A Memory-Based Approach to Learning Shallow Natural Language Patterns

Shlomo Argamon and **Ido Dagan** and **Yuval Krymolowski**
Department of Mathematics and Computer Science
Bar-Ilan University
52900 Ramat Gan, Israel
{argamon,dagan,yuvalk}@cs.biu.ac.il

Abstract

Recognizing shallow linguistic patterns, such as basic syntactic relationships between words, is a common task in applied natural language and text processing. The common practice for approaching this task is by tedious manual definition of possible pattern structures, often in the form of regular expressions or finite automata. This paper presents a novel memory-based learning method that recognizes shallow patterns in new text based on a bracketed training corpus. The training data are stored as-is, in efficient suffix-tree data structures. Generalization is performed on-line at recognition time by comparing subsequences of the new text to positive and negative evidence in the corpus. This way, no information in the training is lost, as can happen in other learning systems that construct a single generalized model at the time of training. The paper presents experimental results for recognizing noun phrase, subject-verb and verb-object patterns in English. Since the learning approach enables easy porting to new domains, we plan to apply it to syntactic patterns in other languages and to sub-language patterns for information extraction.

1 Introduction

Identifying local patterns of syntactic sequences and relationships is a fundamental task in natural language processing (NLP). Such patterns may correspond to syntactic phrases, like noun phrases, or to pairs of words that participate in a syntactic relationship, like the heads of a verb-object relation. Such patterns have been found useful in various application areas, including information extraction, text summarization, and bilingual alignment. Syntactic patterns are useful also for many basic computational linguistic tasks, such as statistical word similarity and various disambiguation problems.

One approach for detecting syntactic patterns is to obtain a full parse of a sentence and then extract the required patterns. However, obtaining a complete parse tree for a sentence is difficult in many cases, and may not be necessary at all for identifying most instances of local syntactic patterns.

An alternative approach is to avoid the complexity of full parsing and instead to rely only on local information. A variety of methods have been developed within this framework, known as *shallow parsing, chunking, local parsing* etc. (e.g., (Abney, 1991; Greffenstette, 1993)). These works have shown that it is possible to identify most instances of local syntactic patterns by rules that examine only the pattern itself and its nearby context. Often, the rules are applied to sentences that were tagged by part-of-speech (POS) and are phrased by some form of regular expressions or finite state automata.

Manual writing of local syntactic rules has become a common practice for many applications. However, writing rules is often tedious and time consuming. Furthermore, extending the rules to different languages or sub-language domains can require substantial resources and expertise that are often not available. As in many areas of NLP, a *learning* approach is appealing. Surprisingly, though, rather little work has been devoted to learning local syntactic patterns, mostly noun phrases (Ramshaw and Marcus, 1995; Vilain and Day, 1996).

This paper presents a novel general learning approach for recognizing local sequential patterns, that may be perceived as falling within the memory-based learning paradigm. The method utilizes a part-of-speech tagged training corpus in which all instances of the target pattern are marked (bracketed). The training data are stored as-is in suffix-tree data structures, which enable linear time searching for subsequences in the corpus.

The memory-based nature of the presented algorithm stems from its deduction strategy: a new instance of the target pattern is recognized by examining the *raw* training corpus, searching for positive and negative evidence with respect to the given test sequence. No model is created for the training corpus, and the raw examples are not converted to any other representation.

Consider the following example[1]. Suppose we

[1] We use here the POS tags: DT = determiner, ADJ = adjective, ADV = adverb, CONJ = conjunction, VB=verb, PP=preposition, NN = singular noun, and NNP = plural noun.

want to decide whether the candidate sequence

DT ADJ ADJ NN NNP

is a noun phrase (NP) by comparing it to the training corpus. A good match would be if the entire sequence appears as-is several times in the corpus. However, due to data sparseness, an exact match cannot always be expected.

A somewhat weaker match may be obtained if we consider sub-parts of the candidate sequence (called *tiles*). For example, suppose the corpus contains noun phrase instances with the following structures:

(1) DT ADJ ADJ NN NN
(2) DT ADJ NN NNP

The first structure provides positive evidence that the sequence "DT ADJ ADJ NN" is a possible NP prefix while the second structure provides evidence for "ADJ NN NNP" being an NP suffix. Together, these two training instances provide positive evidence that *covers* the entire candidate. Considering evidence for sub-parts of the pattern enables us to generalize over the exact structures that are present in the corpus. Similarly, we also consider the negative evidence for such sub-parts by noting where they occur in the corpus without being a corresponding part of a target instance.

The proposed method, as described in detail in the next section, formalizes this type of reasoning. It searches specialized data structures for both positive and negative evidence for sub-parts of the candidate structure, and considers additional factors such as context and evidence overlap. Section 3 presents experimental results for three target syntactic patterns in English, and Section 4 describes related work.

2 The Algorithm

The input to the Memory-Based Sequence Learning (MBSL) algorithm is a sentence represented as a sequence of POS tags, and its output is a *bracketed sentence*, indicating which subsequences of the sentence are to be considered instances of the target pattern (*target instances*). MBSL determines the bracketing by first considering each subsequence of the sentence as a *candidate* to be a target instance. It computes a score for each candidate by comparing it to the training corpus, which consists of a set of pre-bracketed sentences. The algorithm then finds a consistent bracketing for the input sentence, giving preference to high scoring subsequences. In the remainder of this section we describe the scoring and bracketing methods in more detail.

2.1 Scoring candidates

We first describe the mechanism for scoring an individual candidate. The input is a candidate subsequence, along with its *context*, i.e., the other tags in the input sentence. The method is presented at two levels: a general memory-based learning schema and a particular instantiation of it. Further instantiations of the schema are expected in future work.

2.1.1 The general MBSL schema

The MBSL scoring algorithm works by considering *situated candidates*. A situated candidate is a sentence containing one pair of brackets, indicating a candidate to be a target instance. The portion of the sentence between the brackets is the *candidate* (as above), while the portion before and after the candidate is its *context*. (Although we describe the algorithm here for the general case of unlimited context, for computational reasons our implementation only considers a limited amount of context on either side of the candidate.) This subsection describes how to compute the score of a situated candidate from the training corpus.

The idea of the MBSL scoring algorithm is to construct a tiling of subsequences of a situated candidate which covers the entire candidate. We consider as *tiles* subsequences of the situated candidate which contain a bracket. (We thus consider only tiles within or adjacent to the candidate that also include a candidate boundary.)

Each tile is assigned a score based on its occurrence in the training memory. Since brackets correspond to the boundaries of potential target instances, it is important to consider how the bracket positions in the tile correspond to those in the training memory.

For example, consider the training sentence

[NN] VB [ADJ NN NN] ADV PP [NN] .

We may now examine the occurrence in this sentence of several possible tiles:

VB [ADJ NN occurs positively in the sentence, and

NN NN] ADV also occurs positively, while

NN [NN ADV occurs negatively in the training sentence, since the bracket does not correspond.

The positive evidence for a tile is measured by its *positive count*, the number of times the tile (including brackets) occurs in the training memory with corresponding brackets. Similarly, the negative evidence for a tile is measured by its *negative count*, the number of times that the POS sequence of the tile occurs in the training memory with non-corresponding brackets (either brackets in the training where they do not occur in the tile, or vice versa). The *total count* of a tile is its positive count plus its negative count, that is, the total count of the POS sequence of the tile, regardless of bracket position. The score $f(t)$ of a tile t is a function of its positive and negative counts.

```
Candidate: NN VB [ ADJ NN NN ] ADV
MTile 1:      VB [ ADJ NN NN ]
MTile 2:      VB [ ADJ
MTile 3:         [ ADJ NN
MTile 4:              NN NN ]
MTile 5:                 NN ] ADV
```

Figure 1: A candidate subsequence with some of its context, and 5 matching tiles found in the training corpus.

The overall score of a situated candidate is generally a function of the scores of all the tiles for the candidate, as well as the relations between the tiles' positions. These relations include tile adjacency, overlap between tiles, the amount of context in a tile, and so on.

2.1.2 An instantiation of the MBSL schema

In our instantiation of the MBSL schema, we define the score $f(t)$ of a tile t as the ratio of its positive count $\text{pos}(t)$ and its total count $\text{total}(t)$:

$$f(t) = \begin{cases} 1 & \text{if } \frac{\text{pos}(t)}{\text{total}(t)} > \theta \\ 0 & \text{otherwise} \end{cases}$$

for a predefined threshold θ. Tiles with a score of 1, and so with sufficient positive evidence, are called *matching tiles*.

Each matching tile gives supporting evidence that a part of the candidate can be a part of a target instance. In order to combine this evidence, we try to cover the entire candidate by a set of matching tiles, with no gaps. Such a covering constitutes evidence that the entire candidate is a target instance. For example, consider the matching tiles shown for the candidate in Figure 1. The set of matching tiles 2, 4, and 5 covers the candidate, as does the set of tiles 1 and 5. Also note that tile 1 constitutes a cover on its own.

To make this precise, we first say that a tile T_1 *connects* to a tile T_2 if (i) T_2 starts after T_1 starts, (ii) there is no gap between the end of T_1 and the start of T_2 (there may be some overlap), and (iii) T_2 ends after T_1 (neither tile includes the other). For example, tiles 2 and 4 in the figure connect, while tiles 2 and 5 do not, and neither do tiles 1 and 4 (since tile 1 includes tile 4 as a subsequence).

A *cover* for a situated candidate c is a sequence of matching tiles which collectively cover the entire candidate, including the boundary brackets, and possibly some context, such that each tile connects to the following one. A cover thus provides positive evidence for the entire sequence of tags in the candidate.

The set of all the covers for a candidate summarizes all of the evidence for the candidate being a target instance. We therefore compute the score of a candidate as a function of some statistics of the set of all its covers. For example, if a candidate has many different covers, it is more likely to be a target instance, since many different pieces of evidence can be brought to bear.

We have empirically found several statistics of the cover set to be useful. These include, for each cover, the number of tiles it contains, the total number of context tags it contains, and the number of positions which more than one tile covers (the amount of overlap). We thus compute, for the set of all covers of a candidate c, the

- Total number of different covers, $\mathbf{num}(c)$,
- Minimum number of matches in any cover, $\mathbf{minsize}(c)$,
- Maximum amount of context in any cover, $\mathbf{maxcontext}(c)$, and
- Maximum total overlap between tiles for any cover, $\mathbf{maxoverlap}(c)$.

Each of these items gives an indication regarding the overall strength of the cover-based evidence for the candidate.

The score of the candidate is a linear function of its statistics:

$$\begin{aligned} f(c) \quad = \quad & \alpha\, \mathbf{num}(c) - \beta\, \mathbf{minsize}(c) + \\ & \gamma\, \mathbf{maxcontext}(c) + \\ & \delta\, \mathbf{maxoverlap}(c) \end{aligned}$$

If candidate c has no covers, we set $f(c) = 0$. Note that **minsize** is weighted negatively, since a cover with fewer tiles provides stronger evidence for the candidate.

In the current implementation, the weights were chosen so as to give a lexicographic ordering, preferring first candidates with more covers, then those with covers containing fewer tiles, then those with larger contexts, and finally, when all else is equal, preferring candidates with more overlap between tiles. We plan to investigate in the future a data-driven approach (based on the Winnow algorithm) for optimal selection and weighting of statistical features of the score.

We compute a candidate's statistics efficiently by performing a depth-first traversal of the *cover graph* of the candidate. The cover graph is a directed acyclic graph (DAG) whose nodes represent matching tiles of the candidate, such that an arc exists between nodes n and n', if tile n connects to n'. A special start node is added as the root of the DAG, that connects to all of the nodes (tiles) that contain an open bracket. There is a cover corresponding to each path from the start node to a node (tile) that contains a close bracket. Thus the statistics of all the covers may be efficiently computed by traversing the cover graph.

2.1.3 Summary

Given a candidate sequence and its context (a *situated candidate*):

1. Consider all the subsequences of the situated candidate which include a bracket as *tiles*;
2. Compute a *tile score* as a function of its *positive count* and *total counts*, by searching the training corpus. Determine which tiles are *matching tiles*;
3. Construct the set of all possible *covers* for the candidate, that is, sequences of *connected* matching tiles that cover the entire candidate;
4. Compute the *candidate score* based on the statistics of its covers.

2.2 Searching the training memory

The MBSL scoring algorithm searches the training corpus for each subsequence of the sentence in order to find matching tiles. Implementing this search efficiently is therefore of prime importance. We do so by encoding the training corpus using suffix trees (Edward and McCreight, 1976), which provide string searching in time which is linear in the length of the searched string.

Inspired by Satta (1997), we build two suffix trees for retrieving the positive and total counts for a tile. The first suffix tree holds all pattern instances from the training corpus surrounded by bracket symbols and a fixed amount of context. Searching a given tile (which includes a bracket symbol) in this tree yields the *positive count* for the tile. The second suffix tree holds an unbracketed version of the entire training corpus. This tree is used for searching the POS sequence of a tile, with brackets omitted, yielding the *total count* for the tile (recall that the negative count is the difference between the total and positive counts).

2.3 Selecting candidates

After the above procedure, each situated candidate is assigned a score. In order to select a bracketing for the input sentence, we assume that target instances are non-overlapping (this is usually the case for the types of patterns with which we experimented). We use a simple constraint propagation algorithm that finds the best choice of non-overlapping candidates in an input sentence:

1. Examine each situated candidate c with $f(c) > 0$, in descending order of $f(c)$:
   (a) Add c's brackets to the sentence;
   (b) Remove all situated candidates overlapping with c which have not yet been examined.
2. Return the bracketed sentence.

Train Data:

	sentences	words	patterns
NP	8936	229598	54760
VO	16397	454375	14271
SV	16397	454375	25024

Test Data:

	sentences	words	patterns
NP	2012	51401	12335
VO	1921	53604	1626
SV	1921	53604	3044

Table 1: Sizes of training and test data

Len	NP	%	VO	%	SV	%
1	16959	31				
2	21577	39	3203	22	7613	30
3	10264	19	5922	41	7265	29
4	3630	7	2952	21	3284	13
5	1460	3	1242	9	1697	7
6	521	1	506	4	1112	4
7	199	0	242	2	806	3
8	69	0	119	1	592	2
9	40	0	44	0	446	2
10	18	0	20	0	392	2
>10	23	0	23	0	1917	8
total	54760		14271		25024	
avg. len	2.2		3.4		4.5	

Table 2: Distribution of pattern lengths, total number of patterns and average length in the training data.

3 Evaluation

3.1 The Data

We have tested our algorithm in recognizing three syntactic patterns: noun phrase sequences (NP), verb-object (VO), and subject-verb (SV) relations. The NP patterns were delimited by '[' and ']' symbols at the borders of the phrase. For VO patterns, we have put the starting delimiter before the main verb and the ending delimiter after the object head, thus covering the whole noun phrase comprising the object; for example:

```
... investigators started to
[ view the lower price levels ]
as attractive ...
```

We used a similar policy for SV patterns, defining the start of the pattern at the start of the subject noun phrase and the end at the first verb encountered (not including auxiliaries and modals); for example:

```
... argue that
[ the U.S. should regulate ]
the class ...
```

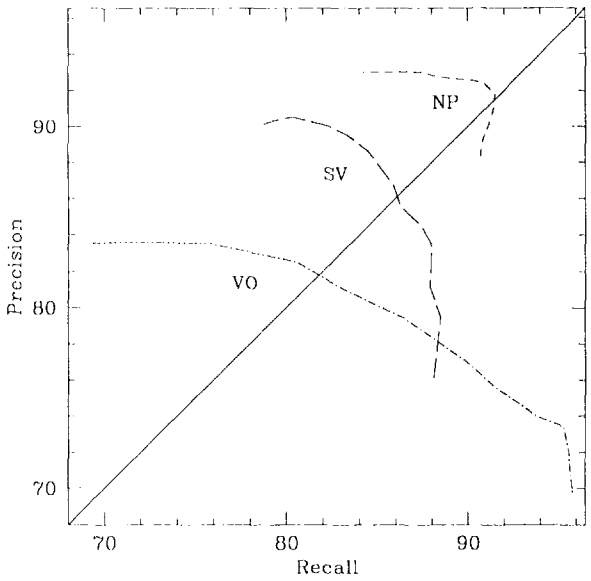

Figure 2: Recall-Precision curves for NP, VO, and SV; $0.1 \leq \theta \leq 0.99$

The subject and object noun-phrase borders were those specified by the annotators, phrases which contain conjunctions or appositives were not further analyzed.

The training and testing data were derived from the Penn TreeBank. We used the NP data prepared by Ramshaw and Marcus (1995), hereafter RM95. The SV and VO data were obtained using T (TreeBank's search script language) scripts.[2] Table 1 summarizes the sizes of the training and test data sets and the number of examples in each.

The T scripts did not attempt to match dependencies over very complex structures, since we are concerned with shallow, or local, patterns. Table 2 shows the distribution of pattern length in the train data. We also did not attempt to extract passive-voice VO relations.

3.2 Testing Methodology

The test procedure has two parameters: (a) maximum context size of a candidate, which limits *what* queries are performed on the memory, and (b) the threshold θ used for establishing a matching tile, which determines *how* to make use of the query results.

Recall and precision figures were obtained for various parameter values. F_β (van Rijsbergen, 1979), a common measure in information retrieval, was used as a single-figure measure of performance:

$$F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R}$$

We use $\beta = 1$ which gives no preference to either recall or precision.

3.3 Results

Table 3 summarizes the optimal parameter settings and results for NP, VO, and SV on the test set. In order to find the optimal values of the context size and threshold, we tried $0.1 \leq \theta \leq 0.95$, and maximum context sizes of 1,2, and 3. Our experiments used 5-fold cross-validation on the training data to determine the optimal parameter settings.

In experimenting with the maximum context size parameter, we found that the difference between the values of F_β for context sizes of 2 and 3 is less than 0.5% for the optimal threshold. Scores for a context size of 1 yielded F_β values smaller by more than 1% than the values for the larger contexts.

Figure 2 shows recall/precision curves for the three data sets, obtained by varying θ while keeping the maximum context size at its optimal value. The difference between $F_{\beta=1}$ values for different thresholds was always less than 2%.

Performance may be measured also on a word-by word basis, counting as a success any word which was identified correctly as being part of the target pattern. That method was employed, along with recall/precision, by RM95. We preferred to measure performance by recall and precision for complete patterns. Most errors involved identifications of slightly shifted, shorter or longer sequences. Given a pattern consisting of five words, for example, identifying only a four-word portion of this pattern would yield both a recall and precision errors. Tag-assignment scoring, on the other hand, will give it a score of 80%. We hold the view that such an identification is an error, rather than a partial success.

We used the datasets created by RM95 for NP learning; their results are shown in Table 3.[3] The F_β difference is small (0.4%), yet they use a richer feature set, which incorporates lexical information as well. The method of Ramshaw and Marcus makes a decision *per word*, relying on predefined rule templates. The method presented here makes decisions on *sequences* and uses sequences as its memory, thereby attaining a dynamic perspective of the

[2]The scripts may be found at the URL http://www.cs.biu.ac.il/~yuvalk/MBSL.

[3]Notice that our results, as well as those we cite from RM95, pertains to a training set of 229,000 words. RM95 report also results for a larger training set, of 950,000 words, for which recall/precision is 93.5%/93.1%, correspondingly (F_β=93.3%). Our system needs to be further optimized in order to handle that amount of data, though our major concern in future work is to reduce the overall amount of labeled training data.

	Con.	Thresh.	BE	Recall (%)	Precision (%)	$F_{\beta=1}$
VO	2	0.5	81.3	89.8	77.1	83.0
SV	3	0.6	86.1	84.5	88.6	86.5
NP	3	0.6	91.4	91.6	91.6	91.6
RM95 (NP)	-	-	-	92.3	91.8	92.0

Table 3: Results with optimal parameter settings for context size and threshold, and breakeven points. The last line shows the results of Ramshaw and Marcus (1995) (recognizing NP's) with the same train/test data. The optimal parameters were obtained by 5-fold cross-validation.

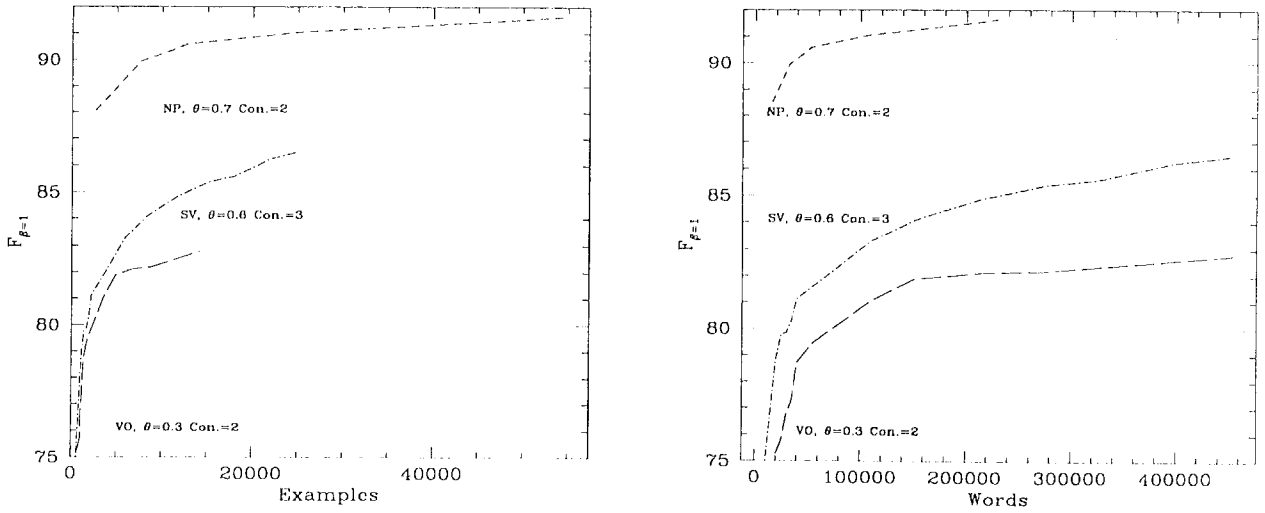

Figure 3: Learning curves for NP, VO, and SV by number of examples (left) and words (right)

pattern structure. We aim to incorporate lexical information as well in the future, it is still unclear whether that will improve the results.

Figure 3 shows the learning curves by amount of training examples and number of words in the training data, for particular parameter settings.

4 Related Work

Two previous methods for learning local syntactic patterns follow the transformation-based paradigm introduced by Brill (1992). Vilain and Day (1996) identify (and classify) name phrases such as company names, locations, etc. Ramshaw and Marcus (1995) detect noun phrases, by classifying each word as being inside a phrase, outside or on the boundary between phrases.

Finite state machines (FSMs) are a natural formalism for learning linear sequences. It was used for learning linguistic structures other than shallow syntax. Gold (1978) showed that learning regular languages from positive examples is undecidable in the limit. Recently, however, several learning methods have been proposed for restricted classes of FSM. OSTIA (Onward Subsequential Transducer Inference Algorithm; Oncina, Garcia, and Vidal 1993), learns a subsequential transducer in the limit. This algorithm was used for natural-language tasks by Vilar, Marzal, and Vidal (1994) for learning translation of a limited-domain language, as well as by Gildea and Jurafsky (1994) for learning phonological rules. Ahonen et al. (1994) describe an algorithm for learning (k,h)-contextual regular languages, which they use for learning the structure of SGML documents.

Apart from deterministic FSMs, there are a number of algorithms for learning stochastic models, eg., (Stolcke and Omohundro, 1992; Carrasco and Oncina, 1994; Ron et al., 1995). These algorithms differ mainly by their state-merging strategies, used for generalizing from the training data.

A major difference between the abovementioned learning methods and our memory-based approach is that the former employ generalized models that were created at training time while the latter uses the training corpus as-is and generalizes only at recognition time.

Much work aimed at learning models for full parsing, i.e., learning hierarchical structures. We refer here only to the DOP (Data Oriented Parsing) method (Bod, 1992) which, like the present work, is a memory-based approach. This method constructs parse alternatives for a sentence based on combinations of subtrees in the training corpus. The MBSL approach may be viewed as a linear analogy to DOP in that it constructs a cover for a candidate based

on subsequences of training instances.

Other implementations of the memory-based paradigm for NLP tasks include Daelemans et al. (1996), for POS tagging; Cardie (1993), for syntactic and semantic tagging; and Stanfill and Waltz (1986), for word pronunciation. In all these works, examples are represented as sets of features and the deduction is carried out by finding the most similar cases. The method presented here is radically different in that it makes use of the *raw sequential* form of the data, and generalizes by reconstructing test examples from different pieces of the training data.

5 Conclusions

We have presented a novel general schema and a particular instantiation of it for learning sequential patterns. Applying the method to three syntactic patterns in English yielded positive results, suggesting its applicability for recognizing local linguistic patterns. In future work we plan to investigate a data-driven approach for optimal selection and weighting of statistical features of candidate scores, as well as to apply the method to syntactic patterns of Hebrew and to domain-specific patterns for information extraction.

6 acknowledgements

The authors wish to thank Yoram Singer for his collaboration in an earlier phase of this research project, and Giorgio Satta for helpful discussions. We also thank the anonymous reviewers for their instructive comments. This research was supported in part by grant 498/95-1 from the Israel Science Foundation, and by grant 8560296 from the Israeli Ministry of Science.

References

S. P. Abney. 1991. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer, Dordrecht.

H. Ahonen, H. Mannila, and E. Nikunen. 1994. Forming grammars for structured documents: An application of grammatical inference. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications (ICGI-94)*, pages 153–167. Springer, Berlin, Heidelberg.

R. Bod. 1992. A computational model of language performance: Data oriented parsing. In *Coling*, pages 855–859, Nantes, France.

E. Brill. 1992. A simple rule-based part of speech tagger. In *proc. of the DARPA Workshop on Speech and Natural Language.*

C. Cardie. 1993. A case-based approach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 798–803, Menlo Park, CA, USA, July. AAAI Press.

R. C. Carrasco and J. Oncina. 1994. Learning stochastic regular grammars by means of a state merging method. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications (ICGI-94)*, pages 139–152. Springer, Berlin, Heidelberg.

W. Daelemans, J. Zavrel, Berck P., and Gillis S. 1996. Mbt: A memory-based part of speech tagger generator. In Eva Ejerhed and Ido Dagan, editors, *Proceedings of the Fourth Workshop on Very Large Corpora*, pages 14–27. ACL SIGDAT.

T. Edward and M. McCreight. 1976. space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April.

D. Gildea and D. Jurafsky. 1994. Automatic induction of finite state transducers for simple phonological rules. Technical Report TR-94-052, International Computer Science Institute, Berkeley, CA, October.

E. M. Gold. 1978. Complexity of automaton identification from given data. *Information and Control*, 37:302–320.

Gregory Greffenstette. 1993. Evaluation techniques for automatic semantic extraction: Comparing syntactic and window based approaches. In *ACL Workshop on Acquisition of Lexical Knowledge From Text*, Ohio State University, June.

L. A. Ramshaw and M. P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora.*

D. Ron, Y. Singer, and N. Tishby. 1995. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT'95)*, pages 31–40, New York, NY, USA, July. ACM Press.

G. Satta. 1997. String transformation learning. In *Proc. of the ACL/EACL Annual Meeting*, pages 444–451, Madrid, Spain, July.

C. Stanfill and D. Waltz. 1986. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, December.

A. Stolcke and S. Omohundro. 1992. Hidden markov model induction by bayesian model merging. In *Proceedings of Neural Information Processing Systems 5 (NIPS-5).*

C. J. van Rijsbergen. 1979. *Information Retrieval.* Buttersworth.

M. B. Vilain and D. S. Day. 1996. Finite-state phrase parsing by rule sequences. In *Proc. of COLING*, Copenhagen, Denmark.