Procesamiento del Lenguaje Natural, nº39 (2007), pp. 89-96 recibido 17-05-2007; aceptado 22-06-2007

Studying CSSR Algorithm Applicability on NLP Tasks

Muntsa Padró and Lluís Padró
TALP Research Center
Universitat Politècnica de Catalunya
Barcelona, Spain
{mpadro, padro}lsi.upc.edu

Resumen: CSSR es un algoritmo de aprendizaje de automatas para representar los patrones de un proceso a partir de datos sequenciales. Este artículo estudia la aplicabilidad del CSSR al reconocimiento de sintagmas nominales. Estudiaremos la habilidad del CSSR para capturar los patrones que hay detrás de esta tarea y en que condiciones el algoritmo los aprende mejor. También presentaremos un método para aplicar los modelos obtenidos para realizar tareas de anotación de sintagmas nominales. Dados todos los resultados, discutiremos la aplicabilidad del CSSR a tareas de PLN.
Palabras clave: Tareas seqüenciales de PLN, aprendizage de automatas, detección de sintagmas nominales

Abstract: CSSR algorithm learns automata representing the patterns of a process from sequential data. This paper studies the applicability of CSSR to some Noun Phrase detection. The ability of the algorithm to capture the patterns behind this tasks and the conditions under which it performs better are studied. Also, an approach to use the acquired models to annotate new sentences is pointed out and, at the sight of all results, the applicability of CSSR to NLP tasks is discussed.
Keywords: NLP sequential tasks, automata acquisition, Noun Phrase detection

1 Introduction

Causal-State Splitting Reconstruction (CSSR) algorithm (Shalizi and Shalizi, 2004) builds deterministic automata from data sequences. This algorithm is based on Computational Mechanics and is conceived to model stationary processes by learning their causal states. These causal states build a minimum deterministic machine that models the process. Its main benefit is that it does not have a predefined structure (as HMMs do) and that if the pattern to learn is simple enough, the obtained automaton is "intelligible", providing an explicit model for the training data.

CSSR has been applied to different research areas such as solid state physics (Varn and Crutchfield, 2004) and anomaly detection in dynamical systems (Ray, 2004). These applications use CSSR to capture patterns representing obtained data. These patterns are then used for different purposes.

This algorithm has been also used in the field of Natural Language Processing (NLP) to learn automata than can be afterwards used to tag new data (Padró and Padró, 2005b; Padró and Padró, 2005a). This is a slightly different use, as it is necessary to introduce some hidden information into the automaton. Furthermore, the alphabets involved in NLP tasks tend to be bigger than the other CSSR applications presented. This is a handicap when using CSSR for NLP tasks, as we will discuss in this paper. Despite of that, the results obtained in first experiments show that this technique can provide state-of-the-art results in some NLP tasks. Given these results, the challenge is to improve them, developing systems rivalling best state-of-the-art systems. To do so, more information should be incorporated into the system but, as it will be discussed in this paper, this can lead to other problems given the nature of the algorithm.

The aim of this work is to study the ability of CSSR to capture a model for the patterns underlying NLP sequences structure, as well as under which conditions it performs better. We focus on studying the models learned by CSSR in NP detection with different data rather than using CSSR to perform the annotating task, which was done in previous work.

ISSN: 1135-5948

2 Theoretical Foundations of CSSR

The CSSR algorithm (Shalizi and Shalizi, 2004) inferres the causal states of a process from data in the form of Markov Models. Thus, the many desirable features of HMMs are secured, without having to make a priori assumptions about the architecture of the system.

2.1 Causal States

Given a discrete alphabet Σ of size k, consider a sequence x^- (history) and a random variable Z^+ for its possible future sequences. Z^+ can be observed after x^- with a probability $P(Z^+|x^-)$. Two histories, x^- and y^-, are equivalent when $P(Z^+|x^-) = P(Z^+|y^-)$, i.e. when they have the same probability distribution for the future. The different future distributions determine *causal states* of the process. Each causal state is a set of histories (suffixes of alphabet symbols up to a preestablished maximum length) with the same probability distribution for the future.

Causal States machines have many desirable properties that make them the best possible representation of a process.They are minimal and have sufficient statistics to represent a process, this is, from causal states it is possible to determine the future for a given past. For that reason we are interested in using these kind of machines in NLP tasks. For more theoretical foundations about causal states and their properties see (Shalizi and Crutchfield, 2001).

2.2 The Algorithm

The algorithm starts by assuming the process is an identically-distributed and independent sequence with a single causal state, and then iteratively adds new states when it is shown by statistical tests that the current states set is not sufficient. The causal state machine is built in three phases briefly described below. For more details on the algorithm, see (Shalizi and Shalizi, 2004).

1. **Initialize:** Set the machine to one state containing only the null suffix. Set $l = 0$ (length of the longest suffix so far).
2. **Sufficiency:** Iteratively build new states depending on the future probability distribution of each possible suffix extension. Suffix sons (ax) for each longest suffix (x) are created adding each alphabet symbol (a) at the beginning of each suffix. The future distribution for each son is computed and compared to the distribution of all other existing states. If the new distribution equals (with a certain confidence degree α) to the distribution of an existing state, the suffix son is added to this state. Otherwise, a new state for the suffix son is created.

   The suffix length l is increased by one at each iteration. This phase goes on until l reaches some fixed maximum value l_{max}, the maximum length to be considered for a suffix, which represents the longest histories taken into account. The results of the system will be significantly different depending on the chosen l_{max} value, since the larger this value is, the longer will be the pattern that CSSR will be able to capture, but also the more training data will be necessary to learn a correct automaton with statistical reliability.
3. **Recursion:** Since CSSR models stationary processes, first of all the transient states are removed. Then the states are splitted until a deterministic machine is reached. To do so, the transitions for each suffix in each state are computed and if two suffixes in one state have different transitions for the same symbol, they are splitted into two different states.

The main parameter of this algorithm is the maximum length (l_{max}) the suffixes can reach. That is, the maximum length of the considered histories. In terms of HMMs, l_{max} would be the potential maximum order of the model (the learned automaton would be an HMM of l_{max} order if all the suffixes belonged to different states).

When using CSSR, it is necessary to reach a trade off between the amount of data (N), the vocabulary size (k) and the used maximum length (l_{max}). According to (Shalizi and Shalizi, 2004), the maximum length that can be used with statistical reliability is given by the ratio $\log N / \log k$.

3 Chunking and NP Detection

This work focus on studying CSSR behaviour when applied to NP detection. This section presents an overview on this task.

Text Chunking consists of dividing sentences into non-recursive non-overlapping phrases (chunks) and of classifying them into a closed set of grammatical classes (Abney, 1991) such as noun phrase, verb phrase, etc. Each chunk contains a set of correlative words syntactically related.

This task is usually seen as a previous step of full parsing, but for many NLP tasks, having the text correctly separated into chunks is preferred than having a full parsing, more likely to contain mistakes. In fact, sometimes the only information needed are the noun phrase (NP) chunks, or, at most, the NP and VP (verb phrase) chunks. For that reason, the first efforts devoted to Chunking were focused on NP-chunking (Church, 1988; Ramshaw and Marcus, 1995), others deal with NP, VP and PP (prepositional phrase) (Veenstra, 1999). In (Buchholz, Veenstra, and Daelemans, 1999) an approach to perform text Chunking for NP, VP, PP, ADJP (adjective phrases) and ADVP (adverbial phrases) using Memory-Based Learning is presented.

As most NLP tasks, Chunking can be approached using hand-built grammars and finite state techniques or via statistical models and Machine Learning techniques. Some of these approaches are framed in the CoNLL-2000 Shared Task (Tjong Kim Sang and Buchholz, 2000).

As the aim of this work is to study the viability of applying CSSR to NLP tasks, specially studying the patterns that CSSR is able to learn, the performed experiments are focused on the task of detecting NPs, ignoring, for the moment, the other kind of chunks.

4 Ability of CSSR to Capture NP Models

This section presents the experiments performed using CSSR to capture the patterns that form language subsequences as NPs. The goal of these experiments is to see how able is this method to infer automata that capture phrase patterns, as well as to study the influence of different l_{max} and amount of training data on the learned automata.

The patterns that may be found in a phrase, depend on the studied word features. For example, there are some orthographical patterns associated with punctuation marks (e.g. after a dot a capitalized word is expected), other more complex patterns associated to syntactic structure of the sentence, etc. Depending on which patterns need to be captured, different features of the words in the sentence should be highlighted.

To use CSSR to learn these patterns, it is necessary to define an alphabet representing the desired features. These features may vary depending on which structures we are really interested in modelling. To learn NP patterns, the used features are the Part of Speech (PoS) tags of words as syntactic structure of sentences depends strongly on them.

The data used for NP detection are extracted from the English WSJ corpus (Charniak, 2000). This is a corpus with full parsing information, with eleven different chunk types and a complete analysis of sentences. though in this work just NP chunks information will be used. The alphabet used to train CSSR consists of a symbol for each PoS tag used in the corpus. The total number of different tags is 44, but there are some PoS tags that never appear inside any NP, so these tags can be merged into one special symbol. With this reduction, the alphabet has 38 symbols.

This training corpus has about 1.000.000 words which means that $l_{max} < \log N / \log k = 3.8$.

To learn an automaton representing NP patterns it is necessary to distinguish the words belonging and not belonging to a NP, even if the PoS tag is the same. To do so each word belonging to a NP is represented by its PoS tag (a symbol of the alphabet) and the words not belonging to NP chunks are mapped into a special symbol. Figure 1 shows an example of how a sentence is translated into a sequence of alphabet symbols.

Word	PoS Tag	Chunk Type	Symbol
He	PRP	NP	PRP
succeeds	VBZ	VP	Out
Terrence	NNP	NP	NNP
Daniels	NNP		NNP
,	,	none	Out
formerly	RB	ADVP	Out
a	DT	NP	DT
Grace	NNP		NNP
chairman	NN		NN
.	.	none	Out

Figure 1: Example of a training sentence and its translation to the alphabet

Sentences encoded in this way are the sequences used to train CSSR. The algorithm

may to learn an automaton representing NP chunks in terms of PoS tags.

Different automata with l_{max} from 1 to 4 were learned, but the obtained automata are not readable, even when minimized [1]. The number of states of the minimized automata varies from 34 for $l_{max} = 1$ to $1,767$ for $l_{max} = 4$.

Given the size of the obtained automata, even after minimization, it is not possible to qualitatively determine if the acquired automata appropriately models NP patterns, so another method to qualitatively evaluate how accurately the generated automaton represents the data was devised, as described in next section.

4.1 Comparing Grammars to Determine the Quality of Learned Models

In order to obtain a qualitative evaluation of the automaton acquired by CSSR for NPs, we will compare it with the regular grammar directly extracted from the syntactic annotations available in the WSJ training corpus.

The grammar obtained from the annotated corpus is regular, since the NP chunks are never recursive and are formed only by terminal symbols in this corpus. So, the grammar consists of the different possible PoS sequences for NPs observed in the corpus, with their relative frequencies.

On the other hand, the automaton learned using CSSR can be used to generate the same kind of patterns: using the transitions and probabilities of the automaton, sequences of PoS tags are generated. The subsequences between two "*Out*" symbols are the NP patterns that CSSR has learned. These patterns, and their occurrence frequencies, are extracted and compared with the grammar acquired from WSJ annotations. The more similar the set of rules produced by CSSR is to the actual WSJ grammar behind the data, the better we can consider the automaton is modelling NP patterns.

To perform the comparison between these two sets of patterns and its frequencies, Jensen-Shannon divergence [2] (Lin, 1991) is used. This divergence gives a measure of the distance between two distributions.

There are two main differences between the rules generated by the CSSR automaton and the rules acquired from corpus annotations. On the one hand, there are rules generated by CSSR automaton that are not present in the corpus. This is due to the fact that CSSR over-generalizes patterns from data. On the other hand, there are some differences in frequencies of common rules, partially due to the probability mass given to wrong rules. Both differences are captured by Jensen-Shannon divergence. The smaller this divergence is, the more similar to the original corpus grammar can the CSSR acquired automata be considered.

The line labelled as "WSJ data" in Figure 2 shows the values of this divergence for different l_{max} values. It can be seen how Jensen Shannon divergence falls as l_{max} grows. This is because the number of overgenerated patterns falls, what means that CSSR generalizes better, as it may be expected. The difference in frequencies of common rules is also lower when using longer histories. For $l_{max} = 4$ the divergence rises again because there are not enough data to learn an automaton with statistical reliability, so using CSSR with this length introduces incorrect patterns.

4.2 Generating Data to Study CSSR Performance

One of the limitations of the study presented in section 4.1 is that, given the size of the alphabet, there are too few available data to learn automata with large l_{max}. As discussed above, the larger l_{max} that can be used with WSJ data is 3, which may be too small to capture long NP patterns.

In order to study the influence of the amount of training data when using such a big alphabet, new data was created in the following way: using the WSJ corpus, which has a complete syntactic analysis, a grammar can be extracted capturing the structure of sentences (divided into different kind of chunks and PoS tags) and of chunks (divided into PoS tags). Each rule has a probability depending on how many times it appears in the training corpus. Using this grammar new data can be generated applying rules recursively until a whole sentence is created.

The generated sentences, are parse trees with the same chunk distribution than the

[1] To minimize the automaton, the probabilistic information of transitions is ignored and a normal minimizing algorithm is applied

[2] A symmetric distance derived from Kullback-Leibler divergence.

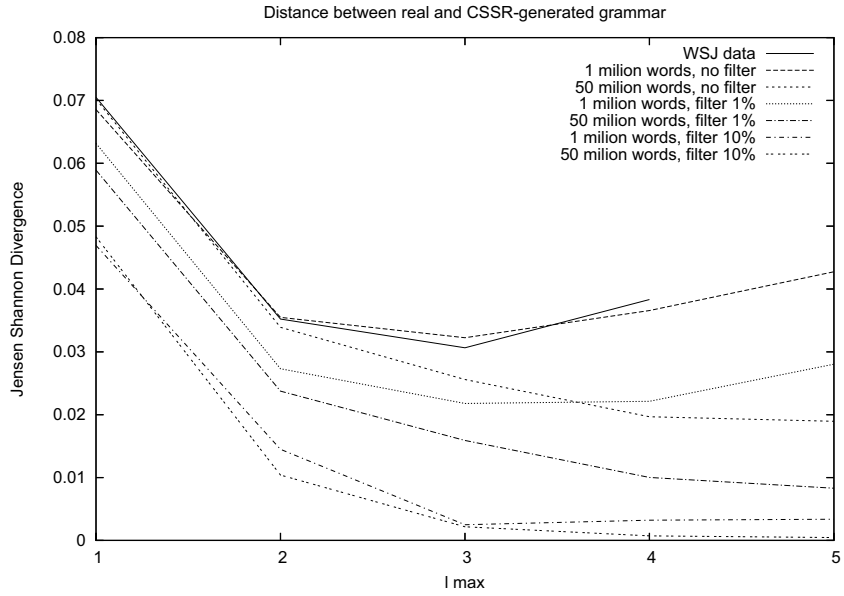

Figure 2: Jensen Shannon divergence between CSSR generated set of rules and real grammar for different values of l_{max} when using different filter levels of the grammar

original corpus. Then, the same method to translate sentences to the NP alphabet described above is performed, and CSSR is used to learn automata.

Note that the NP structures present in the generated data will be the same that the ones observed in real corpus, so creating data in this way is quite similar to replicating the real corpus many times. The aim of this is to simulate that large amounts of data are available and to study the algorithm behaviour under these conditions. In fact, replicating the same data many times is equivalent to artificially simulate that the real data is more significant, and we are interested in studying the influence of doing so in CSSR automata.

Given the nature of the algorithm, repeating the observations N times changes the decision of splitting or not two histories because the statistical significance of the observation changes. This decision is performed using χ^2 statistics and the value of χ^2 is multiplied by N when the data is increased by this value. Thus, generating more data in this way, equals to give more weight to the available data, and the results will show that this leads to learning automata that reproduce data patterns more accurately. The same goal could be theoretically obtained by adjusting the confidence level of the χ^2 tests, but we found this parameter to be less influent on CSSR behaviour.

The reason why in this work we generate data using the grammar rather than replicating the corpus many times is that in this way, experiments can be performed filtering low-frequency rules to get rid of some of the noise from the original corpus. Thus, before generating the data using the learned grammar, the rules that appear less can be filtered and a less noisy corpus can be created. In this way the generated data is expected to be more easily reproduced using CSSR.

The experiments were conducted using different corpora generated with three different grammars: one with all rules learned from WSJ (no filter), which is expected to generate data similar to the WSJ corpus, and two grammars with 1% and 10% of the probability mass filtered. This means that just the most likely rules that sum the 99% or 90% of the mass are conserved.

Using these grammars three different corpora of 50 milions tokens were created. With this amount of data $l_{max} < \log N / \log k = 4.9$ so the maximum usable length is 5. Also, a subset of each corpus of 1 milion tokens was used to perform more experiments, in order to better study the influence of the amount of training corpus.

Figure 2 shows the divergence between the learned automata and the grammar used to generate the corpus, without filtering and with each of the two filters. For each filter level there are two lines: one for the 1 milion words generated corpus and one for the 50

milion words. It can be seen that the results obtained with both non-filtered corpora are very similar to those obtained with WSJ corpus, specially the results obtained with the 1 milion corpus, as this is the size of WSJ. That means that the generated corpus reproduces accurately the NP patterns present in WSJ. Also, it can be seen that the more rules are filtered, the more similar is the learned automaton behaviour to the underlying grammar, since less noisy patterns are more easily captured by CSSR.

These results also show that using more training data enables CSSR to learn more accurate automata for larger l_{max}. While for low l_{max} values increasing the amount of data doesn't introduce significant differences, if enough data is available CSSR can use larger l_{max} and infer more informed automata that reproduce better the grammar behind the real corpus. Generating corpus does not really introduce new patterns, but simulates that the patterns present in real data have more statistical significance.

4.3 Discussion

At the sight of the results, we can conclude that CSSR is a good method for learning patterns, even quite complicated patterns as those of NPs, but it is highly dependent on amount of available data. For each process, there is a necessary l_{max} value that captures the patterns, and if this value is big, large corpus will be necessary. Furthermore, as the minimum amount of data necessary to learn an automaton with a determined l_{max} depends exponentially on the alphabet size ($N > k^{l_{max}}$), to be able to increase l_{max} in 1, it would be necessary to multiply the data size by the size of the alphabet k.

For NP detection, CSSR generated automaton is not readable, but that doesn't mean that it doesn't reproduces NP patterns correctly. The automaton can be qualitatively studied comparing the patterns that it generates with the patterns observed in the training corpus. The more similar are the two sets of patterns, the better is CSSR reproducing the patterns of the task. This comparison shows that for real data CSSR can learn better patterns as l_{max} grows but due to the limited amount of available data, for $l_{max} = 4$ the divergence rises again, as there is not enough data to learn an automaton reproducing corpus patterns with this length. So, the performance of the system is limited by the size of the training corpus.

The generated and not filtered data can be considered equivalent to the real corpus. Also, it can be seen that when using a big amount of generated data the performance is better than for the real data as the system can deal with longer l_{max} . When using small l_{max} the difference between using 1 milion or 50 milion data is not significant. Furthermore, as it was expected, as the number of filtered rules grows, the divergence falls, being really small when l_{max} grows. This means that the easier the patterns to learn are, the better they are captured by CSSR. In the case of filtered rules, the system also performs better with large l_{max} if enough data is available.

Furthermore, in (Padró and Padró, 2005b) similar experiments to those presented here were performed for Named Entity Recognition (NER). In this case, the learned automata were readable when minimized, and captured correctly the patterns of sentences given the chosen sets of features. The conclusion was that CSSR was able to learn correctly the patterns of NEs with the chosen alphabet, what combined with the results presented in this work, can lead to the conclusion that CSSR is a good method to capture language structures if enough data is available.

5 Applying CSSR to Annotating Tasks

This work has focused on the ability of CSSR to learn phrase patterns in terms of some selected sets of features, and has been seen that CSSR can reproduce correctly the patterns of some NLP structures.

Nevertheless, in these NLP tasks it is necessary not only to obtain generative phrase models, but also to develop systems able to annotate new sentences. To perform this tagging task, hidden information about where a NP begins and ends must be taken into account. An usual approach is to encode this information in "B-I-O" tags (Ramshaw and Marcus, 1995): each word has a B, I or O tag, where B stands for words at phrase (chunk or NE) **B**eggining, I for words **I**nternal to a phrase, and O for words **O**utside a phrase.

When CSSR is to be used to annotate new text, it is necessary to introduce this hidden information into the system. In (Padró and Padró, 2005b; Padró and Padró, 2005a) an

approach to use CSSR for NER and Chunking was presented, which will be summarized here in order to discuss the applicability of CSSR to NLP tasks.

The basic idea of the method is that it is necessary to introduce into the alphabet the hidden information of the tag (B, I or O). To do so, each symbol encoding the features previously selected (e.g. $\Sigma = \{ DT, NN, NNP$, etc. $\}$ for NP) is combined with each possible B-I-O tag ($\Sigma = \{ DT_B, DT_I, DT_O, NN_B, NN_I$, etc$\}$). Thus, each word in the training corpus is translated to one of these symbols forming the training sequence.

When a new sentence has to be tagged, the part of the symbol related to context features is known (e.g. "DT", 'NN", etc) but the information about the correct B-I-O tag is not available, so there are three possible alphabet symbols for each word (e.g. DT_B, DT_I, DT_O, if the visible part is a DT).

To find the most likely tag for each word in a sentence –that is, to find the most likely symbol of the alphabet–, (e.g. DT_B, DT_I, DT_O for a DT word) a Viterbi algorithm is applied. For each word in a sentence, the possible states the automaton could reach if the current word had the tag B, I, or O, and the probabilities of these paths are computed. At the end of the sentence, the best probability is chosen and the optimal path is backwards recovered. In this way, the most likely sequence of B-I-O tags is obtained.

5.1 Results on NP Detection

For NP detection experiments, CoNLL-00 shared task (Tjong Kim Sang and Buchholz, 2000) data are used. The training corpus has about 200,000 words, and the best obtained F_1 is 89.11% with $l_{max} = 2$. In fact, in (Padró and Padró, 2005a) chunking with all chunk types was performed, obtaining an overall result of $F_1 = 88.20$ which is comparable to last systems in the competition but is quite far from best systems.

Furthermore, following the strategy depicted in section 4.2, we can force the statistical significance of hypothesis test by reproducing the data many times. Doing so leads to a improvement of the results, obtaining $F_1 = 90.96$ also with $l_{max} = 2$ when the data is replicated 1000 times. So increasing the significance of data leads to better results when performing also annotating tasks.

Also, in (Padró and Padró, 2005b), similar experiments (without replicating the corpus) to perform NER with CSSR were presented. In those experiments the best parametrization led to a F_1 of 88.96%. The system with this parametrization, combined with the NEC system used by the winner of CoNLL-2002 shared task (Carreras, Màrquez, and Padró, 2002), would situate our system in the fifth position of the competition. This is not a bad result, specially taking into account the simplicity of the used features.

5.2 Discussion

The results obtained on NP annotating task, show that the problem with the necessary amount of data becomes worse when trying to use CSSR to tag new sentences.

First experiments with these kind of tasks were promising, as the used approach was very simple and the results were comparable to state-of-the-art systems. Nevertheless, if more information is to be included into the system to try to improve obtained results, a limitation will be found due to the amount of necessary data. Furthermore, even if enough data were available, a computational limitation will be found, specially in tasks such as NP detection, where the alphabet is big and lots of data have to be processed.

The main problem of this approach is that to introduce the hidden information the alphabet size is multiplied by 3, what means that the amount of data necessary to use CSSR with the same l_{max} used without B-I-O information is $3^{l_{max}}$ times bigger than what was needed before. If CSSR can learn an accurate automaton of length l using a training corpus of $N = k^l$ words, $N' = (3k)^l = N * 3^l$ words will be necessary to perform the tagging task under the B-I-O approach.

6 Conclusions and Future Work

A study of how CSSR is able to capture patterns in language has been presented. It has been seen that this algorithm can learn automata representing processes if there are enough data available, or if the process is simple enough.

One of the main limitations of CSSR is that it is useful to learn patterns, but it is not directly prepared to introduce hidden information and to perform annotating tasks. The approach presented in (Padró and Padró, 2005b) gives reasonably good results for NER but not so good results in NP detec-

tion. This is because as the alphabet grows, more than the available data would be necessary to learn an accurate automaton, and the available corpus is not big enough.

The main conclusion of this work is that CSSR can learn correctly the patterns of sequential data, specially if the data is not very noisy, but that it is highly dependent on the amount of data, the size of the alphabet and l_{max} . Furthermore, this dependency is exponential, so to increase a little bit the performance of the system, it would be necessary to magnify the amount of data. So, CSSR can be useful when dealing with systems with small alphabets –as in other applications of CSSR such as those presented in (Varn and Crutchfield, 2004; Ray, 2004)– but to use it in systems with lots of features to be taken into account, as NLP annotating tasks, a limitation due to the amount of available data will be probably found.

In this line, the main future line devised is to modify CSSR to be able to introduce more information into the system. As the alphabet size has to be small, our proposal is to introduce all the features not encoded in the alphabet via Maximum Entropy (ME) models. Thus, the histories would consist of sets of features, instead of suffixes, and CSSR would build the causal states taking into account the probability of seeing a symbol after a determined history, computing it using ME, instead of taking into account just the simple suffixes and its transition probabilities.

References

Abney, Steven. 1991. *Parsing by Chunks*. R. Berwick, S. Abney and C. Tenny (eds.) Principle–based Parsing. Kluwer Academic Publishers, Dordrecht.

Buchholz, Sabine, Jorn Veenstra, and Walter Daelemans. 1999. Cascaded grammatical relation assignment. In *In Proceedings of EMNLP/VLC-99*, pages 239–246, University of Maryland, USA.

Carreras, Xavier, Lluís Màrquez, and Lluís Padró. 2002. Named entity extraction using adaboost. In *Proceedings of CoNLL Shared Task*, pages 167–170, Taipei.

Charniak, Eugene. 2000. Bllip 1987-89 wsj corpus release 1. In *Linguistic Data Consortium*, Philadelphia.

Church, Kenneth W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 1st Conference on Applied Natural Language Processing, ANLP*, pages 136–143. ACL.

Lin, J. 1991. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151.

Padró, Muntsa and Lluís Padró. 2005a. Approaching sequential nlp tasks with an automata acquisition algorithm. In *Proceedings of International Conference on Recent Advances in NLP (RANLP'05)*, Bulgaria, September.

Padró, Muntsa and Lluís Padró. 2005b. A named entity recognition system based on a finite automata acquisition algorithm. *Procesamiento del Lenguaje Natural*, (35):319–326, September.

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

Ray, Asok. 2004. Symbolic dynamic analysis of complex systems for anomaly detection. *Signal Process.*, 84(7):1115–1130.

Shalizi, Cosma R. and James P. Crutchfield. 2001. Computational mechanics: pattern, prediction strucutre and simplicity. *Journal of Statistical Physics*, 104:817–879.

Shalizi, Cosma R. and Kristina L. Shalizi. 2004. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference*.

Tjong Kim Sang, Erik F. and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. In Claire Cardie, Walter Daelemans, Claire Nedellec, and Erik Tjong Kim Sang, editors, *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132. Lisbon, Portugal.

Varn, D. P. and J. P. Crutchfield. 2004. From finite to infinite range order via annealing: The causal architecture of deformation faulting in annealed close-packed crystals. *Physics Letters A*, 324:299–307.

Veenstra, J. 1999. Memory-based text chunking. In *Nikos Fakotakis (ed), Machine learning in human language technology, workshop at ACAI 99*, Chania, Greece.