the training data. Using only 55 sentences for generating patterns proved insufficient, as many relations in the test set did not resemble any of these.

Our training corpus of 1000 sentences contained 256 different protein-protein interactions, but these were in general quite dissimilar from the genic interactions in the LLL data. This difference clearly had an impact on the performance. For instance, our examples did not include any descriptions of regulon family memberships at all. Most of our examples describe actions and bindings, and our system performed better in these categories (see Table 5).

Statistical analyses of the corpora revealed that the pattern protein-interactor-protein (PIP) was used in 78.1% of all cases, and IPP accounted for 18%, leaving 3.9% for PPI. In 58% of the relations, the agent was mentioned before the target.

The results of the system presented here were not overly satisfying. On our own corpus, alignments scored a precision of 91% or a recall of 65%, and Mealy-FSAs yielded 79% precision or 88% recall. These results could be confirmed on the IEPA corpus (Ding et al., 2002), which contains protein-protein interactions as well.

Supplementary Information

For supplementary information, please visit `http://www.informatik.hu-berlin.de/~hakenber/publ/suppl/`.

Abbreviations

FSA, finite state automaton; NER, named entity recognition; PGA, pattern generating algorithm; POS, part-of-speech (tag).

Acknowledgments

This work is supported by the German Federal Ministry of Education and Research (BMBF) under grant contract 0312705B. The Knowledge Management in Bioinformatics Group is a member of the Berlin Center for Genome Based Bioinformatics (BCB). Funding for the Rebholz group is provided by the Network of Excellence "Semantic Interoperability and Data Mining in Biomedicine" (NoE 507505). JH is additionally supported by the German Foreign Exchange Service (DAAD), reference number D/05/26768.

References

Brants, T. (2000). TnT - a statistical part-of-speech tagger. *Proc 6th Applied NLP Conference*. Seattle, USA.

Chen, H., & Sharp, B. M. (2004). Content-rich biological network constructed by mining PubMed abstracts. *BMC Bioinformatics*, *5*, 147.

Daraselia, N., Yuryev, A., Egorov, S., Novichkova, S., Nikitin, A., & Mazo, I. (2004). Extracting human protein interactions from medline using a full-sentence parser. *Bioinformatics*, *20*, 604–611.

Ding, J., Berleant, D., Nettleton, D., & Wurtele, E. (2002). Mining MEDLINE: Abstracts, Sentences, or Phrases? *Pacific Symposium on Biocomputing* (pp. 326–337). Kaua'i, Hawaii, USA.

Genic Interaction Extraction Challenge (2005). Learning Language in Logic Workshop (LLL). http://genome.jouy.inra.fr/texte/LLLchallenge/.

Hakenberg, J., Bickel, S., Plake, C., Brefeld, U., Zahn, H., Faulstich, L., Leser, U., & Scheffer, T. (2005). Systematic Feature Evaluation for Gene Name Recognition. *BMC Bioinformatics*, *6*, S9.

Hirschman, L., Yeh, A., Blaschke, C., & Valencia, A. (2005). Overview of BioCreAtIvE: critical assessment of information extraction for biology. *BMC Bioinformatics*, *6*, 1.

Huang, M., Zhu, X., Hao, Y., Payan, D. G., Qu, K., & Li, M. (2004). Discovering patterns to extract protein-protein interactions from full texts. *Bioinformatics*, *20*, 3604–3612.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, *48*, 443–53.

Plake, C., Hakenberg, J., & Leser, U. (2005a). Learning Patterns for Information Extraction from Free Text. *Proc Workshop des Arbeitskreises Knowledge Discovery*. Karlruhe, Germany.

Plake, C., Hakenberg, J., & Leser, U. (2005b). Optimizing Syntax Patterns for Discovering Protein-Protein Interactions. *Proc ACM Symposium for Applied Computing, Bioinformatics track*. Santa Fe, USA.

Porter, M. (1980). An algorithm for suffix stripping. *Program*, 130–137.

Rosario, B., & Hearst, M. (2004). Classifying semantic relations in bioscience texts. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, ACL*. Barcelona, Spain.

Santorini, B. (1990). *Part-of-speech tagging guidelines for the Penn Treebank Project* (Technical Report). MS-CIS-90-47, University of Pennsylvania.

Saric, J., Jensen, L., Ouzounova, R., Rojas, I., & Bork, P. (2005). Large-scale Extraction of Protein/Gene Relations for Model Organisms. *Proc Symp on Semantic Mining in Biomedicine* (p. 50). Hinxton, UK.

Smith, T., & Waterman, M. (1981). Identification of common molecular subsequences. *J. Mol. Biol.*, *147*, 195–197.

Stapley, B., Kelley, L., & Sternberg, M. (2002). Predicting the sub-cellular location of proteins from text using support vector machines. *Proceedings of the Pacific Symposium on Biocomputing* (pp. 374–385).

Temkin, J. M., & Gilder, M. R. (2003). Extraction of protein interaction information from unstructured text using a context-free grammar. *Bioinformatics*, *19*, 2046–2053.

Xiao, J., Su, J., Zhou, G., & Tan, C. (2005). Protein-Protein Interaction Extraction: A Supervised Learning Approach. *Proc Symp on Semantic Mining in Biomedicine* (pp. 51–59). Hinxton, UK.

Automatically Acquiring a Linguistically Motivated Genic Interaction Extraction System

Mark A. Greenwood M.GREENWOOD@DCS.SHEF.AC.UK
Mark Stevenson M.STEVENSON@DCS.SHEF.AC.UK
Yikun Guo G.YIKUN@DCS.SHEF.AC.UK
Henk Harkema H.HARKEMA@DCS.SHEF.AC.UK
Angus Roberts A.ROBERTS@DCS.SHEF.AC.UK
Department of Computer Science, University of Sheffield, Sheffield, S1 4DP, UK

Abstract

This paper describes an Information Extraction (IE) system to identify genic interactions in text. The approach relies on the automatic acquisition of patterns which can be used to identify these interactions. Performance is evaluated on the Learning Language in Logic (LLL-05) workshop challenge task.

1. Extraction Patterns

The approach presented here uses extraction patterns based on paths in dependency trees (Lin, 1999). Dependency trees represent sentences using dependency relationships linking each word in the sentence with the words which modify it. For example in the noun phrase *brown dog* the two words are linked by an adjective relationship with the noun *dog* being modified by the adjective *brown*. Each word may have several modifiers but each word may modify at most one other word.

In these experiments the extraction patterns consist of linked chains, an extension of the chain model proposed by Sudo et al. (2003) which represents patterns as any chain-shaped path in a dependency tree starting from a verb node. Our model extends this to patterns produced by joining pairs of chains which share a common verb root but no direct descendants. For example the fragment *"...*AGENT *represses the transcription of* TARGET*..."* can be represented by the dependency tree in Figure 1. From such a tree we extract all the chains and linked chains that contain at least one semantic category giving the 4 patterns (2 chains and 2 linked chains) shown in Table 1.

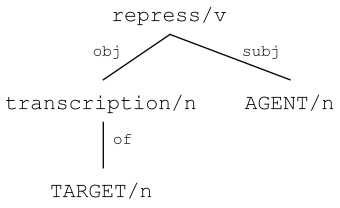


Figure 1. An example dependency tree.

The nodes in the dependency trees from which our patterns are derived can be either a lexical item or a semantic category such as gene, protein, agent, target, etc. Lexical items are represented in lower case and semantic categories are capitalised, e.g. in `verb[v/transcribe](subj[n/GENE]+obj[n/PROTEIN])`[1], `transcribe` is a lexical item while `GENE` and `PROTEIN` are semantic categories which could match any lexical item of that type. These patterns can be used to extract interactions from parsed text by matching against dependency trees.

2. Extraction Pattern Learning

Our approach learns patterns automatically by identifying those with similar meanings to a set of seed patterns known to be relevant. The motivation behind this approach is that language is often used to express the same information in alternative ways. For example *"*AGENT *represses the transcription of* TARGET*"*, *"the transcription of* TARGET *is repressed by* AGENT*"*, and *"*TARGET *(repressed by* AGENT*)"* describe the same interaction. Our approach aims to identify various ways interactions can be expressed by identifying patterns

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

[1] In this pattern representation `+` signifies that two nodes are siblings and a nodes descendants are grouped within `(` and `)` directly after the node.

```
verb[v/repress](subj[n/AGENT])
verb[v/repress](obj[n/transcription](of[n/TARGET]))
verb[v/repress](obj[n/transcription]+subj[n/AGENT])
verb[v/repress](obj[n/transcription](of[n/TARGET])+subj[n/AGENT])
```

Table 1. The patterns extracted from the dependency tree in Figure 1.

which paraphrase one another. A similar method is outlined in more detail in Stevenson and Greenwood (2005).

Extraction patterns are learned using a weakly supervised bootstrapping method, similar to that presented by Yangarber (2003), which acquires patterns from a corpus based upon their similarity to patterns which are known to be useful. The general process of the learning algorithm is as follows:

1. For a given IE scenario we assume the existence of a set of documents against which the system can be trained. The documents are unannotated and may be either relevant (contain the description of an event relevant to the scenario) or irrelevant although the algorithm has no access to this information.
2. This corpus is pre-processed to generate the set of all patterns which could be used to represent sentences contained in the corpus, call this set S. The aim of the learning process is to identify the subset of S representing patterns which are relevant to the IE scenario.
3. The user provides a small set of seed patterns, S_{seed}, which are relevant to the scenario. These patterns are used to form the set of currently accepted patterns, S_{acc}, so $S_{acc} \leftarrow S_{seed}$. The remaining patterns are treated as candidates for inclusion in the accepted set, these form the set $S_{cand} (= S - S_{acc})$.
4. A function, f, is used to assign a score to each pattern in S_{cand} based on those which are currently in S_{acc}. This function assigns a real number to candidate patterns so $\forall\ c \in S_{cand},\ f(c, S_{acc}) \mapsto \mathbb{R}$. A set of high scoring patterns (based on absolute scores or ranks after the set of patterns has been ordered by scores) are chosen as being suitable for inclusion in the set of accepted patterns. These form the set S_{learn}.
5. The patterns in S_{learn} are added to S_{acc} and removed from S_{cand}, so $S_{acc} \leftarrow S_{acc} \cup S_{learn}$ and $S_{cand} \leftarrow S_{acc} - S_{learn}$.
6. If a suitable set of patterns has been learned then stop, otherwise return to step 4.

The most important stage in this process is step 4; the task of identifying the most suitable pattern from the set of candidates. We do this by finding patterns that are similar to those already known to be useful. Similarity is measured using a vector space model inspired by that commonly used in Information Retrieval (Salton & McGill, 1983). Each pattern is represented as a set of pattern element-filler pairs. For instance, the pattern `verb[v/transcribe](subj[n/GENE]+obj[n/PROTEIN])` contains the pairs `verb_transcribe`, `subj_GENE` and `obj_PROTEIN`. The set of element-filler pairs in a corpus can be used to form the basis for a vector space in which each pattern can be represented as a binary vector (where the value 1 for a particular element denotes the pattern contains the pair and 0 that it does not). The similarity of two pattern vectors can be compared using Equation 1.

$$similarity(\vec{a}, \vec{b}) = \frac{\vec{a} W \vec{b^T}}{|\vec{a}||\vec{b}|} \quad (1)$$

Here $\vec{a}$ and $\vec{b}$ are pattern vectors, $\vec{b^T}$ the transpose of $\vec{b}$, and W a matrix listing the semantic similarity between each of the possible pattern element-filler pairs which is crucial for this measure. Assume that the set of patterns, P, consists of n element-filler pairs denoted by $p_1, p_2, ...p_n$. Each row and column of W represents one of these pairs. So, for any i such that $1 \leq i \leq n$, row i and column i are both labelled with pair p_i. w_{ij} is the element of W in row i and column j and is the similarity between p_i and p_j. Pairs with different pattern elements (i.e. grammatical roles) have a similarity score of 0. The remaining elements of W represent the similarity between the filler of pairs of the same element type. Similarity is determined using a metric defined by Banerjee and Pedersen (2002) which uses the WordNet lexical database (Fellbaum, 1998)[2]. This metric measures the relatedness of a pair of words by examining the number of words that are common in their definitions.

Figure 2 shows an example using three potential extraction patterns:

[2]This measure was chosen since it allows relatedness scores to be computed for a wider range of grammatical categories than alternative measures.

Extraction Patterns

a. `verb[v/block](subj[n/protein])`
b. `verb[v/repress](subj[n/enzyme])`
c. `verb[v/promote](subj[n/protein])`

Matrix Labels

1. `subj_protein`
2. `subj_enzyme`
3. `verb_block`
4. `verb_repress`
5. `verb_promote`

Similarity Matrix

1	0.95	0	0	0
0.95	1	0	0	0
0	0	1	0.9	0.1
0	0	0.9	1	0.1
0	0	0.1	0.1	1

Similarity Values

$\text{sim}(\vec{a}, \vec{b}) = 0.925$
$\text{sim}(\vec{a}, \vec{c}) = 0.55$
$\text{sim}(\vec{b}, \vec{c}) = 0.525$

Figure 2. Similarity scores and matrix for an example vector space using three patterns.

`verb[v/block](subj[n/protein])`
`verb[v/repress](subj[n/enzyme])`
`verb[v/promote](subj[n/protein])`

This example shows how these patterns can be represented as vectors and gives a sample semantic similarity matrix. It can be seen that the first pair of patterns are the most similar using the proposed measure despite the fact they have no lexical items in common.

The measure shown in Equation 1 is similar to the cosine metric, commonly used to determine the similarity of documents in the vector space model approach to Information Retrieval. However, the cosine metric will not perform well for our application since it does not take into account the similarity between elements of a vector and would assign equal similarity to each pair of patterns in this example[3].

The second part of a pattern element-filler pair can be a semantic category, such as `GENE`. The identifiers used to denote these categories do not appear in WordNet and so it is not possible to directly compare their similarity with other lexical items. To avoid this problem such tokens are manually mapped onto the most appropriate node in the WordNet hierarchy which is then used in similarity calculations.

An associated problem is that WordNet is a domain independent resource and may list several inappropriate meanings for domain specific words. For example WordNet lists five senses of the word *transcribe*, only one of which is related to the biomedical domain. To alleviate this problem domain specific restrictions are applied to WordNet. In these experiments only specific senses of 58 words are used with the alternative senses for each word being ignored by the system. These 58 words include the 30 verbs detailed in the PASBio project[4] (Wattarujeekrit et al., 2004) and 28 words determined by manual analysis of MedLine abstracts. For example, *transcribe* contains five senses in WordNet but our system considers only the final one; *convert the genetic information in (a strand of DNA) into a strand of RNA, especially messenger RNA.*

We experimented with several techniques for ranking candidate patterns to decide which patterns to learn at each iteration of our algorithm and found the best results were obtained when each candidate pattern was compared against the centroid vector of the currently accepted patterns. At each iteration we accept the four highest scoring patterns whose score is within 0.95 of the best pattern being accepted. For further details of the same approach using predicate-argument structures to perform sentence filtering, see Stevenson and Greenwood (2005).

3. Pattern Acquisition

Two training corpora were used for the experiments reported in this paper:

Basic The basic data set, without coreference, as provided by the LLL-05 challenge organizers.

Expanded The basic data set expanded with 78 automatically acquired *weakly labelled* (Craven & Kumlien, 1999) MedLine sentences. This extra training data was obtained by extracting, from MedLine abstracts[5] containing the phrase *Bacillus subtilis*, those sentences which contain two dictionary entries (or their synonyms) which are known to form an interaction in the basic training data.

The training corpora are pre-processed to produce one sentence per known interaction, replacing the agent and target by representative tags, `AGENT` and `TARGET`, and all other dictionary elements by the tag `OTHER`. The resulting sentences are then parsed using MINI-

[3]The cosine metric for a pair of vectors is given by the calculation $\frac{a.b}{|a||b|}$. Substituting the matrix multiplication in the numerator of Equation 1 for the dot product of vectors $\vec{a}$ and $\vec{b}$ would give the cosine metric. Note that taking the dot product of a pair of vectors is equivalent to multiplying by the identity matrix, i.e. $\vec{a}.\vec{b} = \vec{a}I\vec{b^T}$. Under our interpretation of the similarity matrix, W, this equates to saying that all pattern element-filler pairs are identical to each other and not similar to anything else.

[4]`http://research.nii.ac.jp/~collier/projects/PASBio/`

[5]Only abstracts which appeared after the year 2000 were used in order to comply with the LLL challenge guidelines.

PAR (Lin, 1999) to produce dependency trees from which the candidate extraction patterns (in the form of chains and linked chains) are extracted.

The learning algorithm was used to learn two sets of extraction patterns using the pair of corpora and the seed patterns in Table 2 which where chosen following a manual inspection of the training data. Due to the small amount of training data the learning algorithm was allowed to run until it was unable to learn any more patterns. When trained using the basic corpora the algorithm ran for 74 iterations and acquired 127 patterns. When trained using expanded corpora the algorithm ran for 130 iterations and acquired 236 patterns.

Not all the extraction patterns acquired in this way encode a complete interaction, i.e. they do not contain both `AGENT` and `TARGET` slots. To generate full interactions those agents and targets which are extracted are joined together using the following heuristics:

- Each `AGENT` extracted is paired with all the `TARGET` instances extracted from the same sentence (vice-versa for `TARGETS`).
- Each `AGENT/TARGET` discovered by a pattern is paired with the closest (distance measured in words) dictionary element.

For example imagine a sentence in which all the agents and targets discovered by extraction patterns are tagged as `AGENT` or `TARGET`, all other dictionary elements are replaced by `OTHER`: *`TARGET`*$_1$ *blocks `AGENT` and `OTHER` which inhibits `TARGET`*$_2$. From this sentence the following interactions would be extracted `AGENT`→`TARGET`$_1$, `AGENT`→`TARGET`$_2$ and `AGENT`→`OTHER`, i.e. the `AGENT` would be paired with all `TARGET` instances as well as the closest dictionary element.

4. A Baseline System

A baseline system was developed for comparison with our main approach. This baseline system assumes that interactions exist between all possible pairs of named entities in any given sentence (participants were provided with an exhaustive named entity dictionary). For instance, given a sentence containing three named entities labelled A, B and C, six interactions AB, AC, BA, BC, CA and CB are generated. This baseline will identify many interactions although the precision is likely to be low as many incorrect interactions will also be generated.

5. Evaluation

The official evaluation results, for both the baseline system and the systems trained using the two corpora detailed in Section 3, can be seen in Table 3.

We may expect the baseline system to achieve 100% recall by proposing a link between each pair of entities in each sentence. However certain constructions describe two relations between a pair of entities. For example *"...A activates or represses B..."* describes both repression and activation relationships between A and B while the baseline would propose just one.

In comparison with the baseline system our machine learning approach to pattern acquisition performed poorly due to low recall, although with a precision score over twice that of the baseline. The performance can probably be attributed to the small amount of available training data. It is clear that adding just a small amount of additional training data (78 sentences from MedLine) had a positive effect increasing the overall F-measure from 14.8% to 17.5%. The same effect can be seen if we consider the performance of the systems over the three interaction types; action, bind and regulon. The system trained using just the basic data finds 6 correct interactions 5 of which are actions and 1 a binding interaction (see Table 4 for a full breakdown of the results for all three submissions). The system fails to find any regulon family interactions. This is understandable given the training data which contains different percentages of each of the three interaction types. For instance only three sentences containing a regulon family interaction are provided illustrating just six interactions. Given our method of pattern acquisition this means that even if all the relevant patterns from these three sentences are learnt they would only apply to very similar sentences when used for extraction as they will not have been able to generalise far enough away from the specific instances present in the three example sentences.

5.1. Additional Evaluation

We carried out additional evaluations after the official results for the challenge task had been released.

A more detailed evaluation of the learning algorithm considers the performance of the patterns acquired at each separate iteration as opposed to the results in the previous section which evaluate all the acquired patterns as a single set. Figure 3 shows the F-measure score of the system trained using the expanded corpus (see Section 3) at each iteration of the learning algorithm.

This evaluation highlights a number of interesting

```
verb[v/transcribe](by[n/AGENT]+obj[n/TARGET])
verb[v/be](of[n/AGENT]+s[n/expression](of[n/TARGET]))
verb[v/inhibit](obj[n/activity](nn[n/TARGET])+subj[n/AGENT])
verb[v/bind](mod[r/specifically](to[n/TARGET])+subj[n/AGENT])
verb[v/block](obj[n/capacity](of[n/TARGET])+subj[n/AGENT])
verb[v/regulate](obj[n/expression](nn[n/TARGET])+subj[n/AGENT])
verb[v/require](obj[n/AGENT]+subj[n/gene](nn[n/TARGET]))
verb[v/repress](obj[n/transcription](of[n/TARGET])+subj[n/AGENT])
```

Table 2. Seed patterns used for pattern acquisition.

System	P	R	F
Baseline	10.6% (53/500)	98.1% (53/54)	19.1%
LLL-05 Basic	22.2% (6/27)	11.1% (6/54)	14.8%
LLL-05 Expanded	21.6% (8/37)	14.8% (8/54)	17.5%

Table 3. Evaluation results of our three submissions.

	All Interactions			Action			Bind			Regulon			No Interaction		
System	C	M	S	C	M	S	C	M	S	C	M	S	C	M	S
Baseline	53	1	447	35	1	95	14	0	46	4	0	6	0	0	300
LLL-05 Basic	6	48	21	5	31	7	1	13	2	0	4	0	0	0	12
LLL-05 Expanded	8	46	29	7	29	11	1	13	2	0	4	0	0	0	16

Table 4. Breakdown of the official evaluation results including results for individual interaction types (columns represent **C**orrect, **M**issing, and **S**purious). Precision = **C**/(**C**+**S**), Recall = **C**/(**C**+**M**)

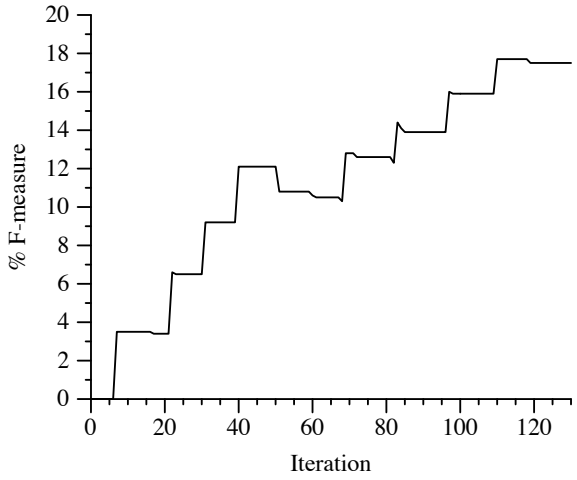


Figure 3. Increasing F-measure scores.

points. Firstly the seed patterns (Table 2) while being possibly representative of the training data do not match any of the interactions in the test set (i.e. the F-measure at iteration zero is 0% reflecting the fact that no correct interactions were extracted by the seed patterns). This is unfortunate as the learning algorithm is designed to acquire patterns which are similar in meaning to a set of known *good* patterns. In this instance, however, the algorithm started by acquiring patterns which are similar to the seeds but which clearly do not represent the interactions in the test set. However, this also means that those interactions extracted by the completed system were done so using only patterns acquired during training and not hand-picked good quality seed patterns.

The per-iteration evaluation in Figure 3 also shows that the learning algorithm is relatively stable even when inappropriate patterns are acquired. At least one pattern is acquired at each iteration and these results show that even if patterns are not able to extract valid interactions they rarely affect the performance of the current set of acquired patterns. The notable exception to this is at iteration 51 when a pattern is acquired which drops the F-measure from 12.1% to 10.8%, although further analysis shows that this was in fact a problem with the extraction procedure and not the acquired pattern. The algorithm acquired the pattern `verb[v/contain](obj[n/TARGET]+subj[n/AGENT])`. Un-

fortunately while the TARGET usually matches against a dictionary element the AGENT often matches other text. This causes the nearest (in words) dictionary element to be used as the AGENT which, in turn, can lead to incorrect interactions being extracted from text.

This analysis of the system's failings highlights a useful feature of our approach. Many machine learning algorithms produce classifiers which are statistical in nature and do not consist of a set of rules but rather a complex combination of probabilities. This makes it difficult to analyse classification mistakes and does not allow the ability to modify the classifier by removing badly performing rules. In contrast to this our approach learns human readable extraction rules which can be easily inspected, modified or removed to suit a given scenario. This allows an expert to examine the extraction rules while automating the time consuming process of rule acquisition.

5.2. Sentence Filtering

Our approach to automatically acquiring IE patterns has been shown to be suitable for determining the relevance of sentences for an extraction task in the management succession domain (Stevenson & Greenwood, 2005). The sentence filtering task involves using the set of acquired patterns to classify each sentence in a corpus as either relevant (containing the description of an interaction) or not. Sentence filtering is an important preliminary stage to full relation extraction. Using the patterns acquired from the expanded corpus (described in Section 3) we can also perform sentence filtering of the LLL challenge test data[6]. The results of this filtering, at different iterations of the algorithm, can be seen in Figure 4.

These results show that set of acquired patterns achieves an F-measure score of 47.5% resulting from precision and recall scores of 57.6% and 40.4% respectively. This compares to results reported by Nédellec et al. (2001) who achieve an F-measure score of approximately 80% over similar data using a supervised approach in which the learning algorithm was aware of the classification of the training instances. It should be noted that our approach was trained using only a small amount of unlabelled training data (181 sentences compared with approximately 900 sentences used by Nédellec et al. (2001)) and the sentence filtering results should be considered in this context.

[6]Thanks to Claire Nédellec for providing the relevant/not-relevant labelling of the sentences required for this evaluation.

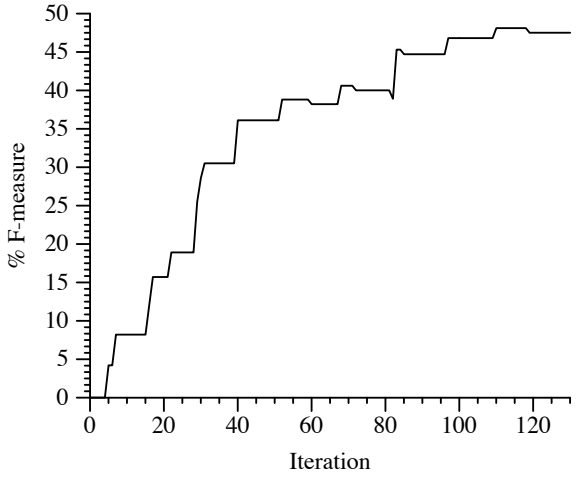


Figure 4. BioMedical Sentence Filtering.

6. Failure Analysis

The experiments reported in this paper have shown that our system is disappointing when used to perform relation extraction. The main failure of the system to extract meaningful relations can be traced back to the lack of training data. When extra data obtained from MedLine was also used to train the system there was an improvement in performance, acquiring more data may further improve performance. Another possible solution to this problem would be to generalise the acquired patterns in some form, perhaps by allowing any synonym of a pattern element filler to match. These could be extracted from WordNet.

One further source of failure was due to errors in the dependency trees introduced by MINIPAR. This is probably because the parser was not trained on biomedical texts and hence suffers from problems with unknown words and grammatical constructions. The approach here relies heavily on access to accurate dependency tree representations of text.

7. Conclusions

In this paper we have presented a linguistically motivated approach to extracting genic interactions from biomedical text. Whilst the performance of the system was disappointing achieving an F-measure score of only 17.5% we believe that the approach is well motivated but suffers from a lack of training data and parsing problems. We showed that increasing the training data using weakly labelled text did in fact increase the performance of the system. The additional evaluation of the extraction patterns showed that the approach is also resilient to the algorithm learning inappropriate extraction patterns.

Acknowledgements

This work was carried out as part of the RESuLT project funded by the Engineering and Physical Sciences Research Council (GR/T06391).

References

Banerjee, S., & Pedersen, T. (2002). An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. *Proceedings of the Fourth International Conference on Computational Linguistics and Intelligent Text Processing (CICLING-02)* (pp. 136–145). Mexico City.

Craven, M., & Kumlien, J. (1999). Constructing Biological Knowledge Bases by Extracting Information from Text Sources. *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology* (pp. 77–86). Heidelberg, Germany: AAAI Press.

Fellbaum, C. (Ed.). (1998). *Wordnet: An electronic lexical database and some of its applications*. Cambridge, MA: MIT Press.

Lin, D. (1999). MINIPAR: a minimalist parser. *Maryland Linguistics Colloquium*. University of Maryland, College Park.

Nédellec, C., Vetah, M. O. A., & Bessières, P. (2001). Sentence Filtering for Information Extraction in Genomics, a Classification Problem. *Proceedings of the Conference on Practical Knowledge Discovery in Databases (PKDD'2001)* (pp. 326–338). Freiburg, Germany.

Salton, G., & McGill, M. (1983). *Introduction to modern information retrieval*. New York: McGraw-Hill.

Stevenson, M., & Greenwood, M. A. (2005). A Semantic Approach to IE Pattern Induction. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*.

Sudo, K., Sekine, S., & Grishman, R. (2003). An Improved Extraction Pattern Representation Model for Automatic IE Pattern Acquisition. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)* (pp. 224–231).

Wattarujeekrit, T., Shah, P., & Collier, N. (2004). PASBio: Predicate-Argument Structures for Event Extraction in Molecular Biology. *BMC BioInformatics*, *5:155*.

Yangarber, R. (2003). Counter-training in the discovery of semantic patterns. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)* (pp. 343–350). Sapporo, Japan.

Learning Biological Interactions from Medline Abstracts

Sophia Katrenko KATRENKO@SCIENCE.UVA.NL
Human-Computer Studies Laboratory, University of Amsterdam, Kruislaan 419, 1098 VA, Amsterdam

M. Scott Marshall MARSHALL@SCIENCE.UVA.NL
Integrative Bioinformatics Unit, University of Amsterdam, Kruislaan 318, 1098 SM Amsterdam

Marco Roos ROOS@SCIENCE.UVA.NL
Integrative Bioinformatics Unit, University of Amsterdam, Kruislaan 318, 1098 SM Amsterdam

Pieter Adriaans PIETERA@SCIENCE.UVA.NL
Human-Computer Studies Laboratory, University of Amsterdam, Kruislaan 419, 1098 VA, Amsterdam

Abstract

In this paper, we describe our approach to the Genic Interaction Extraction Challenge. Our solution combines several elements: 1) a domain theory about the interaction between language, semantics and syntax, 2) a biological ontology identifying amongst other things biomolecular entities and directed interaction verbs in the lexicon, 3) the notion of lexical-semantic-syntactic unification, 4) the notion of partial unification of lexical-semantic-syntactic trees and 5) the application of the standard RIPPER algorithm to the results. Using this approach on the very limited training and test data from the Challenge we show results that are promising. Our method observes a clear separation between domain-independent and domain-specific components. It can therefore easily be extended to other domains. We briefly describe the implementation of the techniques, discuss the results and give suggestions for improvements and further results in the conclusion.

1. Introduction

Extracting interactions from texts is an active field of research in the biomedical domain (Krallinger, M., 2005, for a recent review). Interactions between proteins and genes are often considered essential in the description of biomolecular phenomena, and networks of (protein) interactions are considered as an entré for a Systems Biology approach (Uetz, P. 2005). Interaction networks extracted from literature (Chen, H. 2004) complement interaction data obtained from high-throughput laboratory experiments (Xenarios, I., 2001). Our approach falls into a class that explores the application of semantic models (Yandell, M. D. 2002; Muller, H. M. 2004).

The paper is organized as follows. We first define the task and data supplied. Then, we proceed with the overview of the system which has been developed. The paper concludes with results received on the Challenge data and gives an outlook.

2. Challenge task

2.1. Biology

The Genic Interaction Extraction Challenge is aimed at learning interactions between agents and targets described by individual sentences that have been extracted from a Medline collection of abstracts. As is typical in articles about biology, the abstracts describe the results from various types of experiments and use a mix of biological viewpoints and jargon. The sentences have been extracted from different abstracts, thus there is no relation among the sentences on the discourse level.

For the purposes of the Challenge, a simplified notion of 'interaction' is used: an agent/target pair of (the names of) genes and/or proteins. Gene names can pertain to genetic entities such as the DNA code for a given gene or to physical entities, in which case they

Appearing in *Proceedings of the 4nd Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

may indirectly refer to gene products, i.e. proteins. However, it might happen that the same name for a biomolecular entity can refer to both the gene and the gene product and thereby perform the role of both agent and target. On the other hand, it is also possible to encounter more than one agent for a given target and vice versa.

A typical sentence with explicit interaction stated is the following:

Localization of SpoIIE was shown to be dependent on the essential cell division protein FtsZ.

where *SpoIIE* is the target and *FtsZ* is the agent and the interaction is represented as *genic_interaction(FtsZ,SpoIIE)*. As one may notice, this relation is not symmetric.

We created a simple ontology specifically for use in this Challenge. The main purpose of the ontology was as a proof of principle for the use of semantics during text mining.

2.2. Data

There were 77 annotated training sentences provided by the organizers of the Challenge. Because we feel that syntactic information is vital, we only worked with the enriched training sets. An example sentence contains the following elements: a sentence-ID, the sentence itself, a corresponding list of words, a list of lemmas (stemmed versions of the words), a list of syntactic relations, a list of agents, targets and interactions. They can be grouped according to the presence/absence of the co-references into 22 sentences with co-references and 55 without them. The test set contains 87 sentences in the same format but without the agent-target information. In contrast to the training set, it also contains negative examples.

3. System overview

Following (Russell & Norvig 2003), the general knowledge-based inductive learning has to solve the following constraint:

$$Background \wedge Hypothesis \wedge Descriptions$$

$$\models Classifications$$

In our case, Background can be thought of as ontology and the knowledge it encapsulates, Descriptions are examples (and the data representation chosen). Hypothesis is unknown and has to be consistent with the background knowledge and observations.

The system we have designed includes several components as described further. The main idea of the system has been inspired by the domain theory described in (Adriaans, 1992; Adriaans, 1991). This theory provides a framework for syntactic and semantic language learning under the assumption of compositionality. It allows the learning algorithm to deal with partial information on various levels of language description.

3.1. Data representation

In the context of the LLL-Challenge, learning is achieved by operations on so-called lss-structures:

```
llss_structure := llss(WordId,Word,Lemma,
SemanticType,SyntacticType)
```

In the rest of this paper we will use pseudo-Prolog notation to illustrate the basic concepts. A fundamental operation on lss-structures is lss-unification:

```
lss_univ(llss(_,Word,Lemma,Sem,Syn),
         llss(_,Word,Lemma,Sem,Syn),1):-
         \+Word = unknown,!.
lss_univ(llss(_,_,Lemma,Sem,Syn),
         llss(_,_,Lemma,Sem,Syn),0.75):-
         \+Lemma = unknown,!.
lss_univ(llss(_,_,_,Sem,Syn),
         llss(_,_,_,Sem,Syn),0.5):-
         \+ Sem = unknown,!.
lss_univ(llss(_,_,_,_,Syn),
         llss(_,_,_,_,Syn),0.25):-
         \+ Syn = unknown,!.
lss_univ(llss(_,_,_,_,_),llss(_,_,_,_,_),0):-!.
```

The lemmas and the syntactic types are derived from the annotation provided by the Challenge organizers. The semantic information was obtained from a limited domain-specific ontology that was developed in order to classify concepts encountered in the Challenge (OWL was imported to Prolog using a tool called Thea). Essential concepts such as "BioMolecularEntity" and "BiologicalProcess" were used as semantic tags for words. Classification of text instances such as those in a "named-entity dictionary" (provided by the organizers) was done by hand. A list of verbs was also classified as "DirectedActionVerb" and "InteractionVerb". In an effort to model the homomorphism between semantic and syntactic objects and thereby expose the semantics of syntactic operations, we included a small syntactic ontology in our system.

Lss-structures distinguish between levels of unification. The unification level is 1 when the words and all the other elements unify, 0.75 if the the words do not unify but the rest of the terms do unify, etc. The Prolog code above illustrates the idea. Along the same lines one can define partial unification be-

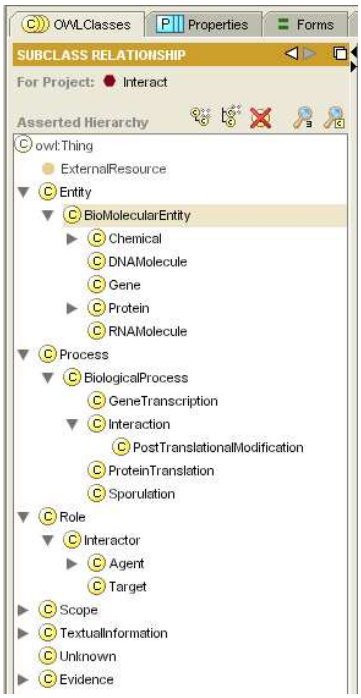


Figure 1. A high-level view of the mini-ontology.

tween two sets of lss-structures. For two sets A and B, `lss_univ(A,B)` is the maximal sum of lss-unifications between elements of unique pairs from $A \times B$, normalized for the cardinality of the biggest set. The value is 1 if A and B are the same and [0,1) otherwise. Syntactic dependency relations are represented in the form of lss-links:

```
lss_link(llss(_,Word1,Lemma1,Sem1,Syn1),
    SyntacticRelation,
        llss(_,Word2,Lemma2,Sem2,Syn2))
```

The definitions for lss-unification can easily be extended to unification for lss-link sets that define partial tree structures.

As input, the system receives sentences accompanied by their syntactic analysis in the form of dependency trees coded as lists of lss-links. The training examples have additional agent-target information. For the test examples this information has to be predicted. The learning process performed on the training set continues along the following lines:

1. For each pair of valid agent-target interactions α and β in a sentence t in the training set we create a proto-rule Γ consisting of the lss-link set related to t, together with the information about the interaction between α and β.
2. For each proto rule Γ predicting interaction between α and β, and each combination of possible targets and agents, γ and δ, in each sentence t of the training set, a *unification signature* is created. The unification signature is a list of unification values for specific parts of lss-link trees related to the location of α, β, γ and δ in the two trees. It also contains the score for the proto rule Γ on t: valid if the interaction between γ and δ is the same as the one between α and β, invalid otherwise.
3. A rule induction program (RIPPER) is applied to the unification signatures to identify useful combinations of proto rules. The resulting set of rules is a set of classifiers that can be applied to the test set.

A unification signature between two lss-link trees is a sequence of real values which reflect weights for the unification chosen by an expert. In case there is no unification, the system outputs 0. Based on the unification results, it is possible to apply different machine learning methods, in our case it is a rule induction.

Taken into account that the training set has not explicitly included negative examples, we have followed the closed world assumption. This way the cartesian product of all agents and targets has been computed over each sentence resulting in 873 combinations, 161 of which are positive examples. The proto-rules have been composed based on these positive examples from the training set. Cartesian product over potential agents and targets for the test set resulted in 568 combinations.

Unification which has been employed in this approach can be thought of in two dimensions. The first dimension presents unification to be carried out on a word and is defined on three levels, in particular, on the lexical, semantic and part of speech levels. While computing the scores we have taken into account the number of levels the unification proceeds on.

On the other hand, the unification can also take place on each level of the syntactic tree. In this case, tree levels are chosen by an expert.

Since it is nearly impossible to perform unification on the complete trees, it has been decided to consider partial trees. From this point of view, the most important parts are roots, ancestors for agent and target, bottom common ancestor for agent and target, and the parent and children for both:

```
rule(ID:A:T:Value,DUA,DDA,DUT,DDT,AncA,AncT,
     CAnc,BCanc,RootA,RSynA,RootT,RSynT)
```

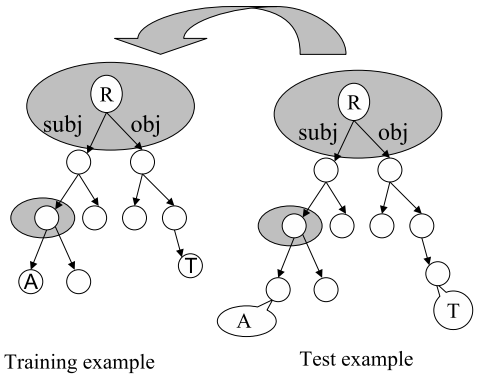


Figure 2. Unification on syntactic trees.

where ID,A,T,$Value$ are sentence identifier, agent, target, and value (true if an interaction is present, false otherwise). In case of the test example, $Value$ equals unknown. DUA,DDA,$AncA$ (DUT,DDT,$AncT$) are a parent, children and all ancestors for a given agent (target), whereas $CAnc$ ($BCanc$) is a list of common ancestors (bottom common ancestor). The values for $RootA$,$RootT$,$RSynA$,$RSynT$ reflect the unification on the root level (the choice of two roots, for an agent and a target has been made assuming the existence of partial syntactic analysis).

For instance, the unification depicted on Fig.2 succeeds on the root level and on the parent level for an agent. Note that it has not been shown how well it succeeds on each of them.

3.2. Classification of interactions

After the unification has been carried out, it is necessary to perform classification. Since we have used weights while performing unification, the result is a sequence of real numbers. Privided that each test example has been unified on all proto-rules, it becomes necessary to reduce the output to a single sequence which has been done by averaging and normalizing. Some of the sequences can be safely eliminated before applying any classification method. They include such unification results for a given test example which have the same scores for each classification level considering two combinations, agent-target and target-agent. Having the same scores for both directions presupposes the potential agent and target to be siblings, which would be unlikely if they were true agent and target.

In order to classify potential interactions into negative and positive interactions, the Weka (Witten & Frank, 2000) implementation of the Ripper rule learner was used. It allows us to construct propositional rules

Table 1. Classification results for Ripper classifier on training and test data sets.

Data set	Precision	Recall	F-Measure
Training			
Positive	71	55.3	62.2
Negative	90.5	94.9	92.7
Test			
Positive	39.2	26.5	31.6

based on the result of unification. This learner has first been tested on the training set making use of 10fold cross validation. The results for both classes are presented in Table 1. After fixing a bug that was present at the time of official testing we found that our results improved from 51,8% (precision), 16,8% (recall) and 25,4% (F-score) (official score) to 39,2%, 26,5% and 31,6% respectively. Please note that the results for the test data have been achieved testing our approach using the full data set (i.e., including coreferences). The precision and recall for the positive and negative examples in the training set have been calculated by us whereas the precision and recall for the test data (only positive examples) have been obtained by using score computation program provided by the organizers. The way of computing scores for the training and test sets slightly differs since for the test data the concept of spurious combinations has been introduced. Spurious combinations are only the elements of negative combinations set and they weren't explicitly marked in the training set.

Using rule induction has several advantages, including the ability to test which levels of unification are the most important for recognizing agents and targets. For example, it has turned out that it is not necessary to use all ancestors for an agent and a target (adding this information usually increases noise), it is sufficient to consider bottom common ancestors. The recognized interactions are of different types, as indicated in Table 2. One of the advantages of our approach lies in the ability to find interactions in the sentences with coreferences. Learning such interactions is considerably more difficult task in comparison to the interactions in the sentences without co-reference or ellipsis.

The lss-unification proved to be useful when comparing to the learning from examples without unification. In this case, the F-score dropped to 42,1% on the training set (for the class of positive interactions). We have also tested the usefulness of using ontology as a background knowledge. As it has turned out, it helped us

Table 2. Distribution of the interaction types .

Co-references	Interaction types		
	Action	Binding	Regulon
Yes	8	2	1
No	5	5	1
Total	13	7	2

to overcome the data sparseness, allowing for the unification on the semantic level even though the actual words or lemmas could be different.

There have been several other learners tested on the Challenge data. In this case, we used the same representation as for the proto-rules (e.g., information about the parents, children, etc.). One of such ILP systems was Aleph (Muggleton, S. H., 1994). However, it didn't provide better results. While increasing precision, the recall dropped significantly since only few interactions (around 10, depending on the settings) have been found. Thus, it would be interesting to test other representation including more information on the path from the agent to the target.

4. Discussion

In this paper, we present an approach that incorporates not only the underlying syntactic structure of the data but also accounts for its semantics. The use of an ontology for semantic annotation enables us to disclose domain knowledge that might be useful for rule induction as well as reason about the rules that have been induced. It also gives us the opportunity to employ unification based on the structure of the ontology, by setting restrictions on the level of unification using the knowledge from the ontology (e.g., unification of elements having the same parent in the hierarchy).

Our ontology was initially simple in order to test ideas within the limited scope of the Challenge. However, a more refined model of the biological domain would strengthen our approach. For instance, we could take into account that not all interactions are of the agent/target type, or that an interaction is sometimes with some aspect of a gene rather than with the gene itself (e.g. with 'localisation of gene X'). We could also model different viewpoints used by biologists, such as the physical (molecular) viewpoint and the genetic one (Demerec, M. 1966). In general, our approach allows one to study the effect of various ontology design decisions.

There are several additional issues to be studied in more detail in the future. First, we would like to augment the current manual semantic tagging with more classification rules for domain-specific concepts, such as interaction verbs and biological processes (e.g. post-translational modification). The additional semantic classification rules could then be applied during a second pass of semantic tagging, in order to enrich the annotation that enables our rules to classify interaction. Additionally, we would like to capture interaction that is stated indirectly, a form of 'undisclosed knowledge' (Swanson, D. R. 1986). There are two types of indirect interaction that we would like to capture: transitive interaction and collective interaction. Presumably, transitive interaction will be captured by forward chaining on a knowledgebase of interactions. However, the indirect and nondeterministic nature of such interactions calls into question the exclusive use of first-order logic; perhaps a probabilistic network model could be useful for this purpose. Such a network model would fit into our plans for a more sophisticated classification mechanism, where the results from several learners could be used to determine confidence in interactions. The second type of indirect interaction, collective interaction, is interaction with an entire class of entities. For instance, if we know that Protein A affects sporulation in general and that Protein B and C are produced during sporulation, we could conclude that Protein A affects Protein B and C. A mechanism to represent collective interaction could be combined with biological annotation that has been mapped into our ontological terms in order to effectively carry out the logic contained in the above example. More specifically, we could add a sporulation property to each protein whose functional category annotation includes sporulation, where sporulation is defined in our ontology.

Another interesting issue is related to interactions that have qualifying phrases such as ".. in the mother cell during late stage sporulation". In this case, the interaction is constrained to a certain location and during a certain stage of development. The constraints in this particular example call for a set of locations and developmental stages to be taken up in the ontology, as well as a way to represent the qualifying information in the semantic tagging system.

From a molecular biology point of view molecular interactions form the basis of many phenomena and are essential to their description. This makes interaction extraction a good starting point in the larger endeavor of knowledge capture from biological documents. Of course, we would eventually like to use interactions alongside other types of extracted knowledge to build and extend knowledge models and ontologies. Sabou et al (Sabou, M. 2005) provide an example of building

a specialized ontology from a small corpus of web service descriptions. A similar approach could be potentially useful in e-science, where there is often a general shortage of resources for the annotation of experimental data.

Acknowledgment

This work was carried out in the context of the Virtual Laboratory for e-Science project (www.vl-e.nl). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

References

Adriaans, P. W. (1992). *Language Learning from a Categorial Perspective*. Doctoral dissertation, Proefschrift.

Adriaans, P. W. (1991). *A Domain Theory for Categorial Language Learning Algorithms*. In Proceedings of the Eighth Amsterdam Colloguium, Amsterdam, The Netherlands. Editors Dekker, Paul and Stokhof, Martin. Institute for Logic, Language and Computation, University of Amsterdam.

Chen, H., & B. M. Sharp. (2004). *Content-rich biological network constructed by mining PubMed abstracts*. BMC Bioinformatics 5: 147.

Demerec, M., E. A. Adelberg, A. J. Clark, & P. E. Hartman. (1996). *A proposal for a uniform nomenclature in bacterial genetics*. Genetics 54: 61-76.

Krallinger, M., R. A. Erhardt, & A. Valencia. (2005). *Text-mining approaches in molecular biology and biomedicine*. Drug Discov Today 10: 439-445.

Muggleton, S. H. & L. De Raedt. (1994). *Inductive Logic Programming: Theory and Methods*. Logic Programming Journal, 19-20: 629-679.

Muller, H. M., E. E. Kenny, & P. W. Sternberg. (2004). *Textpresso: an ontology-based information retrieval and extraction system for biological literature*. PLoS Biol 2: e309.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence. A Modern Approach. Second Edition*. New Jersey: Prentice Hall.

Sabou M., Wroe C., Goble C., & Mishne G. (2005). *Learning Domain Ontologies for Web Service Descriptions: an Experiment in Bioinformatics*. In Proceeedings of the 14th International World Wide Web Conference (WWW2005), Chiba, Japan, 10-14 May, 2005.

Swanson, D. R. (1986). *Fish oil, Raynaud's syndrome, and undiscovered public knowledge*. Perspect Biol Med 30: 7-18.

Uetz, P., and R. L. Finley, Jr. (2005). *From protein networks to biological systems*. FEBS Lett 579: 1821-1827.

Witten, I. H., & Frank, E. (2000). *Data Mining: Practical machine learning tools with Java implementations*. San Francisco: Morgan Kaufmann.

Xenarios, I., and D. Eisenberg. (2001). *Protein interaction databases*. Current Opinion in Biotechnology 12: 334-339.

Yandell, M. D., and W. H. Majoros. (2002). *Genomics and natural language processing*. Nat Rev Genet 3: 601-610.

Learning genic interactions without expert domain knowledge: Comparison of different ILP algorithms

Luboš Popelínský POPEL@FI.MUNI.CZ
Jan Blaťák XBLATAK@FI.MUNI.CZ
Knowledge Discovery Lab, Faculty of Informatics, Masaryk University in Brno, Czech Republic

Abstract

A novel two-step method is proposed in which rules for processing simple examples (sentences that contain a single pair of terms from the dictionary) are learned separately. We also describe an extension of domain knowledge when no domain expert is available, We show that this two-step method performs better on the test set than other learning algorithms.

1. Introduction

In the information extraction task, domain knowledge given by an expert plays a significant role. Especially in mining genic interactions from text, a performance of the learning algorithm is going to increase when knowing e.g. impossible combinations of an agent and a target. In this paper we focus on the situation when no expert domain knowledge is available. We extended the domain knowledge of the LLL challenge task only with predicates for natural language processing – part-of-speech tags, hyperonyma, and the predicate for finding position of a verb between two terms from the domain dictionary. Additionally, we learn syntactic relations as first-order frequent patterns and use them as new predicates.

We propose two-step learning method in which two theories – one for simple examples, another for all examples – are learned separately. We compare results obtained with the domain knowledge without the patterns and with the patterns with performance of class association rules (Liu et al., 1998). For comparison, we also bring results of learning from the data after propositionalization.

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

2. Domain knowledge

Each sentence has been first morphologically tagged with Brill tagger (Brill, 1992). WordNet (WN) has been employed for finding semantic information, actually hyperonyma of the words. A predicate `ffverb` has been added that returns, for a pair of terms from the dictionary, a verb that appears between them. We also removed the `lemma` predicate because it almost never appeared in the learned rules, and the `word` predicate. It resulted in speed up of learning without any decrease of accuracy.

3. Learning algorithms

3.1. Aleph

We employed Aleph[1] and for each example learned its generalization (`induce_max` command). We used default settings except `clauselength=5` (an upper bound on the clause length), `minpos=2` (a lower bound on the number of positive examples covered by the clause), `evalfn=entropy` (Clause utility is p log p + (1-p) log (1-p) where p = P/(P + N) and P, N are the number of positive and negative examples covered) and `nodes=100000` (an upper bound on the nodes to be explored when searching for an acceptable clause).

We learned two kinds of rules, ones that for a given sentence return a pair of agent-target – called here *positive rules* – and also *disambiguation rules* that from all possible pairs in the given sentence aim at removing such pairs that are incorrect.

3.2. RAP

RAP (Blatak et al., 2003; Blatak et al., 2004) was used for finding new predicates as frequent Datalog queries. RAP generates frequent patterns by heuristic or random search what results in a faster acquisition of long patterns than the breadth first search.

[1] `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`

In this work, RAP learned frequent syntactic patterns. Only information contained in the `relation` predicate was exploited, together with `ffverb` and `follows(Word1,Word2)`[2]. The minimal support was 10% and patterns up to the length of 15 literals were generated with best-first search (entropy based heuristics that prefer emerging patterns[3]). A candidate pattern longer than 4 literals that was θ-subsumed by some of the frequent patterns found was removed.

Similarly as in the case of Aleph, both positive and negative rules were learned.

3.3. Propositionalization

All the frequent patterns generated with RAP – together 536 patterns – were transformed into boolean features. For processing this data we used J4.8 decison tree learner, Naive Bayes classifier and instance-based learner IB1 from Weka (Witten, Frank, 1999).

4. Finding genic interactions

The method combines positive and disambiguation rules by the following way. Given four parameters, POSRULES, MINPOS, DISRULES and MINNEG, a pair of two terms, A1 and A2, from the dictionary is a valid genic interaction pair (Agent,Target), if

1. at least POSRULES rules have fired, or
2. a singe rule has fired that covered at least MINPOS positive examples from the learning set, and
3. there is no (A2,A1) after application of all the positive rules.

If there are still unresolved pairs of terms, apply disambiguation rules. For all possible pairs of terms in the sentence remove a pair (A1,A2) if

1. at least DISRULES rules have fired, or
2. a singe rule has fired that covered at least MINNEG negative examples from the learning set.

To summarize, positive rules are applied to test examples first. Consequently, disambiguation rules are employed to remove the remaining ambiguities.

[2] `follows(S,X,Y)` succeeds if the word `X` appeared later in the sentence `S` than the word `Y`.

[3] A pattern is emerging if the coverage on positive and on the negative examples differ significantly.

5. Summary of results

The best result was received for two-step method (see Table 1, AL1). Precision (PRE) as well as recall (REC) and F-measure (F-M) was always higher than 40%.

Table 1. Overview of results

		PRE	REC	F-M
AL2	Aleph, 2-step method	46.5	50.0	48.2
AFP	Aleph + freq.patterns	37.6	64.8	47.6
AL1	Aleph,no freq.patterns	42.5	42.5	42.5
CAR	class association rules	37.2	29.6	32.9
PRO	propositionalization	28.0	29.6	28.8

In the case of AL2, only positive rules were learned. We first selected the sentences that contained only one couple of agent-target, and learned additional rules from this learning set. Then these rules have been added to the rules learned from whole learning set.

6. Discussion of results

6.1. Aleph

Influence of different setting of POSRULES (DISRULES) – the lower limit for the number of positive(disambiguation) rules that cover the example, and MINPOS (MINNEG) – the lower limit for coverage of a positive (disambiguation) rule – for two-step learning is in Table 2. Table 3 displays results for single-step learning. It need to be stressed that for all four learning methods – AL2, AFP, AL1 and CAR – the application of disambiguation rules resulted in increase of F-measure. The last line of Table 2 also displays the result submitted to the challenge.

Table 2. Two-step learning: top 5 results

MINPOS	POSRUL	MINNEG	DISRUL	F-M
5	3	3	2	48.2
6	3	3	2	46.7
5	3	2	2	45.7
5	3	0	0	45.6
4	2	0	0	45.1

6.2. Frequent patterns

We also added to the domain knowledge the frequent patterns learned with RAP. Performance of rules learned with Aleph on the test set did not overcome the best result reached (AL2), however, it was higher

Table 3. Single-step learning: top 5 results

MINPOS	POSRUL	MINNEG	DISRUL	F-M
4	2	3	2	42.5
3	2	-	3	41.8
3	2	3	2	41.5
3	2	2	1	40.0
3	2	0	0	38.0

than Al1 if the disambiguation rules were also used. The appearance of the patterns in the rules learned with Aleph was about 10%. One of the interesting patterns that covers 14 positive and 58 negative examples is: *If there are two words, A and B, and there is a word W in* `comp` *relation with B, and B is right to a verb, then A is not an agent for B.*

6.3. Weka

All the frequent patterns learned with RAP have been taken that had non-zero coverage. These boolean features were used for learning with several propositional learners. The results can be found in Table 4.

Table 4. Results with propositionalized data

	PRE	REC	F-M
SVM	28.0	29.6	28.8
Decision tree	35.4	20.3	25.8
Naive Bayes	22.5	16.6	19.1
IB1	16.4	22.2	18.8

6.4. Domain knowledge

We also checked whether the new predicates – part-of-speech tags (`tag`), hyperonyma (`hyper`), `ffverb` – really help to improve the result. When the part-of-speech tags set by Brill tagger were removed, the F-measure decrease in about 10. Similar situation appears for `ffverb`. The withdrawal of `hyper` has smaller effect. To show how useful the new predicates are, we also checked appearance of these predicates in all the learned theories. `tag` and `ffverb` appeared in each third rule (32.4% and 34.3% respectively). `hyper` appeared in average in 15.6% of rules.

6.5. Maximizing precision

For the single-step learning with Aleph we also explored the possibility of increasing precision and preserving the number of the found pairs high. For the minimum number of correctly recognized agent-target pairs (COR) higher than 15, the highest precision reached 62.9%. The use of disambiguation rules resulted in additional increase of precision but with rapid decrease of correctly recognized agent-target pairs. The best results (PRE $\geq$ 60%) for positive rules are in Table 5.

Table 5. Maximizing precision

MINPOS	POSRUL	COR	PRE
6	5	17	62.9
6	4	17	60.7
7	4	17	60.7
7	3	18	60.0

7. Concluding remarks

We showed that two-step learning method outperformed other ILP approaches. However, neither precision nor recall are high enough for automatic extraction of genic interactions from medical text. Additional increase of precision cannot be reached without employing domain-oriented knowledge, e.g. Gene Ontology (`http://www.geneontology.org/`).

Acknowledgments

We thank to Peter Krutý for his help in the initial phase of this work. Authors have been partially supported by the internal grant of FI MU in Brno.

References

WordNet, a lexical database for the English language. `http://www.cogsci.princeton.edu/~wn/`

J. Blaťák, L. Popelínský, and M. Nepil. Feature construction with RAP. In *ILP'03 Works in Progress*, pages 1–10, 2003.

J. Blaťák, L. Popelínský Mining first-order maximal frequent patterns. Neural Network World 5, 4, pp. 381-390.

Brill, E., A simple rule-based part of speech tagger. Third Conference on Applied Natural Language Processing, Trento 1992.

Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. Knowledge Discovery and Data Mining, pages 80–86, 1998.

Witten, I. H., Frank, E. Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufman, 1999

Learning to Extract Genic Interactions Using Gleaner

Mark Goadrich RICHM@CS.WISC.EDU
Louis Oliphant OLIPHANT@CS.WISC.EDU
Jude Shavlik SHAVLIK@CS.WISC.EDU
Department of Biostatistics and Medical Informatics and Department of Computer Sciences, University of Wisconsin-Madison, 1300 University Avenue, Madison, WI, 53706 USA

Abstract

We explore here the application of Gleaner, an Inductive Logic Programming approach to learning in highly-skewed domains, to the Learning Language in Logic 2005 biomedical information-extraction challenge task. We create and describe a large number of background knowledge predicates suited for this task. We find that Gleaner outperforms standard Aleph theories with respect to recall and that additional linguistic background knowledge improves recall.

1. Introduction

Information Extraction (IE) is the process of scanning unstructured text for objects of interest and facts about these objects. Recently, biomedical journal articles have been a major source of interest in the IE community for a number of reasons: the amount of data available is enormous; the objects, proteins and genes, do not have standard naming conventions; and there is interest from biomedical practitioners to quickly find relevant information (Blaschke et al., 2002, Shatkay & Feldman, 2003, Eliassi-Rad & Shavlik, 2001, Ray & Craven, 2001, Bunescu et al., 2004).

IE can be framed as a machine learning task: given information in unstructured text documents, extract the relevant objects and relationships between them. We believe that Inductive Logic Programming (ILP) is well-suited for IE in biomedical domains. ILP offers the advantages of (1) a straight-forward way to incorporate domain knowledge and expert advice and (2) produces logical clauses suitable for analysis and revision by humans to improve performance.

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

In this article, we report both the data-preparation techniques and the results of applying Gleaner (Goadrich et al., 2004) to the Learning Language in Logic 2005 biomedical information extraction task of learning genic interactions. Gleaner is a two-stage ILP algorithm that (1) learns a broad spectrum of clauses and (2) then combines them into a thresholded disjunctive clause aimed at maximizing precision for a particular choice of recall. We compare our results to standard Aleph (Srinivasan, 2003) using recall and precision, and discuss areas open to future research.

2. Data Preparation

Our dataset for this article is the Learning Language in Logic challenge task[1], where the goal is to learn to recognize the interaction in English sentences between protein agents and their gene targets in *Bacillus subtilis*. Sentences in the training set contained either a direct reference between an agent and a target, such as "GerE stimulates cotD transcription," or an indirect reference, such as "GerE binds to a site on one of these promoters, cotX [...]," where the relation between GerE and cotX is mediated by the phrase "these promoters." The organizers call these two subtasks *without co-reference* and *with co-reference* and we chose to learn on them separately, first learning only relationships without co-reference, and second learning only relationships with co-reference.

The training data consist of 80 sentences found in the Medline[2] database, and contain 106 relations without co-reference and 59 relations with co-reference. For each subtask, we used the other trainset as our tuneset to find an appropriate threshold for making testset predictions. While they are slightly different tasks, we found that the benefit of more examples outweighed dividing the training sets into subfolds.

[1] http://genome.jouy.inra.fr/texte/LLLchallenge/
[2] http://www.ncbi.nlm.nih.gov/pubmed

2.1. Example Filtering

Positive examples for this dataset, consisting of word/word pairings, have been labeled by the challenge-task committee, while negative examples were left up to the participants. We define negative examples on a per-sentence basis by first finding all words which participate in a positive relationship. The pairings among these words which are not labeled as positives are used as negatives for training and tuning. This produced 414 without co-reference negative examples and 261 with co-reference negative examples.

The testset was provided to us unlabeled, and contained sentences for both the task with co-reference and the task without co-reference. Unlike the training data, the testset also contained sentences which did not contain any relations. For the testset, we created examples from the pairing of all possible protein and gene names found in a provided dictionary. This produced 936 total testset examples. In subsequent experiments, we reduced this to 618 examples by removing testset examples where the agent and target of the relation were identical (since this never happened in the trainset). Ultimately there were 54 positive and 410 negative test examples for the without co-reference task and 29 positive and 384 negative test examples for the with co-reference task.

2.2. Background Knowledge

To prepare the data for learning via Inductive Logic Programming, we constructed a variety of background knowledge from sentence structure, statistical word frequencies, lexical properties, and biomedical dictionaries, examples of which can be seen in Table 1.

Our first set of relations comes from the sentence structure. We use the Brill tagger (1995) retrained on the GENIA dataset (Kim et al., 2003) to predict the part of speech for each word. Then we employ a shallow parser created by Burr Settles that uses Conditional Random Fields (Lafferty et al., 2001) trained on a standard corpus (Sang, 2001) to derive a flat parse tree, such that there are no nested phrases, for all sentences in our dataset. All phrases have the sentence as the root, and therefore all words are only members of one phrase. Figure 1 shows a sample sentence parse divided into one level of phrases.

Each word, phrase, and sentence is given a unique identifier based on its ordering within the given abstract, such as `ab11011148_sen1_ph2_w1`. This allows us to create relations between sentences, phrases and words based not on the actual text of the document but on its structure, such as `sentence_child`, `phrase_previous` and `word_next` about the tree structure and sequence of words, and predicates like `nounPhrase`, `article`, and `verb` to describe the part of speech structure. To include the actual text of the sentence in our background knowledge, the predicate `word_ID_to_string` maps these identifiers to the words. In addition, the words of the sentence are stemmed using the Porter stemmer (Porter, 1980), and currently we only use the stemmed version of words.

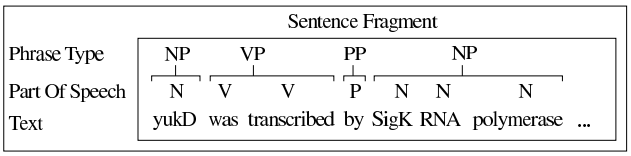


Figure 1. Sample Sentence Parse (N=noun, V=verb, P=preposition, NP=noun phrase, VP=verb phrase, PP=prepositional phrase)

General sentence-structure predicates like `word_before` and `phrase_after` are added, allowing navigation around the parse tree. Phrases are also tagged as being the first or last phrase in the sentence, likewise for words. The length of phrases is calculated and explicitly turned into a predicate, as well as the length (by words and phrases) of sentences. Also, phrases are classified as short, medium or long. An additional piece of useful information is the predicate `different_phrases`, which is true when its two arguments are distinct phrases.

Another group of background relations comes from looking at the frequency of words appearing in the target phrases in the training set. We believe these frequently occurring words could be indicators of some underlying semantic class and will be helpful for identifying correct phrases in the testset. We created Boolean predicates for several ratios - 2 times, 5 times and 10 times the general word frequency across all sentences in a given training set - using the following formula to determine which words matched which ratios:

$$\frac{P(w_i = word | w_i \in Target\ Phrase)}{P(w_i = word | w_i \notin Target\ Phrase)}$$

For example, the words "depend," "bind," and "protein" are at least 5 times more likely to appear in protein phrases than in phrases in general in the without co-reference training set. These gradations are calculated for both target arguments, protein and gene. We automatically create semantic classes consisting of these high frequency words. These semantic classes are then used to mark up all occurrences of these words in a given training and testing set.

A third source of background knowledge is de-

Table 1. Translation from Sample Sentence "ykuD was transcribed by SigK RNA polymerase from T4 of sporulation," to Prolog. This sentence is from the abstract whose PubMed ID is 11011148. Not all predicates created are listed.

Background Knowledge	Some of the Prolog Predicates Created

Sentence Structure	`sentence(ab11011148_sen4).`
	`phrase(ab11011148_sen4_ph0).`
	`phrase(ab11011148_sen4_ph1).`
	`word(ab11011148_sen4_ph0_w0).`
	`word(ab11011148_sen4_ph1_w1).`
	`word(ab11011148_sen4_ph1_w2).`
	`phrase_child(ab11011148_sen4_ph0, ab11011148_sen4_ph0_w0).`
	`word_next(ab11011148_sen4_ph0_w0, ab11011148_sen4_ph0_w1).`
	`word_ID_to_string(ab11011148_sen4_ph1_w1, 'ykuD').`
	`target_arg2_before_target_arg1(ab11011148_sen4).`

Part Of Speech	`np_segment(ab11011148_sen4_ph0).`
	`vp_segment(ab11011148_sen4_ph1).`
	`n(ab11011148_sen4_ph0_w0).`
	`v(ab11011148_sen4_ph1_w1).`
	`prep(ab11011148_sen4_ph1_w3).`

Medical Ontologies	`phrase_contains_mesh_term(ab11011148_sen4_ph3, 'RNA').`

Lexical Properties	`phrase_contains_alphanumeric_word(ab11011148_sen4_ph5).`
	`phrase_contains_specific_word(ab11011148_sen4_ph1, 'transcribed').`
	`phrase_contains_originally_leading_cap(ab11011148_sen4_ph3).`

Word Frequency	`phrase_contains_some_arg_2x_word(ab11011148_sen4_ph3).`

rived from the lexical properties of each word. `Alphanumeric` words contain both numbers and alphabetic characters, (such as "sigma 32" and "Spo0A~P") whereas `alphabetic` words have only alphabetic characters. Other lexical and morphological features include `singleChar` ("a"), `hyphenated` ("membrane-bound") and `capitalized` ("RNA"). Also, words are classified as `novelWord` ("sporulation") if they do not appear in the standard `/usr/dict/words` dictionary in UNIX. Lexical predicates are then augmented to make them more applicable to the phrase level and therefore more general. These predicates are also created for pairs and triplets of words, so we can assert that a phrase has the word "bind" tagged as a verb all in one step when we search the hypothesis space.

For our fourth source, we incorporate semantic knowledge about biology and medicine into our background relations by using the Medical Subject Headings (MeSH)[3]. As we did for the sentence structure, we have simplified this hierarchy to only be one level. Phrases are labeled with the predicate `phrase_contains_mesh_term` if any of the words in the given phrase match any words in MeSH.

[3]http://www.nlm.nih.gov/mesh/meshhome.html

Additionally, predicates are added to denote the ordering between the phrases. `Target_arg1_before_target_arg2` asserts that the protein phrase occurs before the gene phrase, similarly for `target_arg2_before_target_arg1`. Also created are `identical_target_args` (which is true when the protein and gene phrases are the same phrase, such as the phrase "sigmaB dependent katX expression") and `adjacent_target_args` (which says the adjacent phrases contains both the gene and protein), as well as the count of phrases before and after the target arguments. Overall, we defined 215 predicates for use in describing the training examples.

2.3. Enriched Data

Background knowledge was also provided by the challenge task organizers. They processed the corpus with Link Parser (Temperly et al., 1999), a tool for automatically constructing a syntactic parse tree, and refined the output to create two type of additional information. First, each word was assigned its root word, called a *lemma*. For instance, the word "are" would have the lemma "be." The second type of information was the syntactic relations between words. This included appositive, complement, modifier, negation,

Table 2. Pseudo-code for Gleaner Algorithm

Initialize Bins:
Create B recall bins, $bin_{\frac{1}{B}}, bin_{\frac{2}{B}}, ..., bin_1$,
to uniformly divide the recall range [0,1]

Populate Bins:
For $i = 1$ to K until N clauses are generated
  Pick seed example to find bottom clause
  Use Rapid Random Restart to find clauses
  After each generation of a new clause c
    Find the recall bin_r for c on the trainset
    If the *precision* $\times$ *recall* of c is best yet
      Replace c in $bin_{r,i}$

Determine Bin Threshold:
For each bin_j
  Find highest precision theory m and $L_m \in [1, K]$
  on trainset such that
    recall of "At least L of K clauses match
    examples" $\approx$ recall for bin_j

Evaluate On Testset:
Find precision and recall of testset using each
bin's "at least L of K" decision process

object and subject relations about the sentence grammar, as well as predicted parts of speech for each word in a relationship, for a total of 27 possible relations. For example, in the sentence "ykuD was transcribed by SigK RNA polymerase from T4 of sporulation," Link Parser reports that the noun 'yukD' is the subject of the verb 'transcribed', 'polymerase' and 'T4' are complements of 'transcribed', and 'RNA' and 'SigK' are modifiers of 'polymerase'.

We chose to ignore the lemma information, since we previously incorporated the stem of each word, and only focused on the 27 syntactic information predicates. We compare the inclusion versus exclusion of this enriched background information in our results.

3. Gleaner

Gleaner (Goadrich et al., 2004) is a randomized search method which collects good clauses from a broad spectrum of points along the recall dimension in recall-precision curves and employs an "at least N of these M clauses" thresholding method to combine sets of selected clauses. Pseudo-code for our algorithm appears in Table 2.

Gleaner uses Aleph (Srinivasan, 2003) as its underlying engine for generating clauses. As input, Aleph takes

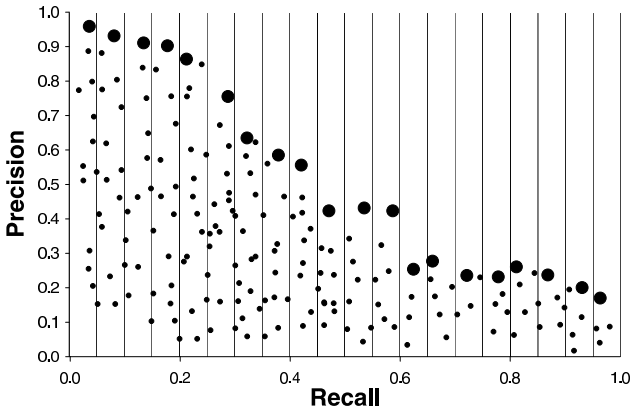


Figure 2. A sample run of Gleaner for one seed and 20 bins, showing each considered clause as a small circle, and the chosen clause per bin as a large circle. This is repeated for 100 seeds to gather 2,000 clauses (assuming a clause is found that falls into each bin for each seed).

background information in the form of either intensional or extensional predicates, a list of modes declaring how these predicates can be chained together, and a designation of one predicate as the "head" predicate to be learned. At a high-level overview, Aleph sequentially generates clauses for the positive examples by picking a random example to be a *seed*. This example is then saturated to create the *bottom clause*, i.e. every relation in the background knowledge that can be connected by relations to this example in a fixed number of steps. The bottom clause determines the possible search space for clauses. Aleph heuristically searches through the space of possible clauses until the "best" clause is found or time runs out. When enough clauses are learned to cover (almost) all of the positive training examples, the learned clauses are combined to form a theory. In our experiments, we will compare Gleaner to standard Aleph theories.

After initialization, the first stage of Gleaner learns a wide spectrum of clauses, as illustrated in Figure 2. We search for clauses using 100 random seed examples to encourage diversity. In our experiments, the recall dimension is uniformly divided into 20 equal sized bins, $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1.00]$. For each seed, we consider up to 25,000 possible clauses using Rapid Random Restart (Železný et al., 2003). As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls. Each bin keeps track of the best clause appearing in its bin for the current seed. We use the heuristic function *precision* $\times$ *recall* to determine the best clause. At the end of this search process, there will be 20 clauses collected for each seed and 100 seed examples for a total of 2,000 clauses (assuming a clause is found that falls

into each bin for each seed).

The second stage (modified slighly from (Goadrich et al., 2004)) takes place once all our clauses have been gathered using random search. Gleaner combines the clauses in each bin to create one large thresholded disjunctive clause per bin, of the form "At least L of these K clauses must cover an example in order to classify it as a positive." Each of these theories could generate their own recall-precision curves, by exploring all possible values for L on the tuneset, starting with $L = K$ and incrementally lowering the threshold to increase recall. These 20 curves will overlap in their recall and precision results, and we choose the theory which created the highest points along this combined curve on the tuneset, irrespective of the bin which generated the points. We will end up with 20 recall-precision points, one for each bin, that span the recall-precision curve.

A unique aspect of Gleaner is that each point in the recall-precision curve could be generated by a separate theory, instead of the usual setup to create a curve, where one hypothesis is transformed into many by ranking the examples and then finding different thresholds of classification. This separate-theory method is related to using the ROC convex hull created from separate classifiers (Fawcett, 2003). We believe using separate theories is a strength of our Gleaner approach, such that each theory, and therefore each point on our curves, is not hindered by the mistakes of previous points; each theory is totally independent of the others.

An end-user of Gleaner will be able to choose their preferred operating point from this recall-precision curve. Our algorithm will then be used to generate testset classifications using the closest bin to their desired recall results by using our found threshold L.

4. Results

There were two dimensions on which to vary our training methods, learning on data containing co-references or on data without co-references, and including the provided linguistic information (enriched) or using only the basic data. Tables 3 and 4 show the results of Gleaner on the testset data for all four combinations, using the restriction that the same word cannot be both agent and target in a relation[4]. A sample clause learned by Aleph can be found in Table 5. This clause has focused on the common property that agents are before targets, agents are nouns with internal capital letters and are complements of nouns which complement verbs, while targets are in noun phrases without negatively correlated words in the training set.

[4]For our challenge-task submission, we used all 936 possible test examples. Using the non-identical restriction resulted in a small increase in our precision results.

Table 3. Results of Gleaner, Aleph theory, and baseline all-positive prediction on LLL challenge task without co-reference.

Alg	Enriched	F1	Recall	Precision
Gleaner	-	41.7	79.6	28.3
	√	25.1	79.6	14.9
Aleph 1K	-	50.0	62.9	40.6
	√	31.0	59.2	21.0
Aleph 25K	-	30.7	44.4	23.5
	√	26.1	42.5	18.8
All Pos	*N/A*	20.1	100.0	11.2

Table 4. Results of Gleaner, Aleph theory, and baseline all-positive prediction on LLL challenge task with co-reference.

Alg	Enriched	F1	Recall	Precision
Gleaner	-	17.7	79.3	10.0
	√	18.5	82.7	10.4
Aleph 1K	-	31.6	51.7	22.7
	√	19.3	37.9	13.0
Aleph 25K	-	19.9	20.6	19.3
	√	19.1	24.1	15.9
All Pos	*N/A*	12.5	100.0	6.7

We chose our preferred operating point by choosing the bin with the highest F1 measure on the tuning set; these were bin [0.55, 0.60] on the basic dataset without co-reference, [0.65, 0.70] on the enriched dataset without co-reference and bin [0.90, 0.95] on the dataset with co-reference. With the enriched data, similar recall points can still be achieved, however there is a marked decrease in precision for the without co-reference dataset. We plan to explore the use of the enriched data from Link Parser (Temperly et al., 1999) in our future work on this and other information-extraction datasets.

We also show a comparison of Gleaner to two other algorithms. First, we examine the results of a single Aleph theory learned for each training set combination. We restrict each clause learned to have a min-

Table 5. Sample Clause with 20% Recall and 94% Precision on Without Co-reference Training Set

```
agent_target(A,T,S) :-
        n(A),
        complement_of_N_N(G,A),
        complement_by_V_PASS_N(G,_),
        word_parent(A,F),
        phrase_contains_some_internal_cap_word(F,_,A),
        word_parent(T,E),
        phrase_contains_no_arg_halfX_word(E,arg2,_),
        isa_np_segment(E),
        target_arg1_before_target_arg2(A,T).
```

where the variable A is the agent, T is the target, S is the sentence, and '_' indicates variables that only appear once in the clause.

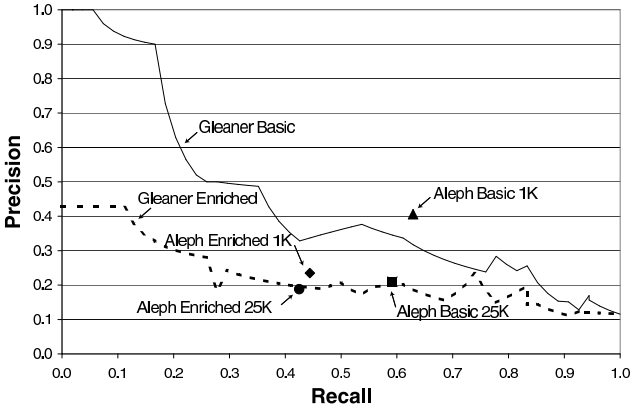


Figure 3. Recall-Precision Curves for Gleaner and Aleph on the dataset without co-reference.

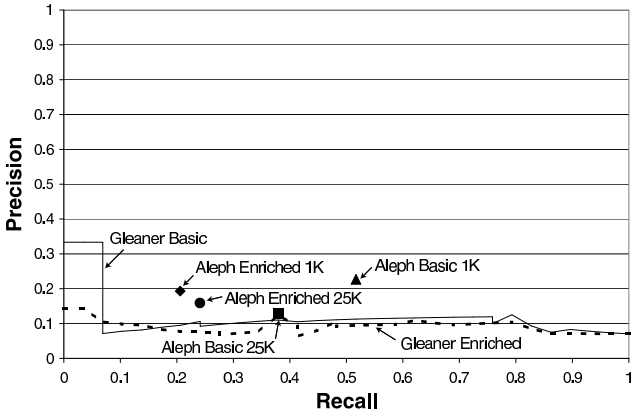


Figure 4. Recall-Precision Curves for Gleaner and Aleph on the dataset with co-reference.

imum precision of 75.0 and to cover a minimum of 5 positives in the training set. We also consider a maximum of both 1,000 and 25,000 clauses for each "best" clause in a theory. With the basic data, we see Aleph improves in precision, however recall is much lower that our results with Gleaner. We also notice a large drop in precision and recall between 1,000 clauses and 25,000 clauses, which we attribute to overfitting. Second, we compare to the algorithm of calling every example positive, which guarantees us 100% recall, and notice that Gleaner has an increase in precision over this baseline in both datasets.

Figure 3 shows recall-precision curves for Gleaner and recall-precision points for the Aleph theories on the dataset without co-reference, while Figure 4 shows results on the dataset with co-reference. Gleaner is able to span the whole recall-precision dimension, although with less than stellar results on the without co-reference dataset.. Gleaner seemed to suffer by not distinguishing well between the agent and target; when genic_interaction(A,B) was predicted, most often we also predicted genic_interaction(B,A), keeping the precision lower than 50%. Another cause of our low results could be the fact that sentences with genes and proteins but no relationships between them were not included in the training sets, but made up almost half of the testing set. This lack of negative sentences in the training sets hampered our ability to distinguish between good and bad sentences when learning clauses. Also, the size of the LLL challenge task was small in comparison to our previous work (Goadrich et al., 2004), creating the possibility of overfitting. Particularly affected were the enriched linguistic predicates and the statistical predicates, which focused on irrelevant words (e.g. specific gene and protein words like "sigma A" and "gerE"). Although collecting labeled

data for biomedical information extraction can be expensive, we believe the benefits are worth the cost.

5. Conclusions

This paper has explored two Inductive Logic Programming approaches to biomedical information extraction: Aleph, which learns many high-precision clauses that cover the training set, and Gleaner, which learns clauses from a wide spectrum of recall points and combines them to create broad thresholded theories. We developed a large number of background knowledge predicates which try to capture both the structure and semantics of biomedical text, and we evaluated these two algorithms on the Learning Language in Logic 2005 Challenge Task.

We believe there is much work remaining in the combination of ILP and biomedical information extraction. The logical structure of sentence parses as well as the biological semantic class information can be readily included in an ILP approach. This genic-interaction dataset was particularly interesting since neither the agent entity nor the target entity was a closed set, and there could be crossover between them. Also worth noting was the difference between the training set and testing set with respect to negative examples. We plan to further explore the issues which arose from using this dataset and perform cross-validation experiments to test for statistical significance of our results and to include negative sentences in the training set.

6. Acknowledgements

We gratefully acknowledge the funding from USA NLM Grant 5T15LM007359-02, USA NLM Grant 1R01LM07050-01, USA DARPA Grant F30602-01-2-0571, and USA Air Force Grant F30602-01-2-0571. Thanks to Burr Settles for help with parsing and tagging the sentences.

References

Blaschke, C., Hirschman, L., & Valencia, A. (2002). Information Extraction in Molecular Biology. *Briefings in Bioinformatics*, *3*, 154–165.

Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*.

Bunescu, R., Ge, R., Kate, R., Marcotte, E., Mooney, R., Ramani, A., & Wong, Y. (2004). Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. *Journal of Artificial Intelligence in Medicine*, 139–155.

Eliassi-Rad, T., & Shavlik, J. (2001). A Theory-Refinement Approach to Information Extraction. *Proceedings of the 18th International Conference on Machine Learning*.

Fawcett, T. (2003). *ROC Graphs: Notes and Practical Considerations for Researchers* (Technical Report). HP Labs HPL-2003-4.

Goadrich, M., Oliphant, L., & Shavlik, J. (2004). Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction. *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*. Porto, Portugal.

Kim, J.-D., Ohta, T., Teteisi, Y., & Tsujii, J. (2003). GENIA corpus - a semantically annotated corpus for bio-textmining. *Bioinformatics*, *19*.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

Porter, M. (1980). An Algorithm for Suffix Stripping. *Program*, *14*, 130–137.

Ray, S., & Craven, M. (2001). Representing Sentence Structure in Hidden Markov Models for Information Extraction. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*.

Sang, E. F. T. K. (2001). Transforming a Chunker into a Parser. *Linguistics in the Netherlands*.

Shatkay, H., & Feldman, R. (2003). Mining the Biomedical Literature in the Genomic Era: An Overview. *Journal of Computational Biology*, *10*, 821–55.

Srinivasan, A. (2003). The Aleph Manual Version 4. *http://web.comlab.ox.ac.uk/ oucl/ research/ areas/ machlearn/ Aleph/*.

Temperly, D., Sleator, D., & Lafferty, J. (1999). An introduction to the Link Grammar Parser. *http://www.link.cs.wisc.edu/link/*.

Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-Search Runtime Distributions may be Heavy-Tailed. *Proceedings of the 12th International Conference on Inductive Logic Programming 2002* (pp. 333–345). Springer Verlag.

Genic Interaction Extraction with Semantic and Syntactic Chains

Sebastian Riedel S.R.RIEDEL@SMS.ED.AC.UK
Ewan Klein EWAN@INF.ED.AC.UK
Institute for Communicating and Collaborative Systems, School of Informatics, The University of Edinburgh, Edinburgh EH8 9LW, Scotland UK

Abstract

This paper describes the system that we submitted to the "Learning Language in Logic" Challenge of extracting directed genic interactions from sentences in Medline abstracts. The system uses Markov Logic, a framework that combines log-linear models and First Order Logic, to create a set of weighted clauses which can classify pairs of gene named entities as genic interactions. These clauses are based on chains of syntactic and semantic relations in the parse or Discourse Representation Structure (DRS) of a sentence, respectively. Our submitted results achieved 52.6% F-Measure on the dataset without and 54.3% on the dataset with coreferences. After adding explicit clauses which model non-interaction we were able to improve these numbers to 68.4% and 64.7%, respectively.

1. Introduction

This paper describes the system that we submitted to the "Learning Language in Logic" (LLL) Challenge of extracting directed genic interactions from sentences in Medline abstracts. As an illustratation of the extraction task, given a sentence like (i), we wish to extract information of the form in (ii):

(i) Localization of SpoIIE was shown to be dependent on the essential cell division protein FtsZ.

(ii) *genic_interaction(FtsZ, SpoIIE)*

The LLL Challenge organizers provided training and test sets containing sentences with and without coreferential expressions. For all data sets, 'enriched' versions were available which contained lemmas and syntactic parses for each sentence.

Our initial goal was to explore whether relation extraction could be effectively carried out over semantic representations, rather than surface strings or parse structures. In particular, we decided to take as our starting point the logical forms produced by ccg2sem (Bos, 2005), a tool whose input is a CCG (Steedman, 2001) dependency tree built by a wide coverage statistical parser (Clark & Curran, 2004) and whose output is a Discourse Representation Structure (DRS) (Kamp & Reyle, 1993). This approach was based on the hope that target relations between entities would be easier to recover once the whole sentence had been converted into a set of semantic relations. Although we still believe this approach has considerable potential, we found that for the LLL Challenge dataset, it was less successful than we had hoped, primarily due to the effect of parser errors and to problems in processing the semantic output that ccg2sem yielded for coordinate structures. However, it turned out that using syntactic information from the enriched data sets improved performance significantly.

We had a strong bias in terms of the clauses to be learned: they had to connect both genes transitively. With this requirement in mind and the observation that our initial attempts to apply ILP systems (FOIL and Aleph) did not yield such clauses, we decided instead to generate a set of clauses based on *chains* of relations between the two genes. Clause candidates that were automatically extracted from the training corpus were fed to a Markov Logic (Domingos & Richardson, 2004) system; this in turn learned probabilistic weights that reflected how often the clause candidates were actually observed in the training data.

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005.

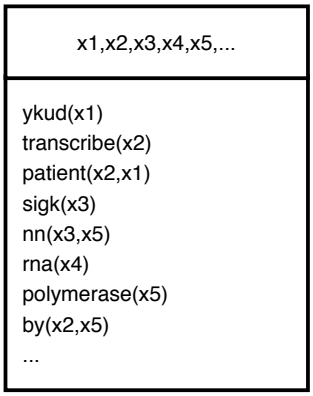


Figure 1. A simple DRS in box representation

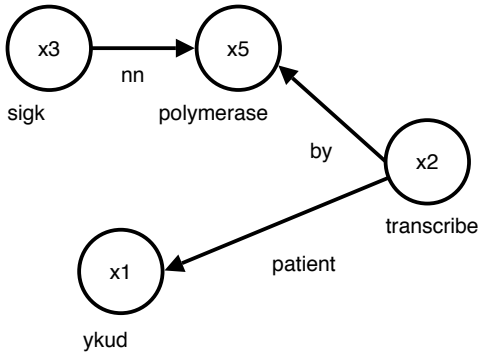


Figure 2. A semantic chain between two discourse referents

2. Data Preprocessing

2.1. Tokenization, Part-of-speech Tagging and Parsing

In order for ccg2sem to create logical forms, the training sentences had to be tokenized, part-of-speech tagged and parsed into a CCG tree. Tokenization was straightforward: the gene terms (retrieved from a list) were mapped into single tokens even when containing whitespace, hyphens or parentheses. part-of-speech tagging and parsing were both handled by the CCG parser(Clark & Curran, 2004).

2.2. DRS Construction

After parsing, the CCG tree was transformed by ccg2sem into a Discourse Representation Structure (DRS), complete with resolved coreferences.

DRSs are defined as ordered pairs of a set of discourse referents and a set of DRS-conditions, where conditions can be n-ary relations and equations over discourse referents (DRs), together with implications, disjunctions and negations of sub-DRSs. For a sentence like

> ykuD was transcribed by SigK RNA polymerase from T4 of sporulation

ccg2sem produced, among others, a discourse referent $x1$ referring to the ykuD gene, $x2$ which refers to a transcription event and $x3$, $x4$ and $x5$ which are associated with *SigK RNA polymerase*. Figure 1 shows a graphical representation of the DRS the upper section of the box contains the discourse referents, and the main box contains the conditions.[1]

[1]Note that *nn* indicates an underspecified compound noun relation.

2.3. Chains

For every gene pair in a sentence, the shortest semantic chain between the corresponding pair of Discourse Referents (DRs) was extracted. To find a DR given a gene token we used the token-to-DR mapping recovered from the DRS file.

Given two DRs a, b corresponding to gene terms, we define a *semantic chain between a and b*, $sem_{a,b}$, as a sequence of DRs $dr_1 = a, dr_2, \ldots, dr_n = b$ together with a sequence of edges $e_1, e_2, \ldots, e_{n-1}$, where each edge $e_i = (rel_i, dir_i)$ consists of a relation rel_i (in the set of relations produced by ccg2sem) between dr_i and dr_{i+1} and a direction $dir_i \in \{\rightarrow, \leftarrow\}$ which indicates whether $rel_i(dr_i, dr_{i+1})$ or $rel_i(dr_{i+1}, dr_i)$ holds. Figure 2 illustrates a semantic chain extracted from the DRS in Figure 1. As can be seen, the circled nodes correspond to DRs, unary relations label the nodes, and binary relations label the arcs between the nodes. Informally, Figure 2 says that there is a transcription event $x2$, the patient of $x2$ is ykuD, the *by* agent of $x2$ is a polymerase $x5$, and SigK is in an underspecified relation to $x5$.

Syntactic chains were calculated in the same fashion, but based on the parse representations supplied in the enriched LLL Challenge training data rather than on the output of the CCG parser. Given a phrase such as *the essential cell division protein*, the set of parse relations would include the clause `relation('mod_att:N-ADJ',13,10)`, where `13` and `10` are the word indices of *protein* and *essential* respectively. More generally, then, syntactic relations were of the form $relation(rel_i, w_i, w_j)$, where rel_i is one of a fixed set of syntactic relations assigned by the LLL parser. We define a *syntactic chain between word indices i and j*, $syn_{i,j}$, as a sequence of word indices $w_1 = i, w_2, \ldots, w_n = j$ and a sequence of edges $e_1, e_2, \ldots, e_{n-1}$, where each edge $e_i = (rel_i, dir_i)$ con-

sists of a syntactic relation rel_i between w_i and w_{i+1} and a direction $dir_i \in \{\rightarrow, \leftarrow\}$ which indicates whether $relation(rel_i, w_i, w_j)$ or $relation(rel_i, w_j, w_i)$ holds.

3. Machine Learning

Initially we tried to apply Inductive Logic Programming (ILP) directly to the conditions of our DRSs. This yielded clauses which were very dependent on the actual gene names due to the small size of the training set. Moreover, it was not possible to extract clauses which included the generalization that interacting genes must have a connecting semantic chain. This can be attributed to two reasons. First, since the chains in question can be rather long, they would induce correspondingly large clauses; however, the latter are filtered out by the ILP algorithms because they failed to cover enough examples relative to their complexity. Second, the chains can have varying lengths and might match only in small subchains, which makes it difficult for the ILP method to find generalizations.

In the light of this observation we decided to directly extract a set of candidate clauses which capture certain subparts of the semantic chains. Instead of discarding clauses which don't hold in all cases we used Markov Logic[2](Domingos & Richardson, 2004) to attach and learn weights for them. In Markov Logic the weight of a formula (or clause in our case) corresponds to the difference in log-likelihood between a world where the formula holds and a world where it doesn't, all other things being equal.

A *Markov Logic Network* L is a set of formula-weight pairs (F_i, w_i). Together with a set of constants it can be mapped to a log-linear joint probability distribution

$$(1)\quad p_w(X = x) = \frac{1}{Z} exp\left(\sum_{f \in Features_L} w_f f(x)\right)$$

over a sequence $X = (X_1, X_2, \ldots)$ of binary variables, one for each possible ground predicate (with respect to the available constants). The value of such a node is 1 if the ground predicate is true, 0 otherwise. The set of feature functions $Features_L$ is created by adding a feature function f for each possible grounding of each formula F_i in L. A function f returns 1 if its corresponding ground formula is true given x, 0 otherwise. w_f is the weight w_i of the original formula F_i defined in L. Z is a normalisation factor.

[2]See (Riedel & Meza-Ruiz, 2005) for a publicly available implementation.

3.1. Candidate Clause Generation

As mentioned above, we automatically extracted a set of candidate clauses from the given data. All these clauses were based on subchains within the shortest semantic and syntactic chains of the gene pair to be classified. Every time we observed a particular subchain we added a clause that entails $genic\_interaction(A, B)$ or $\neg genic\_interaction(A, B)$ depending on the label of the observed gene pair.

For instance, after seeing a $(agent, \leftarrow), (patient, \rightarrow)$ subchain in a genic interaction chain we generated:

$$genic\_interaction(A, B) :-$$
$$contains(sem_{A,B}, (agent, \leftarrow), (patient, \rightarrow))$$

This clause roughly[3] reads "if gene term A is related to some DR which is the agent of an event and something related to gene term B is the patient of this event, then there is a genic interaction between A and B". Note that 'related' here means reflexively and transitively related with respect to the binary relations in the DRS.

In addition, *typed* subchain clauses are extracted. A typed clause requires certain DRs in the chain to be members of a specified predicate:

$$genic\_interaction(A, B) :-$$
$$contains(sem_{A,B}, (agent, \leftarrow), activate)$$

This clause reads "if something related to gene term A activates something that is related to gene term B, then genic_interaction(A, B) holds" because *activate* is the predicate of the DR followed by the agent relation.

In the case of syntactic chains, clauses look as follows:

$$genic\_interaction(A, B) :-$$
$$contains(syn_{A,B}, (subj, \rightarrow), (comp\_of, \rightarrow))$$

Typed syntactic clauses used words to type the relation members; an example is the following (negative) clause:

$$\neg genic\_interaction(A, B) :-$$
$$contains(syn_{A,B}, (comp\_of, \leftarrow), activity)$$

3.2. Weight Estimation

To find the weights, we maximized the logarithm of the conditional likelihood of the training data

$$(2)\quad \sum_{(x_h, x_o) \in T} log\left(p_w(X_h = x_h | X_o = x_o)\right)$$

[3]We say 'roughly' because one also has to bear in mind that only shortest paths are extracted.

where X_h is a list of possible hidden variables and x_h are the corresponding values in a concrete observation. In this case X_h contains all variables refering to possible grounded *genic_interaction* atoms. X_o is the set of observed variables and corresponds to all possible instantiations of the *contains* predicates. T is the set of all training observations (x_h, x_o). In our current setting, each *genic_interaction* atom only depends on *contains* atoms, which are fully observed. This gives rise to

$$(3)\quad p_w\left(X_h = x_h | X_o = x_o\right) = \prod_{Gene\ pairs(a,b)} p_w\left(X_{g(a,b)} = x_{g(a,b)} | X_{c(a,b)} = x_{c(a,b)}\right)$$

where the variable $X_{g(a,b)}$ corresponds to the ground atom $genic\_interaction(a,b)$, and $X_{c(a,b)}$ is a list of all variables corresponding to *contains* atoms that relate to the syntactic and semantic chains between a and b.

With (3), the conditional in (2) simplifies to

$$(4)\quad \sum_{(x_h,x_o)\in T} \sum_{Gene\ pairs(a,b)} log\left(p_w\left(x_{g(a,b)} | x_{c(a,b)}\right)\right).$$

where $p(x|y)$ is an abbreviation for $p(X = x|Y = y)$

To calculate the conditional in (3), Bayes Rule yields

(5)

$$p_w\left(x_{g(a,b)} | x_{c(a,b)}\right) = \frac{p_w\left(x_{g(a,b)}, x_{c(a,b)}\right)}{p_w\left(1, x_{c(a,b)}\right) + p_w\left(0, x_{c(a,b)}\right)}$$

where the joint probabilities can be calculated using a version of (1) that only contains features of ground clauses related to the gene pair (a,b). This is sound due to the independence of genic interaction labels we are assuming in our clauses.

Using the gradient of (4), we run several iterations of L-BFGS (Liu & Nocedal, 1989), a gradient descent implementation, until convergence.

3.3. Inference

To infer the truth value of an actual genic interaction candidate pair, one could run an inference algorithm such as Belief Propagation or Gibbs Sampling in a Markov network equivalent to (1). However, as all hidden atoms only depend on a set of fully observed atoms this is not necessary; we simply calculate

$$(6)\quad p_g\left(a,b\right) = p_w\left(x_{g(a,b)} | x_{c(a,b)}\right)$$

for each possible gene pair (a,b) using (2). In the case of the gene pair $ykuD, SigK$ in our example in section 2.2 $x_{g(a,b)}$ would correspond to a truth value for the atom

$$genic\_interaction(ykuD, SigK)$$

and $x_{c(a,b)}$ would contain a sequence of truth values for atoms such as

$$contains(sem_{ykuD,SigK}, (patient, \leftarrow), (by, \rightarrow))$$

(true) or

$$contains(syn_{ykuD,SigK}, (comp\_of, \leftarrow), activity)$$

(false).

We classified a pair a, b as participating in a genic interaction if $p_g(a,b) > 0.5$. If both (a,b) and (b,a) are classified as genic interactions we only accept the one with the highest probability. However, in future versions this could be done more soundly using a clause such as

$$\neg genic\_interaction(A,B) :-$$
$$genic\_interaction(B,A)$$

on the assumption that genic interaction is always asymmetric.

Table 1. Clause sets

No.	Clause sets	Neg. clauses
1	syn2-0, syn2-1	yes
2	sem2-0, sem2-1	yes
3	1, 2	yes
4	syn2-0, syn2-1	no
5	3, sem1-1l, sem1-1r	no

4. Results

We tested different sets of clauses as candidates for the weight estimation. Table 1 enumerates the combinations of clause sets we used during the experiments. *sem1-0, sem2-0* and *syn2-0* refer to (untyped) clause sets with semantic or syntactic chains of length one and two, respectively. *sem1-1l, sem1-1r, syn1-1l* and *syn1-1r* represent sets where either the left or right node of all clauses is typed with a predicate or word, respectively. *sem2-1* and *syn2-1* type the middle node of subchains with length two.

4.1. Without Coreferences

Table 2 shows results for the test data without coreferences. Note that the submitted result was clause

Table 2. Results on the test data without coreferences

Clause set No.	Precision	Recall	F-M
1	65.0	72.2	68.4
2	68.5	44.4	53.9
3	64.8	64.8	64.8
4	60.8	51.8	55.9
5	60.9	46.2	52.6

set 5, since this was the clause set that yielded the best results on the cross-validated training set (65% F-Measure). On the test data, it only achieved 52.6% F-measure, as shown. Clause set 5 did not use negative clauses — this functionality was not implemented by the submission date. It combined semantic and syntactic chains which seemed to help only when no negative clauses were added. Finally, it used chains of length one — this only helped during cross-validation, maybe due to the similarity in genic interaction pairs in training and test folds when sentences were drawn from the same abstract.

4.1.1. Syntactic vs. Semantic Chains

Results for the clause sets 1, 2 and 3 in Table 2 show that using syntactic chains alone (i.e., clause set 1) yielded a significantly higher F-Measure compared with using semantic chains or a combination of semantic and syntactic chains.

The main weakness of the semantic approach is its low recall. It can be attributed to two factors. First, incorrect CCG parses were responsible for missing or incorrect semantic chains within the DRSs. Second, ccg2sem generates multiple domain referents for indefinite objects of coordinated expressions.[4] For instance, in the DRS for the sentence

> Most cot genes, and the gerE gene, are transcribed by sigmaK RNA polymerase.

there are two discourse referents for *sigmaK RNA polymerase*, due to the fact that the part-of-speech tagger integrated into the CCG parser failed to label the term as a proper name. Although it might be possible to modify the chain extraction component to deal with this, it would be preferable to change the output of the tagger prior to syntactic analysis.

[4]Semantically, this is indeed the correct result.

Table 3. Results on the test data with coreferences using the clause sets 1, 2 and 5 from Table 1

Clause set No.	Precision	Recall	F-M
1	63.2	66.2	64.7
2	50.0	33.7	40.2
5	55.6	53.0	54.3

4.1.2. Negative clauses

Clause sets 1 and 4 contain the same positive clauses, but clause set 1 benefits from the addition of negative clauses. As shown in Table 2, the performance of clause set 1 is significantly superior to that of clause set 4 in both recall and precision. Having a model of what it means to have no interaction can improve *precision* whenever positives clauses would "vote" for a genic interaction with low confidence, while negative clauses "vote" against a genic interaction with high confidence. Without negative clauses, this scenario would result in a false positive. Negative clauses can increase *recall* when they are highly confident that a gene pair is "not a non-interaction" while positive clauses are uncertain.

4.2. With Coreferences

As we mentioned briefly earlier, an attractive feature of ccg2sem is its built-in coreference resolver. Thus, for testing and training on data with coreferences[5] we expected better results with semantic chains compared to syntactic chains. However, even in this case syntactic chains outperform semantic chains, as shown in Table 3. This can be explained partly by the fact that many coreferences were appositions which the parser could extract. Furthermore, most sentences in the test and training set lacked coreferences, and this also contributed to syntactic chains performing better.

The submitted clause set was 5, since again this was the best performing set under cross-validation on the training data.

5. Conclusion

Perhaps the most striking observation is the dramatic effect of adding negative clauses to the rule base. It seems clear that clauses modeling non-interaction are essential for good performance, increasing both recall and precision as explained above.

[5]This corresponds to the "With and Without Coreferences" data set of the LLL Challenge.

When comparing syntactic and semantic chains, syntactic chains appear to be the clear winner. However, this conclusion has to be tempered by the fact that the syntactic chains were based on manually corrected parses, whereas the semantic chains were based on a completely automatic statistical parser (trained on the Penn Treebank). Moreover, there is also potential in exploiting semantic chains to incorporate domain knowledge in future experiments.

It also turned out that we were unable to apply ILP directly to the problem, due to the small size of the training set and to the varying length and structure of the chains we were looking for. Instead of carefully biasing ILP towards the clauses we had in mind, we decided to extract a set of chain-based clauses and learn probabilistic weights. However, the two approaches are in fact orthogonal — it would make perfect sense to first generate a set of candidate rules using an ILP system and then learn their weights using the Markov Logic approach.

Acknowledgments

Many thanks to Malvina Nissim and Claire Grover for their helpful input during the course of the project.

References

Bos, J. (2005). Towards wide-coverage semantic interpretation. *Proceedings of IWCS-6, Tilburg, The Netherlands*.

Clark, S., & Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. *Proceedings of ACL* (pp. 103–110).

Domingos, P., & Richardson, M. (2004). Markov Logic: A unifying framework for statistical relational learning. *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields, Banff, Canada: IMLS.* (pp. 49–54).

Kamp, H., & Reyle, U. (1993). *From discourse to logic*. Studies in Linguistics and Philosophy. Kluwer Academic.

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Math. Program.*, *45*, 503–528.

Riedel, S., & Meza-Ruiz, I. (2005). Markov The Beast - Markov Logic software platform **url**: `http://homepages.inf.ed.ac.uk/s0349492/thebeast`.

Steedman, M. (2001). *The syntactic process*. The MIT Press.

Author Index