Applying Causal-State Splitting Reconstruction Algorithm to Natural Language Processing Tasks

Tesi Doctoral - PhD Thesis

per a optar al grau de
Doctora en Informàtica

by

Muntsa Padró Cirera

advisor

Dr. Lluís Padró Cirera

Ph.D. Program in Artificial Intelligence
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Barcelona, June 2008

Al Martí, a la meva família i als meus amics.
Gràcies per ser-hi sempre.

Abstract

This thesis is focused on the study and use of Causal State Splitting Reconstruction (CSSR) algorithm for Natural Language Processing (NLP) tasks. CSSR is an algorithm that captures patterns from data building automata in the form of visible Markov Models. It is based on the principles of Computational Mechanics and takes advantage of many properties of causal state theory. One of the main advantages of CSSR with respect to Markov Models is that it builds states containing more than one n-gram (called history in computational mechanics), so the obtained automata are much smaller than the equivalent Markov Model.

In this work, we first study the behavior of the algorithm when learning patterns related to NLP tasks but without performing any annotation task. This first experiments are useful to understand the parameters that affect the algorithm and to check that it is able to capture the patterns present in natural language sentences.

Secondly, we propose a way to apply CSSR to NLP annotation tasks. The algorithm is not originally conceived to use the hidden information necessary for annotation tasks, so we devised a way to introduce it into the system in order to obtain automata including this information that can be afterwards used to annotate new text. Also, some methods to deal with unseen events and a modification of the algorithm to make it more suitable for NLP tasks have been presented and tested. These three aspects conform the first line of contributions of this research, altogether with a deep experimental study of the proposed methods.

The experimental study of the proposed approach is performed in three different tasks: Named Entity Recognition in general and Biomedical domain and Chunking. The obtained results are promising in the two first tasks though not so good for Chunking. Nevertheless, it is not easy to improve the obtained performance following the same approach, since CSSR needs quite reduced feature sets to build correct automaton and that limits the performance of the developed system. For that reason, we propose to combine CSSR with graphical models, in order to enrich the features that the system can take into account.

This combination conforms the second line of contributions of this thesis. There is a variety of possible graphical models that can be used, but for the moment we propose to combine CSSR algorithm with Maximum Entropy (ME) models. ME models can be used as a way of introducing more information into the system, encoding it as features. In this line, we propose and test two methods for combining CSSR and ME models in order to improve the results obtained with original CSSR. The first method is simple and does not modify the automata-building algorithm while the second one is more sophisticated and builds automata taking into account the ME features. We will see that though much more simpler, the first method leads to an important improvement with respect to original CSSR but the second method does not.

Resum

Aquesta tesi es centra en l'estudi i en l'ús de l'algorisme "Causal State Splitting Reconstruction (CSSR)" per tasques de Processat de Llenguatge Natural (PLN). El CSSR és un algorisme que captura els patrons d'un conjunt de dades construint autòmats d'estats finits en la forma de Models de Markov visibles. Es basa en els principis de la Mecànica Computacional tot i traient profit de les moltes propietats interessants de la teoria d'estats causals. Un dels principals avantatges del CSSR respecte els Models de Markov és que construeix estats que contenen més d'un n-gram, per tant els autòmats que s'obtenen són molt més petits que el Model de Markov equivalent.

En aquest treball, primer de tot estudiem el comportament de l'algorisme quan l'apliquem a l'aprenentatge de patrons relacionats amb tasques de PLN però sense realitzar cap tasca d'anotació. Aquests experiments inicials ens serveixen per entendre els paràmetres que afecten l'algorisme i per comprovar que el CSSR pot aprendre els patrons que es troben en les frases de llenguatge natural.

Seguidament, proposem un mètode per aplicar el CSSR a tasques d'anotació de seqüències de llenguatge natural. L'algorisme no està originalment pensat per incloure la informació oculta que es necessita en aquest tipus de tasques, per tant hem dissenyat un mètode per introduir-la al sistema i així obtenir autòmats que inclouen aquesta informació i poden ser usats per anotar text nou. De la mateixa manera, proposem dos mètodes per tractar els esdeveniments no observats en les dades i una modificació de l'algorisme que el fa més apte per tasques de PLN. Aquests tres aspectes conformen la primera línia de contribucions d'aquesta tesi, juntament amb un estudi experimental detallat dels mètodes proposats aplicats a diferents tasques de processat de llenguatge natural.

Aquest estudi experimental es realitza sobre tres tasques diferents: reconeixement d'entitats amb nom tant en un domini general com en el domini Biomèdic i detecció de sintagmes. Els resultats obtinguts són prometedors en les dues primeres tasques, però no tan bons en l'última. No obstant, no és fàcil millorar els resultats obtinguts seguint el mateix mètode, ja que el CSSR necessita tractar amb un nombre reduït de característiques per construir autòmats correctes i això limita la potència del sistema, ja que no pot tractar informació complicada. Per aquesta raó, proposem combinar el CSSR amb models gràfics, per així poder introduir informació més sofisticada al sistema.

Aquesta combinació és la segona línia de contribucions d'aquesta recerca. Hi ha diversos models gràfics que es podrien usar, però de moment nosaltres proposem combinar el CSSR amb models de Màxima Entropia (ME). El primer mètode que proposem és el més simple i no modifica l'algorisme de construcció de l'autòmat sinó que només usa els models de ME per la tasca d'anotació. El segon mètode és més sofisticat i modifica el CSSR per tal que els autòmats construïts tinguin en compte tota la informació usant els models de ME. Veurem que els primer mètode, tot i ser molt més simple, aporta una millora important dels resultats, mentre que el segon mètode no aconsegueix millorar-los significativament.

Agraïments

Aquesta tesi no hauria estat possible sense l'ajuda de moltes persones a qui vull donar les gràcies. Primer de tot, vull agrair al meu germà i director Lluís la seva ajuda, bon humor i guia durant tots aquests anys. D'ell vaig aprendre, ja des de petita, el pensament crític i racional. Durant el meu doctorat m'ha ensenyat, a més, què significa fer investigació.

També vull agrair a en Cosma Shalizi i la Kristina Klinkner, creadors de l'algorisme que he utilitzat en aquesta tesi, la seva ajuda en tots els temes teòrics i pràctics que envolten aquest algorisme. Sense ells i les seves pacients respostes, no hauria arribat al nivell de comprensió necessari per realitzar aquest treball.

Hi ha molta més gent a qui vull mencionar. Tinc la sort d'haver treballat en un grup on regna el bon humor i l'amabilitat. Agraeixo especialment a l'Horacio Rodríguez els seus desinteressats consells pel que fa a la meva recerca i a la redacció d'aquesta tesi. I vull mencionar amb afecte a totes les persones amb qui he compartit despatx, estones de feina, vermuts i sopars. Gràcies Eli, Robert, Victoria, Luís, Jesús, Montse, Jordi A., Xavi, Edgar, Pere, Meritxell, Maria, Dani, Roberto, Jordi P., Emili, JuanFra, Mauro i Sergi pels somriures cada matí. I gràcies també a la resta del grup de Processat de Llenguatge Natural de la UPC: Jordi T., Marta, Neus, Ma Teresa, Lluís M., Alicia, Manu, Bernardino, Núria, Jordi D., Gerard, Javier, David, Ramon, Àngels, Samir, Patrik i Carme.

Així mateix, vull donar les gràcies a en Bill Keller per haver-me acollit durant quatre mesos a la Universitat de Sussex. A ell i a en James Dowdall els agraeixo el treball que vam realitzar junts, i a tota la gent del grup de processat de llenguatge natural de la Universitat de Sussex la seva hospitalitat.

També vull expressar el meu agraïment a totes les persones que han revisat i llegit aquesta tesi. Agraeixo la seva atenció als membres del tribunal i als tres revisors anònims que m'han fet arribar els seus comentaris sobre una versió preliminar d'aquest document.

Part de la meva recerca ha estat finançada pel Comissionat per a Universitats i Recerca del Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya i del Fons Social Europeu a través d'una beca predoctoral FI, vinculada al projecte ALIADO (Tecnologías del Habla y el Lenguaje para un Asistente Personal, TIC2002-04447-C02-01). També m'han donat suport econòmic diferents projectes d'àmbit europeu i estatal: CHIL (Computers In the Human Interaction Loop, IP 506909), KNOW (Desarrollo de Tecnologías Multilingües a Gran Escala para la Comprensión del Lenguaje Natural), EurOpenTrad (Traduccion Automática Avanzada de Código Abierto para la Integracion Europea de las Lenguas del Estado Espanyol, FIT-350401-2006-5) i HOPS (Enabling an Intelligent Natural Language Based Hub for the Deployment of Advanced Semantically Enriched Multi-channel Mass-scale Online Public Services, IP 507967).

També vull expressar el meu agraïment a la gent de l'administració i del laboratori de càlcul del departament de LSI de la UPC que han fer possible la meva feina i estudis aquí.

En el camp personal, vull agrair profundament el seu suport i dedicar aquesta tesi a tota la gent que m'ha fet costat tots aquests anys. A la meva família, per creure sempre en mi. A en Martí, per la seva infinita paciència, suport i alegria, i per ajudar-me a veure les coses clares quan tot s'enfosqueix. I als meus amics, per fer-me la vida molt més agradable. No els puc mencionar a tots, però es mereixen una especial atenció en Jordi, per donar-me un gran suport moral i força de tècnic, l'Alba, per ser molt més que una germana i estar sempre al meu costat, l'Alina, la Teresa, en Sandri, la Vir, la Montse i en Dídac per tanta complicitat i en Robert, en Pere, l'Anna, els tres Davids, l'Olga, en Marc i tota la gent amb qui he compartit la meva passió per tota la diversió, comprensió i felicitat.

Acknowledgments

This thesis would not have been possible without the help of many people to whom I am very grateful. First of all, I want to thank my brother and supervisor Lluís for his help, high spirits and guidance during all these years. From him I have learnt, since I was a small girl, critical and rational thinking. During my PhD he has taught me, also, the meaning of research work.

I also want to thank Cosma Shalizi and Kristina Klinkner, authors of the algorithm used in this thesis, for their help in all the theoretic and practical aspects of this algorithm. Without them and their patient answers I wouldn't have been able to understand all the necessary details to perform this work.

There are many other people I would like to mention. I have been so lucky as to work in a group where there was always friendliness and a good atmosphere. I am specially thankful to Horacio Rodriguez for his help during my research and the writing of this dissertation. Also, I want to mention with warm affection all the people with whom I have shared my office, my working time, celebrations and dinners. Thank you Eli, Robert, Victoria, Luís, Jesús, Montse, Jordi A., Xavi, Edgar, Pere, Meritxell, Maria, Dani, Roberto, Jordi P., Emili, JuanFra, Mauro and Sergi for your smiles every morning. And thank you also to the rest of the UPC Natural Language Processing Group: Jordi T., Marta, Neus, Ma Teresa, Lluís M., Alicia, Manu, Bernardino, Núria, Jordi D., Gerard, Javier, David, Ramon, Àngels, Samir, Patrik and Carme.

Furthermore, I am very grateful to Bill Keller for receiving me as a visiting student at the University of Sussex for four months. I thank him and James Dowdall for the work we did together and I also thank all the people of Natural Language Processing group of the University of Sussex for their hospitality.

A special acknowledgment goes to the people that have revised and read this thesis. I sincerely thank the three anonymous reviewers that provided me with very useful feedback about a previous version of this thesis and the people of the thesis committee.

This research has been partially funded by the Catalan Goverment (Comissionat per a Universitats i Recerca del Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya) through a pre-doctoral grant related to the ALIADO project (Tecnologías del Habla y el Lenguaje para un Asistente Personal, TIC2002-04447-C02-01). Also, some other European and Spanish projects have supported my research economically: CHIL (Computers In the Human Interaction Loop, IP 506909), KNOW (Desarrollo de Tecnologías Multilingües a Gran Escala para la Comprensión del Lenguaje Natural), EurOpenTrad (Traducción Automática Avanzada de Código Abierto para la Integración Europea de las Lenguas del Estado Español, FIT-350401-2006-5) and HOPS (Enabling an Intelligent Natural Language Based Hub for the Deployment of Advanced Semantically Enriched Multi-channel Mass-scale Online Public Services, IP 507967).

Finally, I also want to express my gratitude to the people from the administrative and technical offices of the LSI department of the UPC that have made possible the years I have been working and studying here.

Contents

CHAPTER 1

Introduction

Natural Language Processing (NLP) tasks have an intrinsic difficulty due to the fact that human language is very complex and ambiguous. In fact, the various kinds of ambiguity (e.g. lexical, syntactic, semantic, referential...) are one of the main problems in NLP. Solving those ambiguities is necessary to develop reliable tools (PoS taggers, Chunking, Named Entity Recognizers...) that are required to build more sophisticated systems devoted to applications such as Question Answering, Machine Translation, Information Retrieval, or Summarization among others.

1.1 Motivation

Although a Natural Language sequence may encode a semantically or pragmatically very complex message, it is encoded and transmitted in the form of a sequence of sounds or words. These word sequences follow certain patterns which are usually the focus of most of the NLP basic processors. For instance, Named Entity Recognition, Chunking, Part of Speech Tagging, Multi-word Expressions Detection, are tasks that depend largely on those sequential patterns and may be approached with systems taking into account this sequential structure.

This thesis is focused on approaching some NLP sequential tasks within the framework of learning graphical models from data, more concretely, we are interested in learning Probabilistic Finite State Automata (PFSA). From the various algorithms that statistically induce this kind of models, we focus on the study of a specific automata learning algorithm (CSSR) and on its applicability to some sequential NLP tasks.

We focus on this algorithm because it is similar to other algorithms successfully applied to NLP sequential tasks, such as Markov Models or Variable length Markov Models, but has some different properties that may be useful when learning NLP patterns, for example that it represents the learned patterns in the form of a PFSA.

1.2 Overview

NLP tasks can be approached using techniques based on linguistic knowledge and hand-built rules or grammars, or can be also approached using Machine Learning (ML) techniques. We are specially interested in the latter. Many ML techniques have been applied successfully to various NLP tasks, but when dealing with sequential structures, it is important to use Machine Learning techniques that take into account this sequence information. According to Dietterich (2002), the supervised learning algorithms may fail to learn this kind of processes because the training data are not drawn independently and identically as these methods usually assume, but, as said before, the values in a sequence may influence each other.

1.3 Goals

The main focus of this work is the study and treatment of graphical models applied to sequential NLP tasks. These graphical models often model the sequences using Finite State Automata (FSA) or grammars. There are many approaches to this kind of tasks, some of the most widely used are Hidden Markov Models (HMM) or their extensions (Maximum Entropy Markov Models, Input-Output HMMs, Conditional Random Fields...). In these models, language utterances are modelled as state sequences where the states represent the hidden information and the observations are modelled as emissions from these states. Once the way to encode the hidden information is set, the model is built by computing the various involved probabilities. For example, when using a trigram HMM to perform Part of Speech (PoS) tagging, the states represent tag bigrams (one state for each possible combination of two PoS tags) and the emitted symbols are the words in a sentence. Then, the model is trained by computing the transition probabilities from state to state (that is, the trigram probabilities) and the emission probabilities (probability of emitting each word being in a determined state).

The main problem of this kind of algorithms is that the acquired graphical model has a predefined structure, so they are not very flexible and may not adapt well to some kinds of problems. Furthermore, they suffer from data sparseness. However, there are some algorithms that learn graphical models without predefining the structure of the problem. These algorithms are more flexible and can adapt better to each problem. Also, they can build better models with sparse data.

This work is focused on the study of Causal-State Splitting Reconstruction algorithm (CSSR) (Shalizi & Shalizi, 2004) and on its application to sequential NLP tasks. This algorithm is based on Computational Mechanics (Crutchfield, 1994) and is originally conceived to model stationary processes by learning their causal states given data sequences. These causal states build a deterministic automaton that models the process. Its main benefit is that it builds automata in the form of Markov Models but do not need to have a predefined structure. Furthermore, the built models are always minimal in the sense that the automata have always less (or equal) number of states than the equivalent Markov Model, since the causal states join various suffixes in the same state, building a much smaller model than the Markov Models.

Also, another interest of using CSSR is that, since it builds minimal models, if the pattern to learn is simple enough, the obtained automaton is "intelligible", providing an explicit model for the training data.

Once the algorithm and its ability to learn patterns has been studied, we will present

an adaptation of CSSR to apply it to some NLP tasks such as text Chunking and Named Entity Recognition both in general and medical domains. In all these tasks, the automata learned using CSSR are used as a tool to tag new sentences. To do so, it is important to introduce into the system knowledge about the tags taken into account, that is, about the hidden information. In this work, we present a simple approach to encode this hidden information into the alphabet.

1.4 Structure of this Document

The rest of this thesis is organized in the following chapters:

- **Chapter 2: State of the Art.**

  This chapter reviews the state of the art of a variety of Machine Learning algorithms, with and special attention to algorithms for capturing sequential structures. Also, we introduce the NLP sequential tasks approached in this thesis and some systems devoted to those tasks.

- **Chapter 3: Computational Mechanics and CSSR.**

  This chapter introduces the basic concepts of Computational Mechanics and the CSSR algorithm with some simple examples of how it learns automata. Once it is presented, we compare CSSR with related techniques and we also review applications of the algorithm in various research fields.

- **Chapter 4: Study of CSSR Ability to Capture Language Subsequence Patterns**

  Chapter 4 presents some experiments that we performed to study the applicability of CSSR algorithm in NLP tasks (Padró & Padró, 2007b). We used the algorithm to learn the patterns of language subsequences such as Named Entities and Noun Phrases and studied the correctness of these patterns and the parameters that have the largest influence on the algorithm performance. In these first experiments no annotation task was performed, the algorithm is used just to capture the patterns of language subsequences.

- **Chapter 5: Adapting CSSR to Perform Annotation Tasks.**

  This chapter introduces our proposal to use CSSR for annotating tasks (Padró & Padró, 2005a). This is one of the main contributions of this thesis. Since the algorithm is not directly prepared to introduce hidden information, we devised a method to introduce this information into the system and use it to tag new text. Furthermore, two methods to deal with unseen events that may be found in the annotation step are explained. Once all these contributions are introduced we present some simple experiments used to validate the proposed method.

- **Chapter 6: Experiments and Results with CSSR in Tagging Tasks.**

  In this chapter, the performed experiments and the obtained results with the proposed method are presented and discussed. The approached tasks are: Named Entity Recognition, both in general (Padró & Padró, 2005c) and Biomedical domains (Dowdall et al., 2007), and Text Chunking (Padró & Padró, 2005b). Studying the obtained

results, it is shown that one of the main error sources is the way in which CSSR determines the recurrent states of the automaton, that are deleted. For that reason a modification of this step of the algorithm is proposed and tested, obtaining better results than using original CSSR.

- **Chapter 7: Extending CSSR with ME models.**

  This is the second main contribution of this thesis. In this chapter, we present two different approaches for combining Maximum Entropy Models and CSSR (Padró & Padró, 2007a). In this way, we expect to improve the performance of CSSR alone since with ME models we can introduce more complicated information into the system.

- **Chapter 8: Experiments and Results with ME-extended CSSR.**

  In this chapter, the experiments and results performed with the two extensions of CSSR using Maximum Entropy Models are presented and discussed. These experiments are performed for Named Entity Recognition task.

- **Chapter 9: Conclusions and Future Directions.**

  To conclude this work, chapter 9 reviews the main contributions and conclusions of this thesis as well as the main future lines that this research opens.

CHAPTER 2

State of the Art

This chapter presents a review of techniques related to Machine Learning for sequential NLP tasks as well as automata-learning algorithms in general and CSSR in particular. Also, the tasks approached in this work and some state of the art systems devoted to them are introduced.

2.1 Sequential Tasks

We call sequential tasks those that deal with data organized as sequences, i.e. data where the elements have significant sequential correlation. The data concerning these kind of tasks consist of sequences of (x, y) pairs, where x is an observation and y some hidden information about it (e.g. a word and its correct PoS tag, a Named Entity and its class, a word and the tag regarding the chunk it belongs to, etc.). These (x, y) pairs will be usually related to other values in the sequence, where the order in which they appear is important.

We choose to focus on algorithms for extracting patterns from sequential data because this kind of structures are very frequent in Natural Language, since sentences are sequences of words influencing each other. In fact, many basic NLP tasks can be regarded as sequential tasks. Depending on the studied task, the relevant hidden information we are interested in will be different.

Next section presents a review of different techniques that induce models from sequential data. We focus specially on Machine Learning (ML) techniques for learning sequential models, since this is our framework and the algorithm studied in this thesis belongs to this algorithm class.

2.2 Learning Sequential Models

There are many different ways of representing sequential processes, and also plenty of algorithms for learning the different possible representations from data. We are specially interested in the study of statistical graphical models for sequential tasks.

Graphical models (Lauritzen, 1996) are often classified into two classes: directed and undirected. Undirected graphical models are those that are represented as a graph with undirected edges. Some examples are Markov Networks or Conditional Random Fields. On the other hand, directed graphical models are represented as directed acyclic graphs where each node is a random variable and the arcs represent statistical dependencies between the variables. Some models belonging to this class are Hidden Markov Models or Neural Networks, among others. In this chapter, we will review some directed and undirected graphical models, with a special interest to those that also belong to the statistical models class.

Statistical models can be classified as generative or discriminative models (also known as conditional models). In next sections, we make a brief survey of different possible pattern representations and algorithms for learning them using this last classification.

2.2.1 Generative Models

Generative models define a joint probability distribution over observation and label sequences $p(x, y)$. From this joint probability, a conditional distribution can be built from a generative model through the use of Bayes rule, though very strict independence assumptions must be done. These models can be used to generate data or, using the conditional probability, to deduce the unknown class y.

Generative models have been widely used in many applications of different fields. In this category of algorithms there are, for example, Hidden Markov Models (Rabiner, 1990), Input-Output HMMs (IOHMM) which are HMMs for which the emission and transition distributions are conditional on another sequence (Bengio & Frasconi, 1995; Bengio, 1996), Generalized Hidden Markov Models that are Markov Models with the transition matrices generalized in such a way that they can contain any real number, so they do not represent probabilities (Upper, 1997), or Stochastic Grammars (Lari & Young, 1990). These methods have been successfully used in a wide variety of NLP problems, not only in text processing but also in speech recognition. See (Manning & Schütze, 1998) for an overview on these applications.

Nevertheless, these models have the problem that very strict independence assumptions between observations are made when using the Bayes rule to compute the conditional distribution from the joint probability distribution. Furthermore, it is not practical to represent multiple interacting features or long-range dependencies of the observations, since the inference problem for such models is intractable.

2.2.2 Conditional Models

The limitations that generative models introduce, due to the independence assumptions that must be done, make interesting to explore discriminative or conditional models. These models are used for modeling the conditional probability of an unobserved variable y on an observed variable x, $p(y|x)$. This probability can be used for predicting y from x without having to use Bayes rule and the consequent independence assumptions. On the other hand, conditional models, differently from generative models, do not allow the generation of samples.

In this section, we focus on graphical conditional models, though there are also conditional models that do not have a graphical representation, such as Maximum Entropy Models (Berger et al., 1996; Ratnaparkhi, 1997).

This kind of conditional models are non-generative finite-state models based on next-state classifiers that specify the probabilities of possible label sequences given an observation sequence. These models can take into account the possible dependency of the transition probability between labels with not only the current but also with other observations. The independence assumption of generative models does not allow this. Some examples of conditional models are Maximum Entropy Markov Models (MEMMs) (McCallum et al., 2000), discriminative Markov Models (Bottou, 1991) and Conditional Random Fields (CRF) (Lafferty et al., 2001). In fact, CRFs perform better and avoid some MEMMs limitations in many experiments performed by Lafferty et al. (2001).

All the models mentioned above, both generative and conditional, have the limitation that the structure of the model has to be previously determined, they make strong assumptions about the nature of the data-generating process. Conditional models have the benefit that extra features can be introduced which can give more information to the system. Nevertheless, the only information acquired from the training set are the probability distributions for the models. For that reason, we are interested in studying those algorithms that learn finite state automata (FSA) from data. Though FSAs are a subset of generative models, we are interested in studying them separately because the algorithm studied in this work belongs to this class. Furthermore, this kind of representation has the benefit that it is not necessary to pre-set the structure of the graphical model so it is interesting to distinguish it from other generative models such as Markov Models.

2.2.3 Finite State Automata

There is a wide range of finite state automata, as well as of automata-learning algorithms. Vidal et al. (2005a; 2005b) present a survey of different FSAs with a detailed theoretical study. Here we will summarize some of the possible FSA representations and some algorithms to learn them.

A Finite State Automaton is a model of behavior consisting of a finite set of states, an input alphabet, and a transition function that maps input symbols and current states to a next state. It also may have an initial state and a set of final states. Each state stores information about the past, i.e. it reflects the input changes from the system start to the present moment. The automaton evolutes from one state to another depending on the transition function. A transition indicates a state change and is labeled with an alphabet symbol. Furthermore, each transition can have an assigned probability, in this case we will say that the automaton is *probabilistic* (PFSA). If there is only one possible transition for each state with each symbol, the automaton is said to be *deterministic*, since there is only one possible future state given the current state and the transition symbol. When we talk about FSA, we will refer to deterministic automata. If being in a determined state there is more than one state receiving a transition with a given symbol, the automata is *indeterministic*.

Non-statistical Automata Inference

A first group of automata learning algorithms consist of algorithms that infer automata via non-statistical learning. Under this group, we will distinguish between algorithms that learn non-probabilistic automata from those that learn probabilistic automata.

One of the algorithms that learn non-probabilistic automata via non-statistical learning is RPNI (Regular Positive and Negative Inference) (Oncina & Garcia, 1991; Garcia et al.,

2000) that infers regular languages from positive and negative examples. Another example is the one proposed by Trakhtenbrot and Barzdin (1973) and applied to grammatical inference by Gold (1978). Other algorithms of this kind are Error Correction Grammatical Inference (ECGI) (Rulot & Vidal, 1987; Rulot, 1992), which is a heuristic that incrementally constructs a regular grammar from a positive sample set, and Grammatical Inference methods based on morphic generators (Garcia et al., 1987; Segarra, 1993) that recognize a specific class of languages (the local languages) but allow the incorporation of *a priory* knowledge.

Regarding the algorithms that build rules or automata in a non-statistical way but use some statistics to weight the rules or the transitions in an automaton, for example ECGI grammars can be extended introducing statistical information about the usage probability for each rule. There are different ways to do so (N.Prieto, 1995; Rulot, 1992), one of these extensions is named extended ECGI (ECGIE) (Pla, 2000). Another possibility is to estimate the probabilities of context-free rules to build Stochastic Context Free Grammars (SCFG) (Baker, 1979; Lari & Young, 1991).

Statistical Automata Inference

Finally, there are the reconstruction algorithms that build statistical automata, inferring both architecture and parameters, using different techniques. This kind of algorithms are more flexible and are expected to be better for complex NLP tasks where a big amount of data would be necessary to learn the automaton that represents (or approximates) them.

Some algorithms in this class are those that build Variable-Length Markov Models (VLMM). To construct VLMM from sequential data, the "context" algorithm (Rissanen, 1983) and its descendants (Buhlmann & Wyner, 1999; Willems et al., 1995; Tino & Dorner, 2001) can be used. In fact, these algorithms are related to causal state reconstruction algorithms (discussed below), being actually included as a special case of them (Shalizi et al., 2002).

The power of VLMM is that the constructed models take into account histories of different lengths depending on their relevance, which does not happen in traditional HMMs. This is a strength of those models, since they can use long suffixes when necessary without studying all suffixes of that length, what will be computationally very expensive.

VLMMs have been successfully used for example in speech recognition (Nadas, 1984; Jelinek, 1990)

Methods related to VLMMs are those that build Prediction Suffix Trees (PST) (Ron et al., 1994a; Ron et al., 1994b) since it is always possible to transform a PST into a Finite State Automaton. A PST is an extension of Suffix Trees (Galil & Giancarlo, 1988) that introduces probabilities in the edges and states. PST is a tree where each node has a pair (s, γ_s) where s is a string (drawn from an alphabet Σ) associated to the walk starting at that node and ending in the root of the tree, and γ_s is the next symbol probability distribution for string s. From each node there is at most one outgoing edge labeled with each symbol. A walk on the underlying graph of the automaton always ends in a state labeled by a suffix of the sequence. Ron et al. (1994a; 1994b) propose an algorithm to implement a variable length model expanding only the tree branches that add statistically different information that the shorter suffixes.

PSTs have been used in NLP field to learn the structure of English to be used to correct corrupted texts (Ron et al., 1994b) and to tasks such as Part of Speech tagging (Schütze

& Singer, 1994). Similar tree machines were presented in (Rissanen, 1983) and applied to universal data compression (Weinberger et al., 1992; Weinberger et al., 1995)

A recent extension of PSTs are Looping Prediction Suffix Trees (Holmes & Isbell, 2006) which is centered on inferring hidden state when the environment takes the form of a Partially Observable Markov Decision Process.

There are also Utile Suffix Memory (USM) algorithm and its extension UTree (McCallum, 1995b; McCallum, 1995a) that are very similar to VLMM learning methods but with a reward-based stopping criterion that expands a child node only if the distribution of next-step reward appears to differ statistically significantly from that of its parent.

ϵ-machines Inference

Another group of algorithms for statistical automata inference, in which we are specially interested, are those that build ϵ-machines. ϵ-machines are statistical automata formed by the causal states of a process. The causal states of a process are sets of suffixes that have the same probability distribution for the future. They are very powerful and have many desirable properties, as will be introduced in chapter 3. According to (Shalizi & Shalizi, 2004), VLMM are a particular case of ϵ-machines, under restrictive and generally underappreciated assumptions. In fact, causal state methods have some advantages over VLMM methods, beginning with that they are more widely applicable. Furthermore, VLMM are generally more complicated than ϵ-machines.

To build ϵ-machines, there are different kinds of algorithms. On the one hand there are State-Merging ϵ-Machine Inference Algorithms that use what one might call state compression or merging. The initial assumption of these algorithms is that each distinct history encountered in the data is a distinct causal state. Histories are then successively merged into causal states when their morphs are close enough. These algorithms can learn HMMs (Stolcke & Omohundro, 1993) and finite automata (Trakhtenbrot & Barzdin, 1973; Murphy, 1996). Some algorithms in this group are, for example, subtree-merging algorithm (Crutchfield & Young, 1989; Crutchfield & Young, 1990; Hanson, 1993; Crutchfield, 1994) and "topological" merging procedure (Perry & Binder, 1999).

On the other hand, there is the algorithm used in this work, Causal-State Splitting Reconstruction (CSSR) (Shalizi & Shalizi, 2004), also belonging to ϵ-machine inference algorithm class. It induces deterministic automata from sequential data by inferring the causal states of the process (see chapter 3 for details).

Compared to other algorithms for automata learning such as ECGI (or ECGIE), CSSR has the advantage that the learned automata do not depend on the order the sequences appear in the training data and that it is more general because it infers the causal states of processes by studying the future probability distribution for each suffix, so it generalizes the pattern of the sequence using suffix information instead of creating a pattern for each occurrence. Also, since it builds the automaton using statistical information, it is expected to capture better the behavior of the process.

Chapter 3 will introduce the concepts of Computational Mechanics and the CSSR algorithm. Once that has been presented, a more detailed discussion about the similarities and differences specially between CSSR and VLMM and Suffix Trees will be performed. Also the algorithm used in this work will be compared with other algorithms that learn ϵ-machines.

2.3 NLP Sequential Tasks

As we have said in section 2.1, this work is focused on studying the applicability of CSSR algorithm to NLP sequential tasks. In this section, the tasks under consideration in this work (Named Entity Recognition in general and Biomedical domains and Text Chunking) are briefly presented.

There are other NLP tasks that may be interesting to approach with CSSR algorithm. For example, one task that we are planning to study in the future is Part of Speech tagging. Nevertheless, for the moment we focus on tasks that imply detecting subsequences of words in a sentence, since in this case the hidden information is much simpler. To mark a subsequence as a Named Entity (NE) or a chunk, we need just three tags marking if a word is at the beginning of a subsequence, at the end of it or neither of those, or alternatively we can mark if a word is at the beginning, inside or outside an NE. Apart from that we will need the information about which kind of subsequence is it (type of chunk or NE), but in our approach we first perform the recognition of the subsequence. If we want to apply CSSR to PoS tagging, for example, or to other tasks that have more possible hidden tags, it is not possible to use the approach we propose in this work, so other methods have to be explored.

2.3.1 Named Entity Recognition and Classification

Named Entity Recognition and Classification (NERC) task, also known as Named Entity Extraction (NEE), consists of detecting lexical units in a word sequence, referring to concrete entities and of determining which kind of entity the unit is referring to (persons, locations, organizations, etc.). This information is used in many NLP applications such as Question Answering, Information Retrieval, Summarization, Machine Translation, Topic Detection and Tracking, etc. The more accurate the extraction of Named Entities is, the better the performance of the system will be.

NERC consists in two steps that could be approached either sequentially or in parallel. The first step is Named Entity Recognition (NER) which consists of detecting which parts of a text belong to an NE. The second step is Named Entity Classification (NEC) that consists of classifying the NEs into a set of predefined classes (person, location, organization, etc).

Though first systems related to this task date from 1991 (Rau, 1991) the concept of Named Entity was introduced in 1996 within the framework of the Sixth Message Understanding Conference (MUC-6) (Grishman & Sundheim, 1996), devoted to Information Extraction, that included for the first time an NERC task.

After the introduction of this important task, many other conferences included shared tasks and workshops devoted to NERC. Some examples are HUB-4 (Chinchor et al., 1998), MUC-7 and MET-2 (Chinchor, 1998), IREX (Sekine & Isahara, 2000), CoNLL-2002 (Tjong Kim Sang, 2002a) and CoNLL-2003 (Tjong Kim Sang & De Meulder, 2003) shared tasks, ACE (Doddington et al., 2004) and HAREM (Santos et al., 2006).

Framed in these and other conferences, several approaches to Named Entity Recognition and Classification task can be found. Nadeau and Sekine (2007) present a complete survey of some of NERC techniques and of the most used features and evaluation methods. Another interesting survey on a variety of Named Entity Recognition and Classification approaches is presented by Mansouri et al. (2008a).

In next sections we will mention some of these systems dividing them into three classes:

hand-made or knowledge-based systems, supervised learning systems and semi-supervised and unsupervised methods.

Knowledge-Based Systems

This kind of method was specially used at the beginning of the research in NERC systems. These approaches have typically used manually constructed finite state patterns, or rule sets. Some examples of those systems are found in MUC-6 and MUC-7 conferences or in subsequent related work (Appelt et al., 1995; Weischedel, 1995; Krupka & Hausman, 1998; Humphreys et al., 1998; Black et al., 1998; Aone et al., 1998; Mikheev et al., 1998; Mikheev et al., 1999) for English NERC. More recently, Budi and Bressan (2003) propose an NERC method based on association rules, a technique widely used in data mining field.

The performance obtained with such systems is usually high, however, rule-based approaches lack the ability of coping with the problems of robustness and portability. Furthermore they need a great human effort to adapt to each new language.

Supervised Learning Systems

Supervised Learning techniques are currently the most widely used for NERC task. First approaches include the use of variants of Hidden Markov Models (Bikel et al., 1997), Decision Trees (Sekine, 1998), Maximum Entropy Models (Borthwick et al., 1998; Borthwick, 1999) or a variant of Brill (1995) transformation-based rules (Aberdeen et al., 1995). Besides, some hybrid approaches combining machine learning techniques with gazetteer information or hand-made rules were tested in MUC-7 conference (Yu et al., 1998; Borthwick et al., 1998; Mikheev et al., 1998) .

More recent supervised approaches to NERC task were framed in CoNLL-2002 (Tjong Kim Sang, 2002a) and CoNLL-2003 (Tjong Kim Sang & De Meulder, 2003) shared tasks. Both tasks consisted of detecting and classifying NEs, on texts written in different languages. All the participant systems used Supervised or Semi-Supervised Learning approaches.

Some examples of supervised methods used for NERC presented both in these and other conferences are Support Vector Machines (McNamee & Mayfield, 2002; Mayfield et al., 2003; Asahara & Matsumoto, 2003; Mansouri et al., 2008b), decision trees (Black & Vasilakopoulos, 2002), transformation-based learning (Black & Vasilakopoulos, 2002; Florian et al., 2003), Hidden Markov Models (usually combined with other systems) (Jansche, 2002; Burger et al., 2002; Malouf, 2002a; Zhou & Su, 2002; Florian et al., 2003; Klein et al., 2003; Mayfield et al., 2003; Whitelaw & Patrick, 2003), Maximum Entropy Models (Malouf, 2002a; Chieu & Ng, 2002; Chieu & Ng, 2003; Bender et al., 2003; Curran & Clark, 2003; Florian et al., 2003; Klein et al., 2003), memory-based learning (Tjong Kim Sang, 2002b; Cucerzan & Yarowsky, 2002; Meulder & Daelemans, 2003; Hendrickx & van den Bosch, 2003), AdaBoost (Carreras et al., 2002; Tsukamoto et al., 2002; Wu et al., 2002; Carreras et al., 2003b; Wu et al., 2003), Winnow techniques (Zhang & Johnson, 2003; Florian et al., 2003), perceptrons (Carreras et al., 2003a), Conditional Random Fields (McCallum & Li, 2003), neural networks (Hammerton, 2003) and combination of various methods (Florian, 2002; Florian et al., 2003; Klein et al., 2003; Mayfield et al., 2003; Wu et al., 2003; Munro et al., 2003).

The problem with Supervised Learning techniques is that large amount of tagged data are necessary to develop accurate systems. Obtaining these data may be very difficult and expensive, specially in the case of minority languages. For that reason, in last years systems using semi-supervised and unsupervised techniques have been developed.

Semi-Supervised and Unsupervised Systems

Semi-supervised methods are those that need some annotated data to learn a first classifier and then improve this classifier using not-annotated data. These methods are specially useful when dealing with tasks or languages with small amount of available data. Here we mention some approaches to NERC task with semi-supervised learning techniques.

Collins and Singer (1999) parse a complete corpus in search of candidate NE patterns. They use the output of a PoS tagger that can mark proper names and also lexical and syntactic information. Another idea is to learn several types of NE simultaneously what allows the finding of negative evidence and reduces over-generation (Cucerzan & Yarowsky, 1999; Collins, 2002).

One of the most widely used semi-supervised technique is bootstrapping (Riloff & Jones, 1999; Cucchiarelli & Velardi, 2001). Some of these systems use hand annotated seeds and others use the output of existing NERC systems to start the learning of their classifier.

In (Ando & Zhang, 2005) a semi-supervised method based on structural learning called SVD-ASO is presented and applied to NERC and Chunkhing tasks obtaining a high performance.

Ji and Grishman (2006) show that an existing NE classifier can be improved using bootstrapping methods. They demonstrate that relying upon large collection of documents is not sufficient by itself. Selection of documents using information retrieval-like relevance measures and selection of specific contexts that are rich in proper names and coreferences bring the best results in their experiments.

Regarding unsupervised learning methods, typical approaches use clustering, lexical resources, lexical patterns or statistics computed on a large unannotated corpus. Some examples are the use of WordNet and topic signatures (Alfonseca & Manandhar, 2002), the identification of hypernyms using the web Evans (2003) or the counting of the number of occurrences of passages that usually precede NEs (Cimiano & Völker, 2005). Also, (Shinyama & Sekine, 2004) present the use of the fact that Named Entities appear, more often than common names, at the same time and related to the same topics in multiple news sources. Etzioni et al. (2005) use Pointwise Mutual Information and Information Retrieval as a feature to assess that an NE can be classified under a given type. In (Kim et al., 2002) a combination of three learning systems (Maximum Entropy, memory-based learning and sparse network of Winnows) is used to build an unsupervised learning model that uses as seeds a small-scale named entity dictionary.

2.3.2 Biomedical Named Entity Extraction

Named Entity (NE) Extraction in medical texts consists of identifying the medical terms of a text and classifying them into different classes.

The ability to extract sequences from a document collection that convey domain specific meaning not only provides the raw material for the creation of domain specific thesauri and ontologies (Sanjuan et al., 2005), but also represents a crucial preprocessing stage for NLP applications that operate over specialized domains (Weeds et al., 2005). However, terminology extraction still remains a challenging task to which statistical, linguistic and hybrid algorithms have all been applied with their differing levels of computational complexity and success rates (Castellví et al., 2001).

This extraction task is complicated by two factors. First, the flexibility with which a term can be defined leads to many competing and sometimes incompatible notions of what

constitutes a term in a particular field. Broadly the definitions range from statistically significant sequences (Spasic et al., 2005) to specific syntactic phrasal groupings (minimal or fully project noun phrases and coordinated conjunctions involving ellipsis) (Jacquemin & Bourigault, 2003). However, while the exact definition is flexible it is not arbitrary, but results directly from the application to which the terms will be employed (Cabré et al., 2005; L'Homme et al., 2003). For example, in constructing a domain specific thesaurus it would be useful to extract the embedded term "NF-kappa B" from the second term in sentence 2.1. However, as a preprocessing stage for further document analysis (such as in Question Answering (Rinaldi et al., 2004a; Rinaldi et al., 2004b)) the complete minimal NP "NF-kappa B activation" is more useful.

The second complicating factor is the level of semantic granularity required for a sequence to be considered a domain specific term. While the name of an organization will always be considered a Named Entity regardless of where it appears, terms are inextricably linked to the corpus they appear in. For example, in sentence 2.1 the phrase "*reactive oxygen production*" is not considered a term from this domain since it is not specific enough. Given a different document collection, with a different domain of focus this same sequence may well be marked as terminological.

(2.1) ***IL-2 gene expression*** *and* ***NF-kappa B activation*** *through* ***CD28*** *requires reactive oxygen production by* ***5-lipoxygenase***

These factors have lead to much success in measuring domain specificity through corpus analysis and a growing understanding of the different features useful in extracting terminology.

There are many different approaches to this task. For example, the least domain-specific methods for term extraction use weighted document frequency counts to measure the probable importance of terms in the domain e.g. C/NC techniques (Spasic et al., 2005). Other approaches are based in corpus frequencies, such as a system that performs a corpus comparison using the range and frequency of word forms (Chung, 2003) or the system developed by Nakagawa and Mori (2003) based on the statistics about the relation between a compound noun and its constituents that are simple nouns.

Other state of the art methods can be found in the framework of the shared task at BioNLP/NLPBA 2004 (Kim et al., 2004) that was devoted to Bio-Entity Recognition Task. In this task, the system that performed best (GuoDong & Jian, 2004) combined HMM and SVM with features specially defined for the task and the second one (Finkel et al., 2004) used Maximum Entropy Markov Models combined also with gazetteers and web resources.

A much simpler approach has been proposed by Ponomareva et al. (2007a), that uses an HMM with poor knowledge obtaining very competitive results. In (Ponomareva et al., 2007b) also a comparison between HMMs and Conditional Random Fields applied to this task is performed.

All this methods perform both recognition and classification of Biomedical NEs. As for NER in general domain, in this thesis we will only approach the recognition task. There are some systems that also perform this task separately, to which we will be able to compare our results. On the one hand, there is the system of Lin et al. (2004) that uses Maximum Entropy Models, and some post-processing techniques to expand the limits of NEs depending on the boundary words. On the other hand, another system combining SVMs (Lee et al., 2004) and some post-processing techniques to extend the boundaries of NEs based in dictionaries is also comparable to our work.

2.3.3 Text Chunking

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, syntactically related words

This task is usually seen as a preparatory step in full parsing, but in fact, for many NLP tasks, having the text correctly separated into chunks is preferred than having a full parse tree, 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 & Marcus, 1995), others deal with NP, VP and PP (prepositional phrase) (Veenstra, 1999). In (Buchholz et al., 1999) a complete 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 & Buchholz, 2000).

Three participant systems of this shared task used rule-based systems (Vilain & Day, 2000; Johansson, 2000; Déjean, 2000), one was implemented using memory-based systems (Veenstra & van den Bosch, 2000) and most participants used statistical systems. The statistical techniques used were maximum entropy based methods (Osborne, 2000; Koeling, 2000) and Markov Models related methods (Pla et al., 2000; Zhou et al., 2000). Finally, the best systems used system combination. Tjong Kim Sang (2000) trained and tested five memory-based learning systems and then combined them buy majority voting. van Halteren (2000) used Weighted Probability Distribution Voting (WPDV) for combining the results of four WPDV chunkers and a memory-based system. Kudo and Matsumoto (2000) presented a combination with a dynamic programing algorithm of 231 support vector machines. An improvement of this system was presented in (Kudo & Matsumoto, 2001).

Posterior systems related to CoNLL-2000 shared task are, for example, (Zhang et al., 2001; Zhang et al., 2002) that use Winnow-related methods and (Carreras & Màrquez, 2003) that uses a perceptrons. Another interesting approach is presented by Thollard and Clark (2002), where Probabilistic Suffix Trees are applied to Chunking task. The data used in that work is the same as used in CoNLL-2000, so all results can be compared.

The semi-supervised method presented by Ando and Zhang (2005), already introduced in section 2.3.1, also obtains good results in chunking task using the CoNLL-2000 data.

Wu et al. (2006) and Lee and Wu (2007) present a general, multi-lingual phrase chunking system based on combining Support Vector Machines with a masking method that introduces a significant improvement of the obtained performance.

Regarding chunking systems devoted to less common languages, an interesting approach for the Korean is presented in (Park & Zhang, 2003). This approach combines a set of hand-written rules with a memory-based learning method that corrects some of the errors of the rule-based system. For Chinese, Li et al. (2003b) present a chunker based on Maximum Entropy Models, Li et al. (2003a) a system based on HMMs and Li et al. (2004) one that uses support vector machines.

CHAPTER 3

Computational Mechanics and CSSR

This chapter introduces the theoretical foundations of Computational Mechanics (CM) necessary to understand the CSSR algorithm and presents the algorithm itself.

Section 3.1 introduces the basic concepts of Computational Mechanics, 3.2 presents the CSSR algorithm with its main issues summarized in section 3.2.3, section 3.4 discusses the differences between CSSR and the similar algorithms already mentioned in 2.2 and section 3.5 overviews some other applications of CSSR algorithm either related to NLP tasks or not. For some examples of how CSSR learns automata from sequential data and discussion about some interesting issues of the algorithm, see Appendix 3.3.

3.1 Foundations of Computational Mechanics

Since various research fields require models of observed processes, there is an important interest on finding the best way to extract patterns from data. Some examples could be time series analysis, decision theory or machine learning. *Computational mechanics*, as defined in (Crutchfield, 1994) aims to understand the nature of patterns and pattern discovery. From either empirical data or from a probabilistic description of a determined behavior, it tries to infer a model of the hidden process that generated the observed behavior. This representation captures the patterns of the process reflecting its causal structure what is useful to predict future behavior. The goal of computational mechanics is to find a predictor of maximum strength and minimum complexity.

In fact, computational mechanics goes further than what is usually called *pattern recognition* in the field of computer science. It is not only concerned about learning the patterns but also about which is the best way to represent them.

This chapter introduces the basic theoretical ideas related to the part of computational mechanics we are interested in. These are necessary concepts to introduce the algorithm used in this work.

3.1.1 Definitions

Next sections define some important concepts of computational mechanics. For these definitions, we use the convention of using capital letters to denote random variables and lower-case letters for their particular values. For more details see (Shalizi & Crutchfield, 2001) or (Crutchfield, 1994).

Process, History and Future

A *process* is a sequence of random variables (A_i) taking values from a discrete alphabet Σ determined by a given probability distribution. Thus, a process is a sequence with the form $A_0,\ A_1,\ A_2,\ ...,\ A_T$.

Given a process, we can define a *history* or *past* of length L (X_i^L) as any subsequence of finite length L preceding the i^{th} element $(X_i^L = A_{i-(L-1)},\ A_{i-(L-2)},\ ...,\ A_{i-1})$. Given a history, it is possible to determine the probability of each possible following sequence, called *future* (Z_i^L) determined by the probability distribution of the process. A future is the sequence of length L following (and including) the i^{th} symbol: $Z_i^L = A_i,\ A_{i+1},\ ...,\ A_{i+L-1}$.

We say that a process is *stationary* if and only if the probability of seeing any subsequence is time invariant, so if it does not depend on the index i. This is equivalent to say that $P(Z_i^L = z^L) = P(Z_0^L = z^L)$ for all possible i and L.

From now on, we will consider stationary processes, so we will ignore the index i and refer to histories and futures as X^L and Z^L, or more generally as X^- and Z^+. Then, our goal is to predict all or part of the future Z^+ using some function of some part of the past X^-.

Equivalent Histories and Causal States

Given a concrete history x^- there is a determined probability distribution of seeing a future Z^+ after this history: $P(Z^+|x^-)$. 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 build the equivalence classes that are defined by a function ϵ:

$$\epsilon(y^-) \equiv \{x^- \mid P(Z^+|x^-) = P(Z^+|y^-)\}$$

Those equivalence classes are groups of histories sharing future distributions and are named *causal states* of the process.

Each causal state s_i has the following information associated:

1. The label (i) of the state.
2. The set of histories belonging to this state.
3. The future probability distribution $P(Z^+|s_i)$, that is equal to $P(Z^+|x^-)$ for all histories x^- belonging to this state s_i.

Causal states have many desirable properties that make them the best possible representation of a process. Some of the main properties are that they are minimal (in the sense that any other representation of the process with equivalent predicting power will be more complex) and have sufficient statistics to represent a process, this is, from causal states it

is possible to determine the future for a given past, so they define correctly and minimally a process. These, and other properties and theorems are proven in (Shalizi & Crutchfield, 2001). Here we summarize the main conclusions and properties:

- Causal states are maximally accurate predictors of minimal statistical complexity what makes them the best possible representation of the patterns of a process.
- Causal states are as good at predicting the future as complete histories.
- The causal states of a process are sufficient statistics for predicting it.
- Causal states are minimal, so any other representation of the process, which is as good as predicting the observations as the causal states, will be more complex
- Causal states are unique
- The causal states of a process form a deterministic machine and are recursively calculable.
- A process's causal states are the largest subsets of histories that are strictly homogeneous with respect to futures of all lengths[1].

Causal State to State Transitions and ϵ-machines

At any given time, the current causal state and the next observed value in the sequence determine the next causal state. Furthermore, given the current causal state, all the possible next values of the future (Z^+) have well defined conditional probabilities. Thus, given a set of causal states representing a process and their future distribution probabilities, we can define a state-to-state transition matrix ($T_{ij}(a)$) for each symbol a, that gives the probability of making the transition from state s_i to state s_j with a determined symbol a.

Also, we can define a machine that combines the set of causal states and the transition matrix $T_{ij}(a)$ that represents the process. This machine is called *ϵ-machine* (Crutchfield, 1994; Upper, 1997). The ϵ-machine is thus a causal representation of all the patterns of the process. Furthermore, it is maximally predictive and minimally complex as causal states are. Here we summarize the most important properties of ϵ-machines:

- ϵ-machines are deterministic so, given a state s_i and a symbol a there is at most one state s_j receiving the transition from state s_i with this symbol. This implies that for every history $x^- \in s_i$ the history x^-a is a history of state s_j.
- ϵ-machines are markovian, what means that the causal state at time t is determined by the state in time $t-1$, being independent of the previous causal states. This gives a basis to the claim that the causal states can be considered a generalization of Markovian states tough, in fact, ϵ-machines have richer properties than Markov processes (Kemeny & Snell, 1976; Kemeny et al., 1976), as will be argued in section 3.4.1.

ϵ-machine Reconstruction

ϵ-machine reconstruction is any procedure that given a process produces the ϵ-machine that represents it.

[1] A set X is strictly homogeneous with respect to a random variable Y when the conditional distribution $P(Y|X)$ for Y is the same for all measurable subsets of X.

ϵ-machines can be analytically calculated from given distributions or systematically approached from empirical data. In the first case, a mathematical description of the process is needed. In the second case, the only evidence of the process available are some data sequences and it would be necessary to empirically estimate the probability distribution of the process. There are plenty of algorithms that performs this estimation and the reconstruction of the ϵ-machine from them. Some, such as those used in (Crutchfield, 1994; Crutchfield & Young, 1990; Crutchfield & Young, 1989; Hanson, 1993) take the raw data as a whole and produce the ϵ-machine. Others, as the one used in this work, could operate incrementally, taking individual measurements and re-estimating the set of causal states and their transition probabilities. See section 2.2 for more algorithms on ϵ-machine reconstruction.

3.1.2 Assumptions of the Theory and Limitations of Causal States

All we have presented can be formally proven if some restrictive assumptions are made (Shalizi & Crutchfield, 2001; Crutchfield, 1994). Here we summarize these assumptions:

- The observed process is formed by sequences of discrete symbols from a closed alphabet, so it will not be valid for processes with continuous variables.
- The process is discrete in time.
- The process is a pure time series, e.g. without spatial extent.
- The observed process is stationary.
- Prediction can only be based on the process' past, not on any outside source of information.

These assumptions are necessary to prove the theorems and lemmas behind causal state theory but there are methods and techniques to relax them. In section 3.1.3 we will discuss which of these assumptions affect us and how CSSR algorithm deals with them.

3.1.3 Summary and Important Remarks

Considering all properties of causal states and ϵ-machines it can be said that, ϵ-machines fit the goal of computational mechanics since they represent the patterns of a process. Furthermore these machines do so in the best way possible: with maximal accuracy in the prediction and minimal statistical complexity. Since ϵ-machines can be built via ϵ-machine reconstruction algorithms, it can be claimed that ϵ-machine reconstruction is the best way to discover the patterns of a process.

These properties can be demonstrated under some assumptions (summarized in section 3.1.2) that seem reasonable. It seems clear, then, that causal states are a good way to represent processes that fulfill these assumptions or that can be easily modified to fulfill them.

The problem is, then, how do we generate these ϵ-machines from real processes? There are a variety of algorithms that do so, but will they always be able to learn the correct ϵ-machine of any process that fulfills the assumptions of the theory? Of course not, since if we are dealing with real processes, we will have a limited amount of data that may not be enough to build a correct representation of the process that generated these data.

Furthermore, if the data is taken experimentally, there could be noise or errors in the patterns we want to learn.

Nevertheless, using ϵ-machines reconstruction algorithms is expected to lead at least to as good results in representing data as other techniques. In fact, we can expect to take advantage of the nice properties of the causal states even if the learned set of states from data is not exactly the one that represents the process behind the data.

In next section, we discuss which are the main modifications to the theory that the algorithm used in this work (CSSR) does to learn the causal states of a process from data. Section 3.2 will present the algorithm itself.

In the Real World

First approximation that must be done to create the causal states given some data regards histories and futures. Theoretically, to build causal states we consider all possible histories and futures from length 0 to infinite. Obviously, we could not consider infinite length for suffixes when implementing an algorithm so what CSSR does is, on the one hand, consider histories as suffixes of length from 0 to a preestabished maximum length l_{max} and on the other hand, consider futures of length one. Thus the probability distributions that will be used by the algorithm will have the form $P(A|x^-)$, where A is any symbol of the alphabet (future of length one) and $x^- = A_0A_1...A_{l-1}$ with $0 < l < l_{max} - 1$ (history of length l_{max}).

Another important issue to take into account is how we define that two probability distributions are equal. If we are dealing with real data, we can estimate the probability of seeing each symbol after a determined suffix, but due to the limited amount of data, we can not expect to have two histories with exactly the same probability distribution for the future. In this case, what we have to do is to perform a hypothesis test to compare two distributions and determine, with a certain confidence degree, if two histories can be said to have different future distributions or not.

Then, each causal state is a set of suffixes that represent histories of length from 0 to l_{max} with probability distributions for the future that can not be considered different according to a determined hypothesis test with a certain confidence degree.

3.2 CSSR algorithm

Causal-State Splitting Reconstruction (CSSR) (Shalizi & Shalizi, 2004) is an algorithm that infers the causal states of a process from sequential data. 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.

Another important parameter of the algorithm is the confidence degree α of the hypothesis test performed to determine if two histories can be considered to belong to different causal states or not. Also, the hypothesis test used can modify the obtained results.

3.2.1 The Algorithm

To build the causal states of a process, CSSR algorithm studies the data sequence produced by the process. The input and output of the algorithm are:

- **Input:** Data sequence (d), alphabet (Σ), maximum length of the studied suffixes (l_{max}), confidence degree for the hypothesis test (α).

- **Output:** Automaton (Q) as a set of causal states (containing each state different suffixes) and the transition matrix T (with transition probabilities).

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 state set is not sufficient. The causal state machine is built in three phases:

1. **Initialize:**

   Set the machine (Q) to one state (q_0) containing only the null suffix (λ). 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). Before doing so it is necessary to estimate the probability distribution $\hat{P}(X_t|S = s)$ (where X_t is the random variable for the next alphabet symbol in the sequence) for each state. This is necessary because this probability can change at each iteration when new suffixes are added to a determined state. This probability distribution is estimated (via maximum likelihood, for instance) using the data ($\bar{x}$).

   At this phase, the suffix sons (ax) for each longest suffix (x) are created adding each alphabet symbol (a) at the beginning of each suffix. The future distribution $P(X_t|X_{t-l}^{t-1})$ (probability of each alphabet symbol given the last l symbols) for each son is computed and a hypothesis test with the following null hypothesis is performed,

   $$P(X_t|X_{t-l}^{t-1} = ax) = P(X_t|S = s); \quad \forall a \in \Sigma$$

   If this hypothesis is not rejected, the new distribution is considered to be equal (with a certain confidence degree α) to the distribution of an existing state (s). In this case, the suffix son is added to this state. In fact, the first state for which the null hypothesis is tested is the state containing the suffix x (parent state). If the parent state fails the test, then the test is performed over the other states. If the hypothesis is rejected for all states, a new state for the suffix son is created. To check the null hypothesis we can use a statistical test such as χ^2 or Kolmogorov-Smirnov.

   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 can be significantly different depending on the chosen l_{max} value, since the larger this value is, the more training data will be necessary to learn a correct automaton with statistical reliability. Also, the time needed to learn the automaton grows linearly with l_{max}. So it is necessary to tune the best maximum length for the amount of available data (or vice versa, the amount of necessary data for the desired suffix length).

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.

At the end of this recursion phase, a deterministic automaton is obtained.

In figure 3.1 the pseudo code for this algorithm is presented and section 3.3 shows how this algorithm builds the automaton with some simple examples.

3.2.2 Time and Space Complexity

Shalizi and Shalizi (2004) show that the time complexity of this algorithm is $O(k^{2l_{max}+1}) + O(N)$, where k is the size of the alphabet, N the number of data points and l_{max} the maximum length of the studied histories. Thus, the time complexity is linear in the data size N, since the size of the alphabet (k) will be constant (and usually not very large) and the high exponent in k is reached only in extreme cases: every string spawns its own state, almost all states are transient, etc. According to some experiments reported by Shalizi and Shalizi (2004), CSSR tends to be much faster than this worst-case result.

Regarding the space complexity, it is given by the number of histories the algorithm has to store. Since only the histories of length $l_{max} - 1$ and l_{max} are relevant, the maximum number of histories in the automaton will be $k^{l_{max}-1} + k^{l_{max}}$. Nevertheless, as we will explain in section 3.3, not all the suffixes of length $l_{max} - 1$ are stored, so this limit will be rarely reached.

3.2.3 Important Issues of CSSR

One of the main parameters of this algorithm is the maximum length (l_{max}) of considered suffixes which is the length of the largest studied histories. To properly capture a process' patterns there is a minimum l_{max} necessary depending on the length and complexity of the patterns. Nevertheless, when a limited amount of data is available, they may not provide enough statistical significance for CSSR lo learn the automata with this ideal maximum suffix length.

Then, when using CSSR, it is necessary to reach a trade off between the amount of data (N), the vocabulary size (k) and the chosen maximum length (l_{max}). The size of the alphabet is important because it sets the number of suffixes that must be studied, which are all the combinations of l_{max} symbols that can be done with the k different symbols of the alphabet: $k^{l_{max}}$. If we consider that this is the minimum length that the data sequence has to have to have seen each possible suffix at least one time, the maximum length that can be used with statistical reliability with a sequence of N symbols is $l_{max} \leq \log N / \log k$ (Shalizi & Shalizi, 2004). This is a very basic way to set the maximum usable length, but it has to be seen as an upper limit that for sure we could not pass and learn good automata.

This trade off is very important when using CSSR to learn complicated patterns, specially if the amount of data is limited, as it can happen in NLP tasks. For these kind of tasks, probably the maximum length that can be used will be limited by the amount of available data, so perhaps the longest patterns present in data will not be captured properly. On the other hand, we can try to compensate this limitation by maintaining small the size of the alphabet.

```
Algorithm CSSR (d, Σ, l_max, α)          // Input Parameters: Data sequence, alphabet, maximum
                                         // length of the suffixes, confidence degree for the hypothesis test

l ← 0                                    // Initialization: initialize the machine with a state with
q_0 ← {λ}; Q ← {q_0}                     // only the null suffix.

while l < l_max                          // Sufficiency: build causal states.
  for each s ∈ Q
    estimate P̂(X_t|S = s)                // Estimate the probability distribution for next symbol given the
    for each x ∈ s                       // current state since it may have changed in the last iteration.
      for each a ∈ Σ
      estimate p ← P̂(X_t|X_{t-l}^{t-1} = ax)   // Estimate the probability distribution for this
      Reorganize_States(Q, p, ax, s, α)        // suffix son and perform the hypothesis test.
  l ← l + 1

Remove transient states from Q           // Recursion: remove transient states, makes the machine
repeat                                   // deterministic and fills transition table (T[state, symbol])
  recursive ← True
  for each s ∈ Q
    for each b ∈ Σ
      x_0 ← first x ∈ s
      T[s, b] ← Class(x_0 b, Q)          // Look for the transition for this suffix with this symbol.
      for each x ∈ s, x ≠ x_0
      if Class(xb, Q) ≠ T[s, b]          // If the transition for another suffix goes to a different
      create new state s' ∈ Q            // state, creates a new state with this suffix and move
      T[s', b] ← Class(xb, Q)            // to this state all the suffix with the same transition.
      for each y ∈ s | Class(yb, Q) = Class(xb, Q)
      Move_Suffix(y, s, s')
      recursive ← False
until recursive

return (Q, T)                            // Output: Automaton, transition matrix

function Reorganize_States(Q, p, y, s, α)   // Reorganize_States: test the null hypothesis and decide to
                                            // which state a suffix must be added or create a new state.
  if null hypothesis passes a test of size α for s   // If the probability distribution for y is equal to
    s ← y ∪ s                                        // the s distribution add y to this state.
  else
    if null hypothesis passes a test of size α       // If the probability distribution for y is equal to the s* ≠ s
      for s* ∈ Q, s* ≠ s                             // distribution, add y to this state s*.
      s' = s*
    else                                             // If the probability distribution for y is different from that
      Q ← s'                                         // of all states, create a new state and add y to it.
    Add_Suffix(y, s')

function Add_Suffix(y, s)                // Add_Suffix: add suffix y to state s.
  s ← s ∪ y
  re-estimate P̂(X_t|Ŝ = s)

function Class(y, Q)                     // Class:return the causal state (equivalence class)
  return (s ∈ Q|y ∈ s)                   // a suffix belongs to.

function Move_Suffix(y, s_1, s_2)        // Move_Suffix: move a suffix from one state to another.
  s_1 ← s_1 \ y
  re-estimate P̂(X_t|Ŝ = s_1)
  s_2 ← s_2 ∪ y
  re-estimate P̂(X_t|Ŝ = s_2)
```

Figure 3.1: Pseudo code for the CSSR algorithm.

3.3 CSSR Algorithm Step by Step

In this section, we illustrate with examples how CSSR learns automata from sequential data. We will use this examples also to explain some very specific parts of the algorithm that has not been presented in the algorithm description.

First of all two simple processes will be studied: the "even process" and the "$(a^n b^n)^*$ process" that are correctly learned by CSSR and are useful to understand how the algorithm works. Secondly we will present what we call the "pseudo-even process", that introduces some interesting problems to be studied when CSSR is used to learn its patterns.

3.3.1 The Even Process

The even process is a simple process used in (Shalizi & Shalizi, 2004) to illustrate how CSSR algorithm works. This is an interesting example because this process can be described with a very simple automaton but since it may generate infinite sequences, it is not easy to learn with some pattern learning algorithms.

Given an alphabet $\Sigma = \{0, 1\}$, the even process is a process where the sequences always have an even number of 1's between two zeros. Then, the subsequences as 010, 01110, 0111110, etc, are forbidden. The automaton representing this simple process can be seen in figure 3.2.

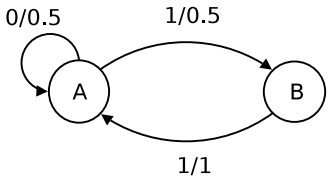

Figure 3.2: Automaton representing the even process

Given some data sequences generated by this process, CSSR can be used to learn an automaton that represents these data. Following sections explain carefully the steps of this algorithm applied to these data.

Initialize

In this step, a single state with the null suffix (λ) is created (figure 3.3) and its transition probabilities for the future computed. In initialize step, these probabilities are the occurrence probabilities of each symbol. In all this example, the shown probabilities are rounded to optimal values though in fact these values will only be obtained with infinite data. The actual obtained values are closer to those shown here as the amount of data grows.

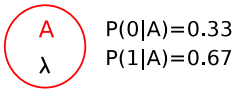

Figure 3.3: State created in the initialization step

Sufficiency

Sufficiency step increases the length of each suffix iteratively until l_{max} is reached. For this example the maximum length used is 3, which is the minimum necessary length to learn the automaton correctly.

First iteration builds suffixes of length 1, computes their distributions and since they are different from distribution of state A and are also different between them, two new states (B and C) are created, as shown in figure 3.4.

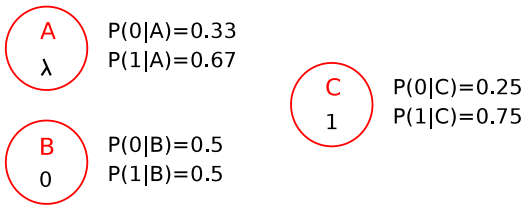

Figure 3.4: Causal states set with histories of length 1

At the end of this iteration, the state A is erased because the original suffix λ creates two sons (0 and 1) that are splitted into different states. Since the sons are longer histories, the future probability estimated for them is expected to be more accurate than the one for the parent. Since these histories contain more accurate information about the future, and they are in two different states, parent history has to be deleted to avoid having repeated information in different states with different futures, what could lead to contradictions.

Next iteration ($l = 2$) enlarges the existing suffixes 0 and 1 to the left and studies the probability distribution of new suffixes. The suffixes 00 and 10 have the same (or similar) probability distribution for the future than state B so they are added to this state. Suffix 01 has a different distribution so a new state (D) is created with this suffix. Suffix 11 has the same probability distribution than the null suffix but as it was removed, a new state (E) for this suffix is also created. The obtained causal states are shown in 3.5.

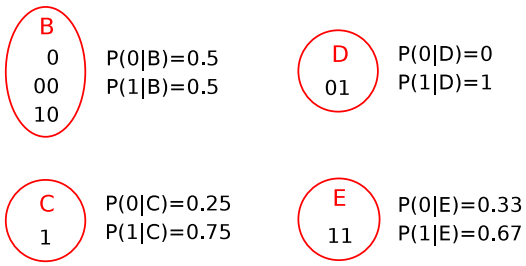

Figure 3.5: Causal states set with histories of length 2

As for $l = 1$, here it can be seen that suffix 1 has its two sons 01 and 11 in two different states, so this suffix has to be deleted since its sons encode more information and are separated in two different states. Since this is the only suffix in state C, this state is deleted and the set of causal states at the end of this iteration has 3 states: B, D, E.

The last iteration creates and studies the suffixes of length 3. At this step the only suffix that creates a new state (F) is 111, the other suffixes share distributions with states B and D as shown in figure 3.6.

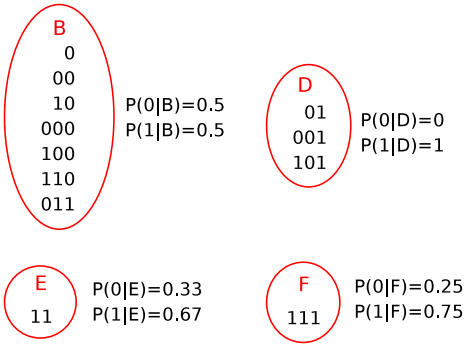

Figure 3.6: Causal states set with histories of length 3

Again, suffix 11 has created two sons 011 and 111 that are in different states so this suffix and the state it belongs to (E) are deleted. Furthermore, the suffixes of length smaller than $l_{max} - 1$ can be ignored, since all the information is encoded in states with longer suffixes. The suffixes of $l_{max} - 1$ are maintained because they will be necessary to determine the transitions between state, as we will explain in the recursion step.

Thus, at the end of the sufficiency step the three causal states showed in figure 3.7 with their different future distributions have been created. From this set of causal states, next step of the algorithm (recursion) will create a deterministic automaton deleting the transient states and splitting the state until the machine is deterministic.

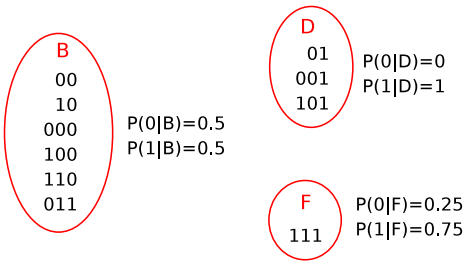

Figure 3.7: Causal states at the end of sufficiency step for the even process with $l_{max} = 3$

Recursion

1- Remove transient states

Once the causal states are created, it is necessary to determine the transient states. To do so, the transitions from the suffixes belonging to each state should be studied. Each suffix is lengthened to the right with each symbol and the state the new suffix belongs to is the state receiving these transition. A state receiving a transition is a non transient (so recurrent) state. The states that do not receive any transition from other states (different from themselves) will be considered transient and for that reason deleted. This process has to be repeated while some states are deleted, since a state receiving transitions just from transitory states is also transient.

This is the basic idea of the recursion process, but when this step is performed, just the suffixes of length $l_{max} - 1$ are taken into account to study which states receive transitions. This is because, in order to check the 'correct' transitions, we append a future symbol to each history (suffix) and search for that extended suffix in our set of causal states. However, for the longest suffixes, we actually do not know where they transition to as they will transition to a history of $l_{max} + 1$. We need the longest suffixes because we need to check where the suffixes of length $l_{max} - 1$ transition to, but all we can do for l_{max} suffixes, is to 'wrap' them, i.e. to append the new future symbol, and chop off the oldest past symbol. However, we can not be sure that this gives us the correct information. According to Kristina Klinkner, formerly Kristina Shalizi (private communication), this is the general problem referred to as 'data closure'.

For example, in the even process, the history 011 will be considered to evolute to history 111 with symbol 1, but then we lose the information about parity of the number of 1's we have seen, as from 111 we will stay in the same suffix with symbol 1. So we would have the transitions for an even number of 1's mixed with the transitions of an odd number of ones. In fact, the state defined by all 1's is a synchronization state, i.e. if all we have ever seen since the beginning of time is all 1's, we do not know which recurrent causal state we are in. Once we have seen the first 0, we will be in state B, and from now on, the system will change between states B and D, but will not never go to state F anymore. Thus, we do not need F state to reproduce the data.

Nevertheless, there is an exception to this rule of looking just the transition for shortest suffixes that happens when in one state there are only suffixes of length l_{max}. In this case, at least one of these suffixes is also used to compute the transitions, because this is the only way we can obtain a transition for this state.

Figure 3.8 shows the considered transitions for the even process. It can be seen, then, that state F is a transient state because it does not have any incoming transition. States B and D both have incoming transitions so they are recurrent and are the two states that will be taken into account in the determinization step.

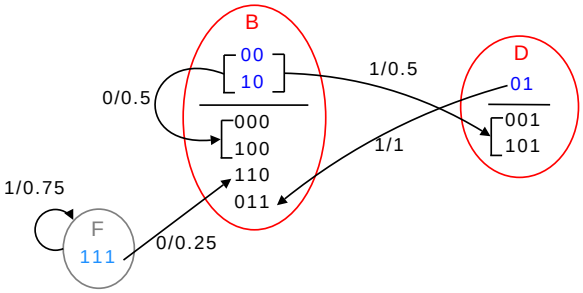

Figure 3.8: Studied transitions to determine the transient states of the even process with $l_{max} = 3$.

2- Determinization

Once the transient states are deleted, it is necessary to determinize the automaton. The transitions for each suffix are studied and if two suffixes in the same state have different target states they are splitted into two different states. In the case of the even process the determinization step does not change anything since the non defined transition of suffix 011

with symbol 1 is generalized to be the same as the other suffixes in the state. The obtained automaton is shown in figure 3.9.

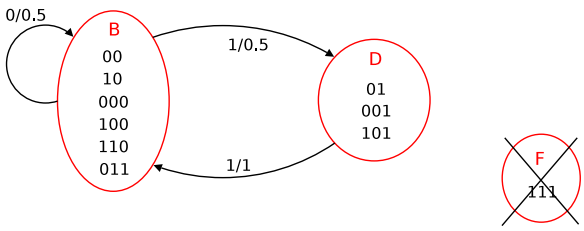

Figure 3.9: Learned automaton with $l_{max} = 3$

As it can be seen, this learned automaton is the same as the automaton that represents the process. Obviously, when using CSSR with real data for the even process, the values of the probabilities may not be exactly as the theoretical ones, but if enough data are used, they will be very close to them.

If the algorithm is run with larger l_{max} , the obtained results are always the same (for $l_{max} > 2$) as there is always a state with the suffix containing just 1's that is considered transient (and deleted) and other two states, one with the suffixes finishing with 0 or an even number of 1's and another one with suffixes finishing with an odd number of 1's.

3.3.2 The $(a^n b^n)^*$ Process

Given an alphabet $\Sigma = \{a, b\}$, we call $(a^n b^n)^*$ process the process that generates sequences of this symbols with the property that given a subsequence of n symbols a, the same number of b symbols will follow. Then, the sequences generated by this process are combinations of strings such as *aabb*, *ab*, *aaabbb*, etc. This process can only be represented by a finite automaton if the maximum n is fixed, so there must be always a limitation to the number of a that can be seen together. In this example, we use $n = 4$ what leads to a process that can be represented by the automaton in figure 3.10

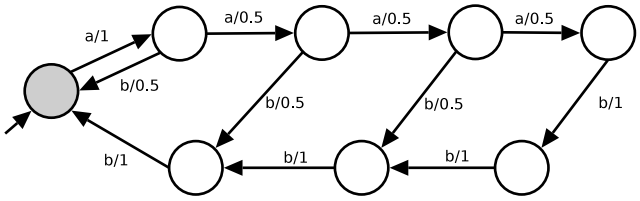

Figure 3.10: Automaton representing $(a^n b^n)^*$ process with $n \leq 4$

As in section 3.3.1, using data sequences generated by this process CSSR can learn an automaton that represents the process. In next sections we will explain the outputs of the steps of the algorithm.

Initialize

This step is equivalent to the initialization of the even process, but with a different probability distribution (see figure 3.11).

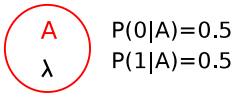

Figure 3.11: State created in the initialization step

Sufficiency

To learn correctly the automaton that represents this process with $n = 4$ with CSSR, it is necessary to use $l_{max} = 7$. Sufficiency step will iteratively create each suffix of length from 1 to 7 and study their probability distribution in order to determine the causal states. The process is equivalent to the one explained in the even process example, so we will not explain each iteration here. What is interesting to see is the causal state set obtained at the end of sufficiency step, once the shortest suffixes have been removed. It is represented in figure 3.12

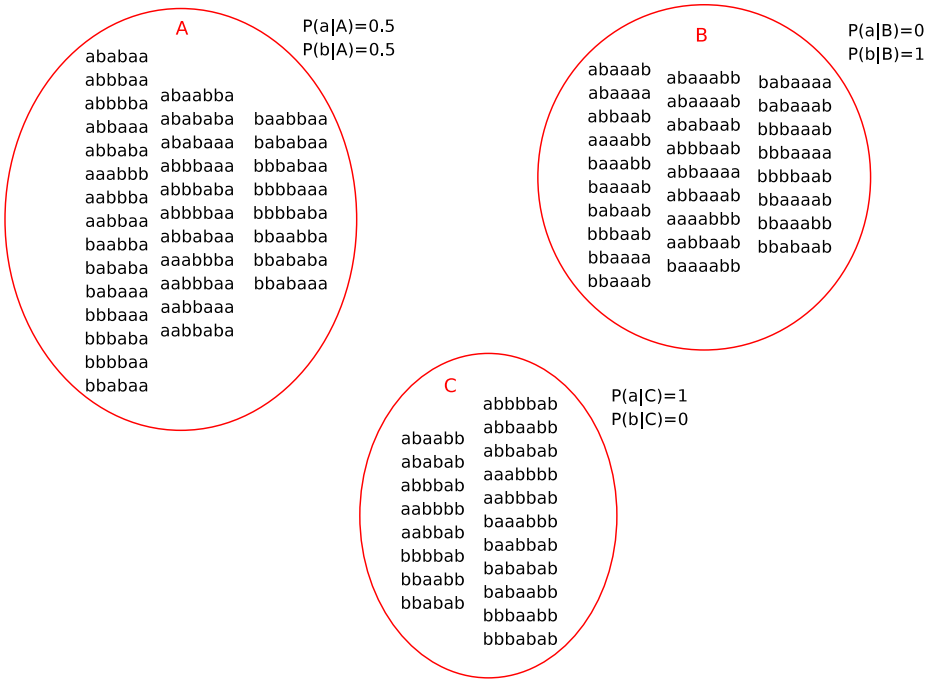

Figure 3.12: Causal states for $(a^n b^n)^*$ process after sufficiency step with $l_{max} = 7$.

As it can be seen in this figure, this process has just 3 causal states: state A containing the suffixes finishing with less than 4 a symbols, from which both a and b can be seen with the same probability, state B with the suffixes finishing with a number of b's inferior to the number of preceding a's, from which just b symbols will be seen, and state C with suffixes finishing with n a's followed by n b's from which just a symbol can be seen, as the system has already generated a whole sequence and is again at the beginning state. Note that, since this is the definition of causal state, here the suffixes are joined just depending on their future distribution. Given these causal states, the recursion step will create an

automaton that reproduces the process patterns.

Recursion

1- Remove transient states

For this process, the set of causal states generated by CSSR does not have any transient states. If all suffixes of $l_{max} - 1$ are extended with both symbols a and b, it can be seen that all states receive at least one transition from another state.

2- Determinization

Next step is to determinize the automaton. To do so, the transitions for each suffix are studied and if two suffixes in the same state evolute to different states with the same symbol, they are separated into two different states.

In the case of $(a^n b^n)^*$ process, this determinization process will split state A into three states corresponding to the three states that have possible transitions with symbols a and b. Each of these three states correspond to suffixes finishing with one, two or three a's. State B will be splitted into the four states that can just emit a b symbol, one for the suffixes finishing with four a's, and three for suffixes ending with b's but lacking one, two or three b's to equal the number of preceding a's. Finally, state C will remain as a unique state, which is the initial and final state. Figure 3.13 shows the final automaton learned using CSSR, that is the correct automaton for this process.

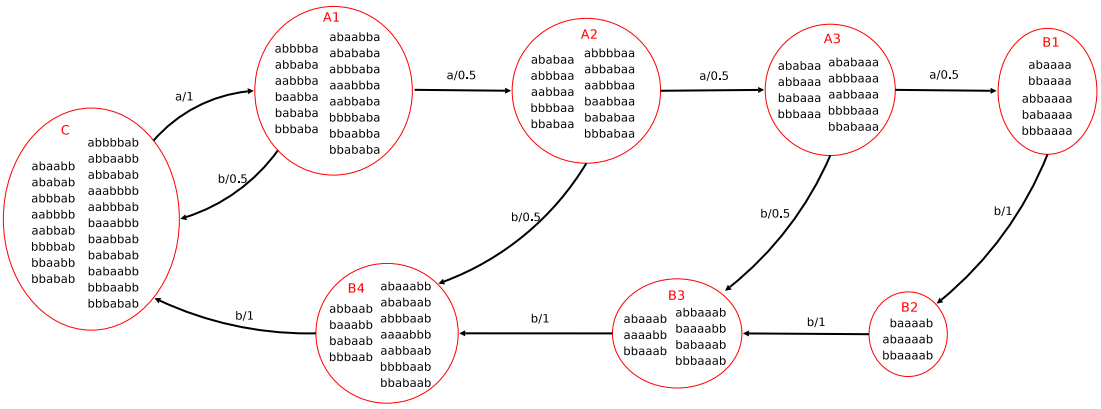

Figure 3.13: Final automaton learned with CSSR for $(a^n b^n)^*$ with $n = 4$ using $l_{max} = 7$

3.3.3 Pseudo-Even Process

The third example presents a process similar to the even process, but with finite length patterns. We call this example the "pseudo-even process".

A pseudo-even process is one that forbids an even number of 1's between zeros if this number of 1's is smaller or equal than a determined threshold (t), but allows any number of 1's if this is higher than the threshold. For example, if the threshold is set to 3, it means that the sequences 010 and 01110 are forbidden, but all other sequences are permitted. Figure 3.14 shows the automaton representing this process for the case of $t = 3$.

If the threshold rises, the number of states grows, but the idea of the automaton is the same.

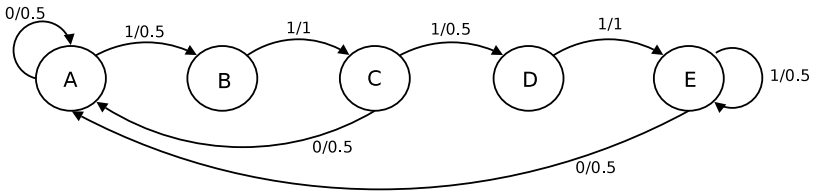

Figure 3.14: Automaton representing pseudo-even process with $t = 3$

Various experiments using CSSR for this process were performed with different thresholds and l_{max} values. Here we summarize the results obtained for $t = 3$ and $l_{max} = 4$.

Sufficiency

The initialization and sufficiency steps are very similar to the even process but in this case we have just two causal states: one for the suffixes from where either 0 or 1 can be seen (suffixes finishing with 0, an odd number of 1's or more than three 1's) and another state for the suffixes finishing with one or three 1's, from where just another 1 can occur. Figure 3.15 shows the causal states obtained at the end of sufficiency step with $l_{max} = 4$. There are some suffixes not appearing in this set of causal states. This can be for two reasons: either they are impossible suffixes, not seen in the data, or, in the case of short suffixes, they have been deleted because their sons are in different states.

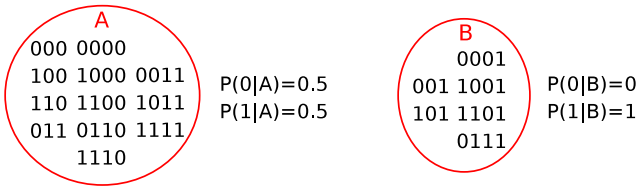

Figure 3.15: Causal states obtained at the end of sufficiency step for the pseudo-even process with $t = 3$ and $l_{max} = 4$

Recursion

1- Remove transient states

In this case, if we study the possible transitions of the shortest suffixes in the learned causal states set, it will be seen that both states are recurrent since both receive transitions from each other, as shown in figure 3.16.

2- Determinization

The next step of the algorithm is to determinize the automaton. Unlike the automaton learned for the even process, this automaton is not deterministic. Although when studying just the shortest suffixes it seems deterministic, in this step CSSR studies the transitions for all suffixes, and in this case we will find that suffix 1111, in state A evolutes to itself with symbol 1 what is different from other histories in state, that go to state B with this

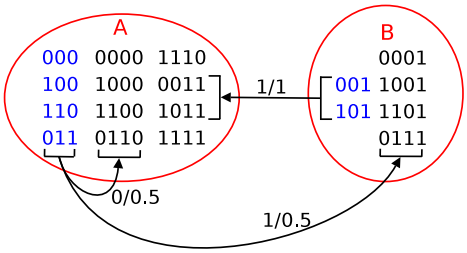

Figure 3.16: Studied transitions to determine the transient states of the pseudo-even process with $t = 3$ and $l_{max} = 4$.

symbol. So, this suffix will be separated in a different state. Then the process has to be repeated until every suffix in every state have the same transitions for both symbols, what will lead to the automaton in figure 3.17.

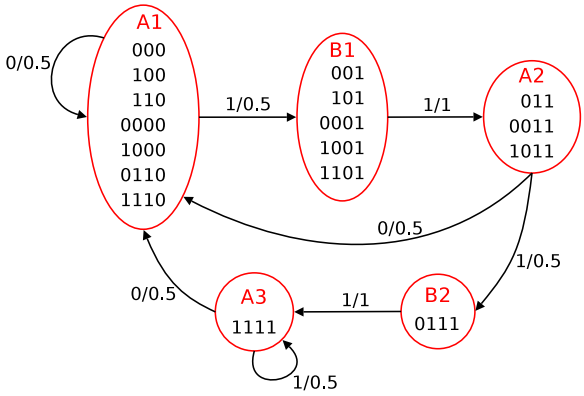

Figure 3.17: Final automaton learned with CSSR for the pseudo-even process with $t = 3$ and $l_{max} = 4$.

3.3.4 Discussion

For the pseudo-even process, we performed several experiments with various thresholds and l_{max} values and from the obtained results it could be seen that, as it is to be expected, the condition that allows CSSR to learn correctly this process is that the maximum length used has to be greater than the threshold ($l_{max} > t$).

For l_{max} smaller than the threshold, the learned automaton is equal than the even process automaton, since the algorithm performs a generalization. This can be understood intuitively because if CSSR has not seen any suffix with an odd number of 1's, because they occur in longer suffixes than the histories taken into account, it will not be able to distinguish this pseudo-even process from the normal even process. Figure 3.18 shows what CSSR would learn with $l_{max} = 3$ for a pseudo-even process with threshold $t = 3$.

Note that this set of causal states is exactly the same obtained for the even process (figure 3.7), so the final automaton obtained after recursion step will be the even process automaton. As it has been said, this behavior is logical, since CSSR needs a longer length

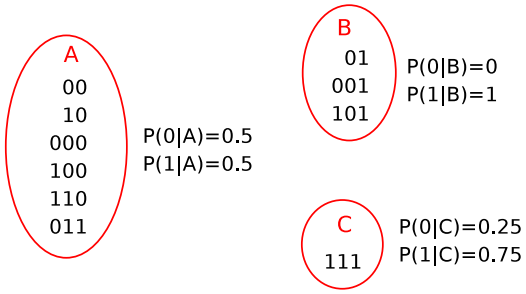

Figure 3.18: Causal state set learned with CSSR for the pseudo-even process with $t = 3$ and $l_{max} = 3$.

to learn correctly the involved patterns, but it can be considered a problem the fact that CSSR generalizes (in this case incorrectly) patterns from data instead of capturing just what can be really seen with the maximum length used.

Adjusting Transient States Deletion

An option to solve this problem would be to consider all the suffixes when studying the transitions to determine the transient states. If CSSR is changed in this way, it would never learn correctly the even process, since it will not be able to generalize the patterns from data. In fact, if all suffixes are considered when looking for transient states with the pseudo-even process or the even process, the learned automaton is always equivalent to a pseudo-even process of threshold equal to the maximum length used.

Note that figure 3.19, that represents what this second implementation learns for the even process with $l_{max} = 6$, has the same states and transitions as the correct automaton for the pseudo-even process with threshold 5 except that the transition probabilities for state G are different. This means that, except for the probabilities, the second implementation will learn the same for the pseudo-even process of threshold $t = 5$ with $l_{max} = 6$ than for the normal even process, since with this approach, no generalization is performed. Furthermore, this is also the automaton that will be learned with this length for any process of threshold bigger than 5.

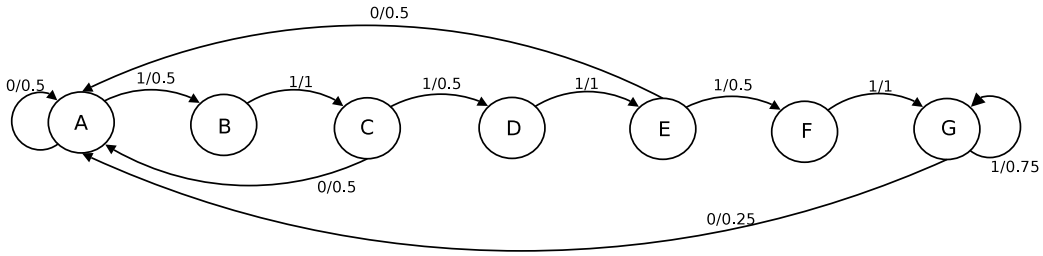

Figure 3.19: Learned automaton for the even process with $l_{max} = 6$ studying all suffixes to determine the recurrent states.

The good point about this second implementation is that it does not over-generalize, but on the other hand it would never learn correctly infinite patterns. This gives experimental

evidence that the fact of using just the short suffixes to determine the recurrent states is the correct option, as it has been argued in section 3.3.1, specially if we want to use all power of causal states. Nevertheless, this example also shows that if l_{max} is too small for the involved patterns, the learned automaton will not be correct, since it will not either be correct using the alternative implementation. This must be taken into account for further experiments in NLP tasks. Let's take for example the chunking task, where CSSR will not be able to capture chunk structure of length l using $l_{max} < l$. Furthermore, given that the limited amount of data will limit the maximum possible value of l_{max} to be used, we have to take into account that longest chunk patterns would be probably lost.

Nevertheless, we can expect that as CSSR is able to perform generalization from data, it will be able to learn patterns that are not specifically present in the data but are also possible in natural language sequences. On the other hand, though, it can also be argued that for NLP patterns it will be better that CSSR does not generalize anything (taking into account all suffixes when determining recurrent states), since we can prefer the system not to generalize from data but to learn just the patterns that has been seen with the l_{max} used. In this thesis, we will study the influence of these two possible approaches when applied to NLP tasks.

3.4 CSSR compared with other techniques

In this section, we discuss the difference between CSSR algorithm (and ϵ-machines in general) and the algorithms presented in chapter 2 that are closer to this technique: Markov Models, VLMM and Suffix Trees. We will also compare CSSR with other algorithms that learn ϵ-machines.

3.4.1 ϵ-machines versus Markov Models

Once causal states have been reduced to sets of suffixes of a maximum determined length, ϵ-machines have the form of Markov Models (MM), where a state is build for each possible n-gram (equivalent to a suffix).

Then, the many desirable features of MMs are secured for ϵ-machine, but the later introduce some interesting features that Markov Models lack. One of the main new issues of ϵ-machines with respect to Markov Models is that for ϵ-machines it is not necessary to make any *a priory* assumptions about the architecture of the system. Each ϵ-machine reconstruction algorithm builds an automaton grouping the histories depending on their future distribution probabilities while MM create a state for each possible suffix. Furthermore, since causal states join various histories in each state, the size of the generated automaton will be really smaller than the equivalent MM. This could be very useful in systems where the amount of data limits the order of the MM that can be learned. We can expect to be able to learn an equivalent ϵ-machine with longer histories with the same memory. Of course, it depends on the studied process how many suffixes will be grouped into a causal states, and thus, how much bigger will be the Markov Model with respect to the ϵ-machine.

Another issue to take into account is that in terms of Markov Models, $l_{max} - 1$ would be the potential maximum order of the model. If the generated ϵ-machine would have a state for each history we would get a Markov Model of order $l_{max} - 1$.

3.4.2 ϵ-machines versus VLMM and Prediction Suffix Trees

Variable Memory Length Markov Models (Rissanen, 1983; Buhlmann & Wyner, 1999; Willems et al., 1995; Tino & Dorner, 2001) and Prediction Suffix Trees (Ron et al., 1994b; Ron et al., 1994a) properties can be studied together since from one given description of data with one of this two techniques, it is always possible to create an equivalent description with the other. In this section, we will compare both to ϵ-machines.

With all ϵ-machines, PST and VLMM, states may be regarded as classes of histories of the observable process. With a PST or a VLMM, every state is defined by a single suffix, and consists of all and only the histories ending in that suffix. The crucial distinction is that the states in an ϵ-machine are not defined by unique suffixes, but may contain multiple suffixes representing different contexts that can lead to that state. Furthermore, there are processes with a finite causal-state representation which do not have finite suffix-tree representations. An example of such a process is the "even process" (see section 3.3.1) that needs an infinite PST to be represented correctly.

The strength of ϵ-machines is that by joining some suffixes in the same state, they perform a generalization capable of building automata for reproducing infinite process. This can not be done using VLMM or PST. On the other hand, every process with a finite suffix tree representation has a finite causal state model, which is discoverable by CSSR or some other ϵ-machine reconstruction algorithm. Thus, CSSR can be considered more powerful than learning PST and for that reason we are interested in studying its performance in NLP tasks.

On the other hand, VLMM and PST have the advantage over CSSR that while CSSR studies all histories up to a fixed maximum length, VLMM and PST consider just long suffixes when necessary, leaving some suffixes unexplored when they do not seem to introduce different statistical information. In this way, it can be claimed that VLMM and PST can deal with longer maximum lengths, since they do not need to study all suffixes, but as CSSR groups histories in states, the computational need is always reduced with respect to the needs of an HMM for example, so CSSR can still compete with VLMM and PST in this aspect.

ϵ-machines, VLMM and PSTs have the advantage over HMMs that the architecture of the system is learned according to the data rather than being predefined as in HMMs. Thus, the number of generated states with these methods is expected to be less than $|\Sigma|^L$ (where $|\Sigma|$ is the size of the alphabet, this is, the number of considered symbols and L is the maximum length of the considered suffixes), which would be the number of states in a HMM.

3.4.3 CSSR versus other ϵ-machine learning algorithms.

The main difference between CSSR and State-Merging ϵ-Machine Inference algorithms (Crutchfield & Young, 1989; Crutchfield & Young, 1990; Hanson, 1993; Crutchfield, 1994) is that while CSSR starts considering that all the suffixes belong to the same state and constructs the automaton splitting the states when necessary, State-Merging ϵ-Machine Inference algorithms start assuming that each suffix belongs to a state and is whittled down by merging.

As algorithms that infere ϵ-machines, CSSR and State-Merging ϵ-Machine Inference algorithms both have the same domain of applicability, but CSSR is more well-behaved and converges faster than merging methods.

Furthermore, CSSR makes use of some properties about causal states (they are deterministic, Markovian, etc) to guide the search, the automaton construction. This information is not used by other reconstruction algorithms.

3.5 CSSR applications

CSSR has been applied to various research areas. Here we present a review of these applications and the obtained results.

Varn and Crutchfield (2004) analyze solid-state phase transformations. They study the transition of annealed ZnS crystals between two different solid structures: 2H and 3C. ZnS crystals are stable in one of these two phases depending on the temperature. These two phases can be characterized by the inter-ML spin distribution. This is, the patterns of spin values of consecutive atoms.

In that work, they study the spatial configuration of the crystals in terms of the spin position during the transformation, given a parameter f that models how close is the crystal to one of the two phases. When f=0 the system is in 2H phase, and for f=1 in 3C. The spin position depending on f and on the position in the crystal form the data that CSSR uses to construct an ϵ-machine. This machine models the structural description of the spin configuration in various steps of the transformation (i.e. various f values). Thus, the learned automaton reproduces the observed spin sequences in the different stages of the transformation. The alphabet used in the transitions has only the two possible spin values values, 0 and 1. The parameter f modifies the transition probabilities modeling in this way the two possible phases and the transition stages. Then, the built automaton changes the allowed transitions depending on f modelling the two different phases and the intermediate stages.

In this framework, CSSR is used to obtain a readable automaton that models the patterns of the two phase crystals and also modelling the transition between them. With this small alphabet and a relatively simple structure two be detected the algorithm is useful and builds accurate automata.

Another CSSR application is presented by Ray (2004). The goal of this work is to perform anomaly detection in complex dynamical systems using tools as Symbolic Dynamics, Finite State Automata and Pattern Recognition.

In a dynamical system, an anomaly is a deviation from its nominal behavior. The idea of the presented work is to use pattern acquisition techniques to build a patterns of the normal behavior of a system and then use this pattern to detect deviations in new data. To do so it is necessary to define a measure of the deviation from the pattern to determine if an observation can be considered an anomaly or not. They also try to deduce if the anomaly has any pattern.

In fact, Ray (2004) uses CSSR theoretical work to develop a pattern representation, named *D-markov chains* that is simpler than ϵ-machines and capture enough information for anomaly detection.

One application in the field of Sociology and also related to Natural Language Processing is presented in (Cointet et al., 2007). The aim of this work is to investigate how various information sources such as weblogs and on-line media are intertemporaly correlated.

To do so, they study the presence of 75 relevant topics in 33 French political corpus and 6 on-line press sources during 30 days. Three categories are built with the blogs depending

on its similarity and a fourth category is used for press. Then, they want to model the interaction between these 4 categories using CSSR algorithm.

The alphabet is created in the following way: for each term, consider the frequency of appearing in each category. If this frequency (properly normalized) is bigger than a determined threshold, the term is considered to appear in this category. Then, each symbol of the alphabet is a possible appearing combination of a term or not appearing in each category. So the alphabet has $2^4 = 16$ symbols.

Thus the presence of each term in the various categories for the different days can be encoded as a sequence of alphabet symbols, one symbol for each day. Using the different terms they expect to find more correlations, since it increases the number of instances.

Using the data encoded in this way, they use CSSR to learn an automaton representing the causal structure of the data. They interpret the possible transitions as existing correlations between groups of blogs having also the probabilities of this correlations.

All the systems presented so far use CSSR as a way to capture the patterns of some data in order to try to understand the behavior of a determined process. Another use of CSSR found in the literature is to use the algorithm to study the complexity of a system or of a data set.

In this line, Shalizi (2001) and Shalizi et al. (2004) propose the use of computational mechanics and CSSR to compute the statistical complexity of a process as defined by Crutchfield and Young (1989). Via the use of the algorithm, not only the statistical complexity of the process can be measured, but also a proposal to define "emergence" and "self-organization" in complex systems is presented. The ideas can be applied to both temporal and spatial patterns.

Following the same line, Boschetti (2007) defines the complexity of an ecological model as the statistical complexity of the output it produces. It is important to define this complexity measure because the ecological models, to become more accurate, are growing in complexity (taking into account more parameters, processes, etc.). This makes the models less tractable, so it is necessary to reach a trade off between the accuracy of the model and its complexity. To do so, it is necessary not only to define but also to measure the complexity of ecological models. The proposal of Boschetti (2007) is to use CSSR to measure this complexity, similarly to the work presented in (Shalizi, 2001; Shalizi et al., 2004).

Also, Park et al. (2006) present an approach to use CSSR to compute the complexity and entropy density of the Korean stock market. The idea is to use CSSR to learn an automaton that models the variation on the Korean Composition Stock Price Index (KOSPI) along time. The time series formed by the variation of this index is converted into a binary time series depending on whether the variation is positive or negative. With these data, an ϵ-machine is computed using CSSR for each year and the complexity of the time series is computed and compared along the different years. Furthermore, the machine is also used to compute the entropy density, that shows how random the process is.

In this thesis, first of all we will study the CSSR ability to learn patterns related to NLP data, since it has been seen that the algorithm is useful to capture patterns in other research fields. Nevertheless, the main goal of this thesis is to apply CSSR to NLP annotation tasks. To do so, it is necessary to introduce some hidden information into the system in order that it can learn an automaton that can be afterwards used to annotate new sentences. As far as we know, there is no previous work in NLP or other fields in the line of using CSSR for annotation tasks.

CHAPTER 4

Study of CSSR Ability to Capture Language Subsequence Patterns

Before applying CSSR algorithm to annotation NLP tasks, some experiments to study the ability of CSSR to capture the patterns that form some language subsequences were performed. The goal of these experiments is to prove that this method is able to capture sentence patterns, as well as to study the influence of the various parameters.

This chapter presents these experiments, that conform a study of the applicability of CSSR to some NLP basic tasks. For this first study, no annotation tasks are performed since we are just interested on studying the patterns that CSSR can learn in the NLP field to see if it can be useful for the annotation tasks.

The experiments are focused on studying the patterns of Named Entities (NEs) and Noun Phrase (NP) chunks that CSSR learns. It will be seen that the algorithm is able to capture the patterns of these language subsequences, and the conditions under which the algorithm performs better and the limitations of CSSR will be discussed.

For NE patterns, we will show that CSSR is able to learn intuitive automata that capture the patterns of sentences in terms of the selected alphabet. For NP patterns, we will see that the learned automata are not so easily interpretable, since they are bigger, so we will propose another way to qualitatively evaluate the correctness of the CSSR automata. Furthermore, and to study the influence of the available amount of data, we will present some experiments with artificially generated data and we will see how the learned automata reproduce better the NP patterns when more data is available.

The study of the patterns learned for NEs, presented in section 4.2, was published in (Padró & Padró, 2005c) and the study for NPs (section 4.3) can be found in (Padró & Padró, 2007b).

4.1 Using CSSR to Capture the Patterns of Language Subsequences

The patterns that may be found in a sentence, 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 kind of pattern we want to capture, 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. For the two tasks studied in this chapter, different sets of features may be relevant.

One characteristic issue of the CSSR algorithm is that it is conceived to model stationary processes. That is, the algorithm learns automata that do not have initial and final states but that are considered to generate infinite sequences, thus modelling a continuous process. But the subsequences in language are not in this category so we have to perform some adaptation to use this algorithm for learning the patterns of these subsequences. To do so, what we did was to regard a text sequence as a stationary process in which determined subsequences occur once a while. Doing so implies the automaton is modelling the pattern of the whole sequence (the text), not the pattern of an isolated subsequence, but these patterns will be included in the automaton that reproduces the patterns of the whole text.

Next sections present the experiments performed using CSSR to learn patterns of NEs and NP chunks. The goal of these experiments is to see if this method is able to capture sentence patterns, as well as studying the influence on the learned automata when using various l_{max} and amount of data. As explained in section 3.2.3, when using CSSR it is necessary to reach a trade off between the amount of data (N), the vocabulary size (k) and the maximum length used (l_{max}). The maximum length that can be used with statistical reliability is given by the ratio $\log N / \log k$. We will use this ratio as an upper limit of the l_{max} used in our experiments.

4.2 Capturing the Patterns of Named Entities

To learn an automaton that may represent the NEs in a sentence, CoNLL-2002 shared task (Tjong Kim Sang, 2002a) training corpus for Spanish was used. These data have to be encoded as a token sequence in a given alphabet. This alphabet should contain the most relevant features regarding NER task. In our experiments, the alphabet Σ consists of the symbols $\Sigma = \{G, S, M, a, w\}$. From the features each token presents, a symbol of Σ is assigned to it according to the following mapping:

- **G:** Beginning of the sentence, capitalized, not containing numbers, not in the dictionary[1].
- **S:** Beginning of the sentence, capitalized, not containing numbers, one of its possible analysis being a noun.
- **M:** Not at the beginning of the sentence, capitalized.

[1]The dictionary we use is the one provided by FreeLing (Carreras et al., 2004; Atserias et al., 2006)

- **a:** Not at the beginning of the sentence, non-capitalized, functional word[2].
- **w:** Other.

Note that each word can have only one symbol, since each one excludes the others. These features encoded in this alphabet are the same that will be used later in NER experiments (Section 6.3).

Once the alphabet was set, the training corpus used in CoNLL-2002 was translated into this alphabet and then, various automata were learned using CSSR with different l_{max} values. The automata learned via CSSR are expected to capture the patterns of sentences in terms of the features presented above.

The training corpus we use has about $N = 260,000$ words, and the the alphabet has $k = 5$ symbols so $\log N / \log k = 7.8$. Then, the maximum length that can be reliably used in this case is between 7 and 8 (see section 3.2.3). Figure 4.1 shows the length distribution of the NEs as found in the CoNLL-2002 training corpus. In this figure, it can be seen that most NEs have 1 to 3 words, so using small l_{max} values should be enough to capture most of the patterns of NEs.

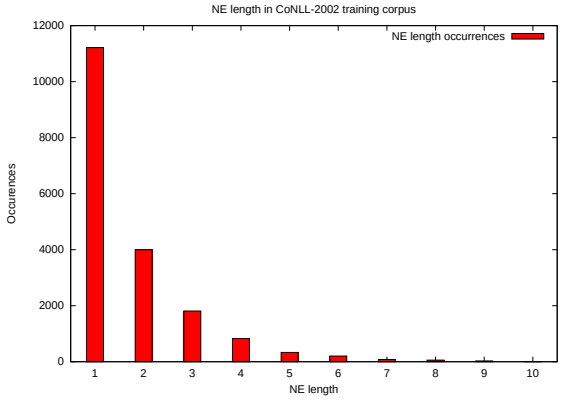

Figure 4.1: Number of NE occurrences in CoNLL-2002 corpus for each NE length.

Since CSSR algorithm builds probabilistic automata, some states that would be melted by a minimizing algorithm if the probabilities were ignored are maintained separated because they have different probability distributions. This makes the built automata a little bit difficult to be read. The CSSR automaton for $l_{max} = 2$, with the suffixes belonging to each state and the transition probabilities, is shown in figure 4.2.

In order to study if the patterns recognized with CSSR automata are correct, the transition probabilities can be ignored and the automata minimized. Doing so implies that we are ignoring that CSSR has produced an stochastic automaton and only check if the non stochastic language it recognizes reproduces the real patterns. If we do so with the CSSR automata of length 2 and 3, the automata shown in figure 4.3 are obtained. It can be seen that they are logical automata that reproduce the intuitive pattern of sentences which may contain NEs, in terms of the selected features. If longer maximum lengths are used, more states are created and when minimized, similar automata to those presented in this figure are obtained.

[2]Functional words are articles or prepositions that are often found inside an NE

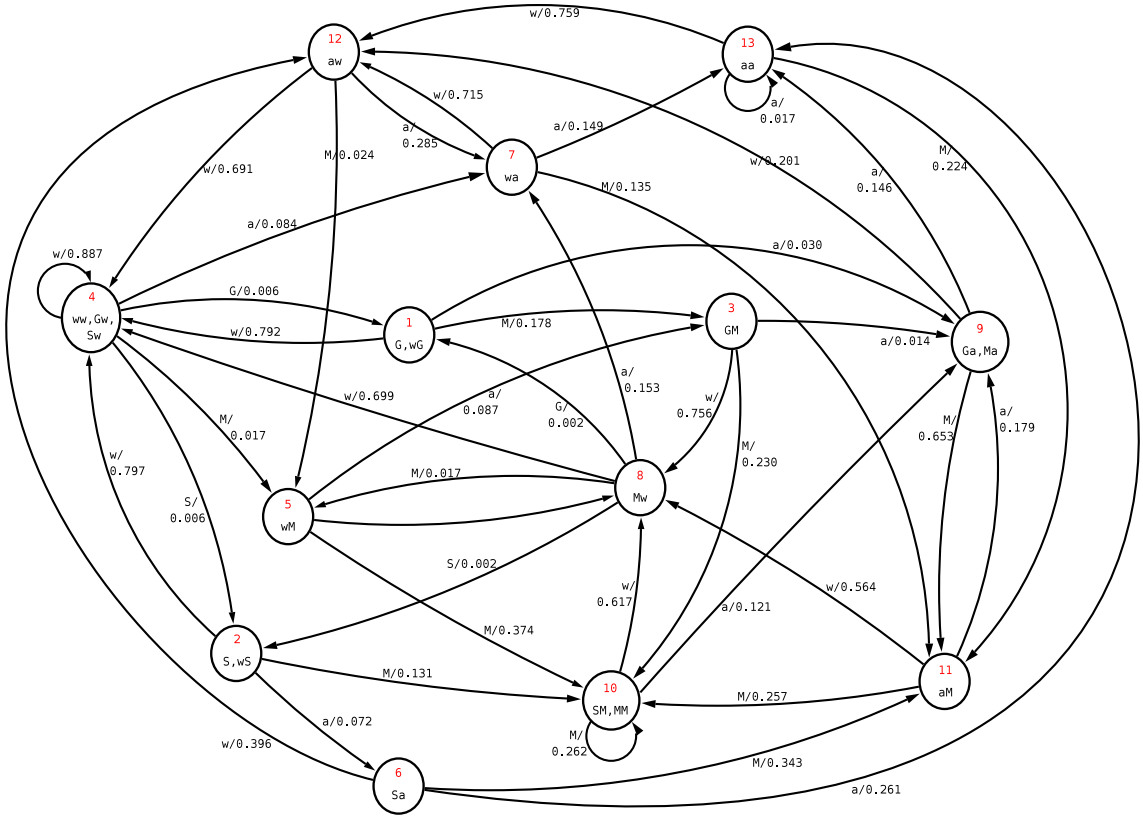

Figure 4.2: Learned automata with $l_{max} = 2$

It can be observed that the difference between the automata obtained with different maximum lengths is that, as the length grows, the automata become more informed and take into account some unseen events. For example, for $l_{max} = 3$, there is an extra state that forbids the existence of the combination "awG" or "awS" . In fact, this combination is never seen in the data, because G and S are symbols that only appear at the beginning of the sentence, so the w symbol preceding a G/S will be the punctuation mark that ends the previous sentence and it could not be preceded by a preposition or an article (a) because a sentence in Spanish never finishes with such a word.

Note that in these minimized automata there is no difference between the symbols G and S. In fact, the only difference is observed with the non-minimized automata, where G and S present different future probabilities so are seen in different transitions. Nevertheless, this distinction between G and S is maintained because it may be useful when performing NER.

Regarding automata with longer histories, the number of states of the CSSR automata and the minimized automata are shown in table 4.1. Note that the number of states grows with l_{max}. This is because the patterns involved with such lengths become more complex and CSSR is not able to build a compact automaton representing them. Furthermore, there are some noisy patterns in the corpus (often those of very long NEs) that are not captured by short maximum lengths but that lead CSSR to confusion if longer lengths are used.

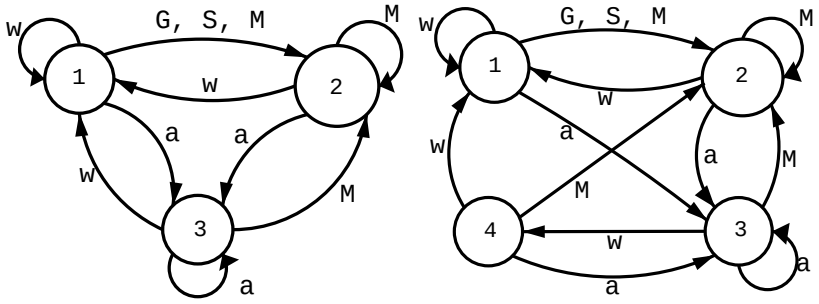

Figure 4.3: Learned automaton with $l_{max} = 2$ (right) and $l_{max} = 3$ (left)

	Number of States	
l_{max}	CSSR Automaton	Minimized Automaton
2	13	3
3	39	4
4	77	15
5	187	53
6	322	151
7	775	533
8	1777	1395

Table 4.1: Number of states of the CSSR automata and the corresponding minimized automata for various maximum lengths.

4.3 Capturing the Patterns of Noun Phrases

In this section, the experiments performed to study CSSR behavior when trying to learn Noun Phrase structure for English are presented. In this case, the features used are the Part of Speech (PoS) tags of words as syntactic structure of sentences depends strongly on these tags.

The data used for NP detection are extracted from the English Penn Treebank II (PTB2) corpus (Charniak, 2000). This is a corpus with full parsing information, with eleven different chunk types and a complete analysis of sentences. Nevertheless, in this work we use this corpus translated into the same format used in CoNLL-2000 shared task data (Tjong Kim Sang & Buchholz, 2000). We focus on the use of this format because this is the one that will be used for Chunking task (section 6.5). The translation from PTB2 to this format is done, as in CoNLL-2000 shared task, using *chunklink.pl*[3] script that converts parsed sentences (e.g. PTB2 files) into a common format containing the same information as the original files. Using this conversion, we selected only the chunks that are not recursive, and formed only for terminal symbols, as in CoNLL-2000 shared task data. Though in these data all chunks are tagged, in the current study just NP chunks information will be used.

In this case, the features used to build the alphabet are the Part of Speech (PoS) tags of each word, since syntactic structure of sentences depends strongly on these tags. To learn an automaton representing NP patterns it is necessary to distinguish the words forming an

[3]Written by Sabine Buchholz: http://ilk.uvt.nl/~sabine/chunklink/README.html

NP from the words outside any NP, even if the PoS tag is the same. To do so, each word belonging to an NP is represented by its PoS tag (that is a symbol of the alphabet) and the words not belonging to NP chunks are mapped into a special symbol "*Out*".

The total number of different PoS tags is 44, but there are some PoS tags that never appear inside any NP, so they will not be part of the alphabet, since the symbol for the words outside NPs is always the same independently of their PoS tag. Taking this into account, the alphabet has 38 symbols, 37 for the different PoS tags observed inside NPs in the corpus and 1 for the special symbol "*Out*". Figure 4.4 shows how a sentence with information about all chunk types will be translated into a sequence that highlights NP structure.

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
W.R.	NNP		NNP
Grace	NNP		NNP
vice	NN		NN
chairman	NN		NN
,	,	none	Out
who	WP	NP	WP
resigned	VBD	VP	Out
.	.	none	Out

Figure 4.4: 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 is expected to learn an automaton representing NP chunk structure in terms of PoS tags.

The training corpus has the order of $N = 1.000.000$ words what means that $l_{max} < \log N / \log k = 3.8$, so the maximum length that can be theoretically used is less than 4. We will perform experiments with l_{max} from 1 to 4 to study the system behavior. Figure 4.5 shows the number of NPs of various lengths observed in the corpus. It can be seen that most of the Noun Phrases have a length between 1 and 4, so an automaton built with length 4 should be able to reproduce the most frequent patterns.

Various automata with l_{max} from 1 to 4 were learned, but since the alphabet used in this case is bigger than the one used in NER, the obtained automata are much bigger, even when minimized. The number of states of the minimized automata varies from 34 for $l_{max} = 1$ to 1767 for $l_{max} = 4$. Though these can be useful automata, it is not possible to perform a qualitative evaluation of the results as in section 4.2 since the generated automata are not intuitively interpretable. Thus, it is not possible to conclude whether CSSR is able to learn a good automaton for NP detection just looking at the automata as we did for NE patterns, so another method to evaluate the correctness of the generated automata was devised.

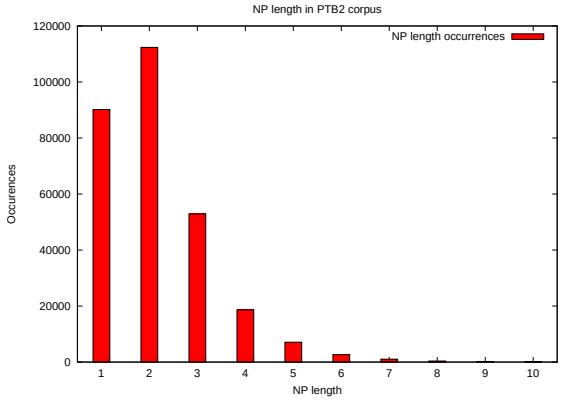

Figure 4.5: Number of NP occurrences in PTB2 corpus for each NP length.

4.3.1 Comparing Grammars to Evaluate CSSR Learning Ability

In order to obtain a qualitative evaluation of the automata acquired by CSSR, the following method was devised: with PTB2 corpus, modified to follow the same format than CoNLL-2000 data, a grammar for NPs is extracted, counting the frequency of appearance of each NP pattern. Using this corpus, the learned grammar is regular since the NP chunks in the modified corpus are never recursive and are formed only by terminal symbols. So, the grammar is a set of all the possible PoS sequences forming NPs observed in the corpus.

On the other hand, the automaton learned using CSSR can be used to generate the same kind of patterns. Thus, using the transitions and probabilities of the automaton, a variety of sequences of PoS tags are generated. The subsequences between two "*Out*" symbols are the NP patterns that CSSR has learned. These patterns, and their frequencies, are extracted and compared with the grammar extracted directly from PTB2. The more similar the set of rules produced by CSSR is to the PTB2 grammar, the more accurate the automaton representing the data can be considered.

There are two main differences between the rules generated by the CSSR automaton and the rules extracted from the corpus. On the one hand, there are rules that the CSSR automaton generates that are not present in the corpus, so CSSR introduces new patterns. This is due to the fact that CSSR over-generalizes patterns from data. As maximum length grows, the number of over-generated patterns falls, what means that CSSR generalizes better, as it may be expected. On the other hand, there are some differences in the frequencies of the common rules. This difference in part is due to the probability mass given to new rules, and also is reduced when maximum length grows. Table 4.2 shows the ten most frequent rules of the PTB2 grammar with their frequency compared with those of the CSSR-produced rules for various maximum lengths.

To perform the comparison between the patterns extracted from PTB2 and the patterns generated by each CSSR automata, Jensen-Shannon divergence (Lin, 1991) is used. This divergence gives a measure of the distance between two distributions. It is derived from Kullback-Leibler divergence introducing the advantage that it is symmetric. If Kullback-Leibler divergence is used, the distance between a distribution P and a distribution Q is different than that between Q and P:

Rule	Rule Frequency				
	PTB2	CSSR			
		$l_{max} = 1$	$l_{max} = 2$	$l_{max} = 3$	$l_{max} = 4$
DT NN	0.1315	0.1256	0.1267	0.1290	0.1224
PRP	0.0713	0.0689	0.0679	0.0686	0.0652
NNP	0.0639	0.0688	0.0611	0.0623	0.0505
NN	0.0576	0.0657	0.0562	0.0584	0.0547
NNS	0.0562	0.0553	0.0544	0.0557	0.0587
NNP NNP	0.0534	0.0255	0.0460	0.0493	0.0515
DT JJ NN	0.0415	0.0267	0.0371	0.0416	0.0412
JJ NNS	0.0319	0.0204	0.0325	0.0334	0.0336
DT NNS	0.0221	0.0218	0.0229	0.0211	0.0257
JJ NN	0.0206	0.0318	0.0223	0.0181	0.0205

Table 4.2: Frequencies of the ten most frequent rules in the PTB2 extracted grammar compared with the CSSR-generated grammars

$$D_{KL}(P,Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \neq D_{KL}(Q,P)$$

Jensen-Shannon divergence solves this problem by building the combined distribution R, and computing the distance between P and Q as a combination of the Kullback-Leibler divergence between P and R and the divergence between Q and R:

$$R = \frac{P+Q}{2}$$

$$D_{JS}(P,Q) = \frac{1}{2}D_{KL}(P,R) + \frac{1}{2}D_{KL}(Q,R)$$

Our goal is to compare the frequencies of the the PTB2 grammar with those of the CSSR generated grammar. To do so, we use Jensen-Shannon divergence as a measure of the differences between these two sets of weighed rules. If a rule is present in one grammar but not in the other, it is also computed as a difference between frequencies. The smaller the divergence is, the better CSSR can be considered to reproduce data patterns, since the more similar are the frequencies of the two rule sets. Figure 4.6 shows the values of this divergence for various maximum lengths.

In this figure, it can be seen how Jensen Shannon divergence falls as l_{max} grows, till $l_{max} = 3$. For $l_{max} = 4$ the divergence rises again because there is not enough data to learn an automaton with statistical reliability, so using CSSR with this length introduces incorrect patterns.

In order to know if these obtained divergences are big or small, we measured the divergence between two parts of the PTB2 corpus. To do so, we splitted the corpus in two parts of similar sizes, we extracted a set of rules from each subcorpus and then we compared the two sets of rules using Jensen-Shannon divergence with the same procedure used to compute the distance between PTB2 grammar and the set of CSSR generated rules. To minimize the effect of splitting the corpus in two parts just once, we repeated the process several times

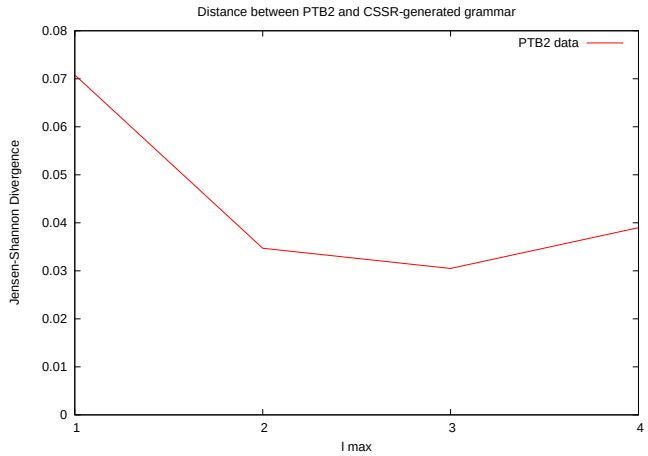

Figure 4.6: Jensen Shannon divergence between CSSR generated set of rules and grammar extracted from PTB2, for various values of l_{max}.

using various partitions of the corpus. Thus we obtained 20 measures of Jensen-Shannon divergence between two parts of the PTB2 corpus, and the average distance was 0.06 ± 0.01.

This means that CSSR can be considered to learn the patterns in PTB2 corpus correctly, since the obtained divergence between PTB2 grammar and CSSR generated rules is smaller (for l_{max} greater than 1) than the average distance between two parts of the same corpus.

4.3.2 Generating Artificial Data to Study CSSR Performance

One of the limitations of the study presented in last section is that, given the size of the alphabet, there are too few available data to learn automata with large l_{max}. In fact, it has been seen that the larger l_{max} that can be used with PTB2 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 artificial data was created in the following way: using the PTB2 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 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 behavior 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 β 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 β when the data is increased by this value. So two histories maintained in the same state with the original corpus can be considered significantly different when the corpus is repeated β times. 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 behavior.

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 4.7 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. We also reproduce the results obtained with the real corpus (already shown in figure 4.6) in order to compare the results obtained with the real corpus with those obtained with the generated corpus. 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, since 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 behavior 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 does not 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.

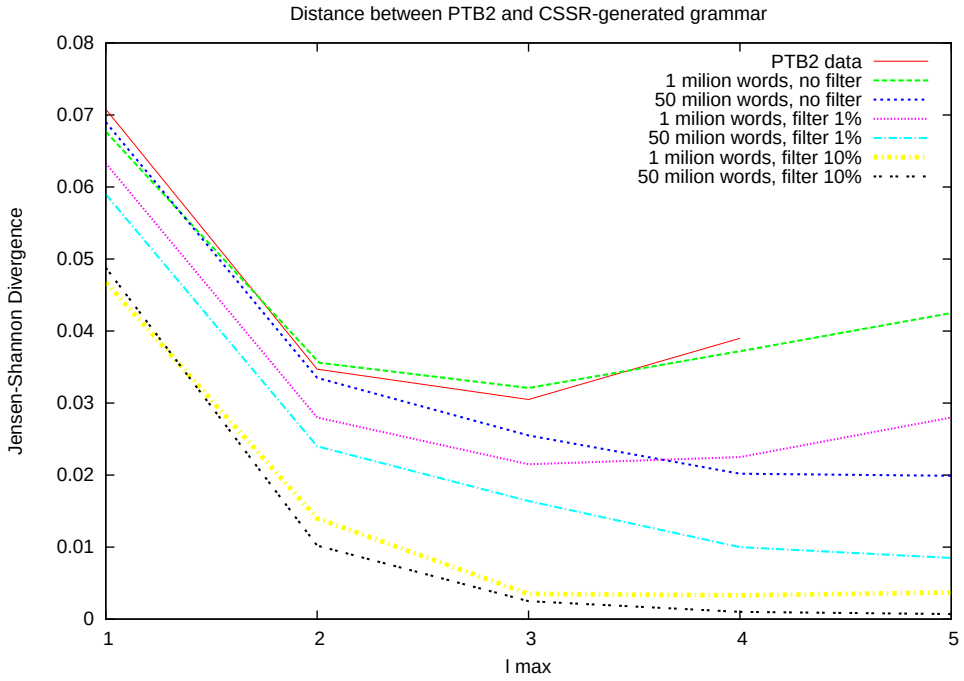

Figure 4.7: Jensen Shannon divergence between CSSR generated set of rules with a generated corpus and grammar extracted from PTB2, for various values of l_{max} when using various filter levels of the grammar

4.4 Discussion

Considering 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 depending on the amount of available data. For each process, there is a necessary l_{max} value to be able to capture the patterns, and if this value is high, large corpus will be necessary. Furthermore, since the minimum amount of data necessary to learn a good 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 NE patterns, it has been seen that the automaton learned using CSSR is intuitive and captures the patterns of sentences in terms of the designed alphabet. This is partly due to the fact that for this task the alphabet is quite small, so it is easier to build an understandable automaton than in NP task.

For NP detection, the linguistic meaning of CSSR generated automaton can not be intuitively understood, but that does not mean that it does not 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, since 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 since 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.

4.5 Conclusions of this Chapter

This chapter has presented a study of how CSSR is able to capture some patterns in language. It has been seen that this algorithm can learn an automaton representing a process if there is enough data available without, or if the process is simple enough. The influence of the amount of data and the noise level in the generated automaton has been qualitatively studied.

The main conclusion of this chapter 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} value. 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. CSSR has been proved also to be useful when dealing with systems with smaller alphabets –as in other applications of CSSR such as those presented in (Varn & Crutchfield, 2004; Clarke et al., 2003; Cointet et al., 2007)– but to use it in systems with lots of features to be taken into account, as NLP tasks, this could be limited by the amount of available data.

Nevertheless, since the algorithm has proven to be useful to learn NLP patterns, we think it is interesting to apply it to annotating tasks in order to see if the characteristics that make CSSR theoretically more powerful than other techniques such as HMM or VLMM, are reflected in the obtained results.

In fact, 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. Chapter 5 presents our proposal to introduce this hidden information into the system, and folowing chapters will present the results obtained with this technique.

CHAPTER 5

Adapting CSSR to Perform Annotation Tasks

So far, this work has focused on the performance of CSSR when learning sentence patterns in terms of some selected sets of features encoded into the alphabet. It has been seen that CSSR is able to capture the patterns underlying some language subsequences. Nevertheless, for NLP tasks, it is necessary to develop systems able to annotate new sentences. This is, given a new sentence, the system has to detect the subsequences we are interested in. In the case of Noun Phrase (NP) detection, for example, we would have sentences with their PoS, without the information about which words form an NP chunk, and the system should detect the NPs of this sentence. To perform this annotation task with CSSR, it is necessary to encode into the system the hidden information about where certain subsequences begin and end. Originally, CSSR is not conceived to encode this hidden information, so in this chapter we propose an approach to adapt CSSR to include and use it to annotate new sentences.

In this approach, we focus on tasks where some subsequences of words, with specific properties, have to be detected. Some examples of this kind of task are Named Entity Recognition (NER) and Chunking. For the NER task, the subsequences we are interested in are Named Entities (NE) and for Chunking, they are the different kind of phrases. As a general way of referring to these subsequences we are interested in detecting, we will call them "chunks", understanding this concept as the wide idea of a subsequence of words belonging to a class. Also, the task of determining subsequences of words will be referred to as "chunk detection".

The general idea of our approach is to use CSSR to learn an automaton representing the chunk structure taking into account the hidden information that says where each subsequence begins or ends. Once the automaton is learned with this information, it can be applied to detect subsequences in untagged text.

This approach and its application to various NLP tasks has been published in (Padró & Padró, 2005a; Padró & Padró, 2005b; Padró & Padró, 2005c).

5.1 Representing the Hidden Information

First of all, we need to state a way of representing the hidden information we want to introduce into the system. Since we are interested in subsequence annotation tasks, this hidden information is simply if a word belongs or not to a determined chunk or subsequence.

There are many different possible approaches to tagging this kind of structures (Tjong Kim Sang & Veenstra, 1999) one of the most widely used is the "B-I-O" approach (Ramshaw & Marcus, 1995): each word has a B, I or O tag, being B the tag for a word where a subsequence or chunk begins, I when the word is part of a chunk but not the beginning, and O the tag for the words not belonging to any subsequence.

Another approach often used to annotate subsequences, specially syntactic chunks, consists of using brackets to determine their boundaries (Church, 1988): the first word in a chunk has an opening bracket "[" and the last one a closing bracket "]". All the words inside two consecutive opening and closing brackets belong to the same chunk.

There are other ways of representing chunks but they are mainly modifications or combinations of these two main methods. For a specific review on them, see (Tjong Kim Sang & Veenstra, 1999).

In this work, we will use the "B-I-O" approach, since it is one of the most widely used in all tasks regarding subsequence detection, and also because our experiments will use the data of CoNLL 2000 and 2002 shared tasks[1] that follow this "B-I-O" approach. Nevertheless, the proposed approach could be used with the other possible annotation systems, since the idea is general enough.

5.2 Learning the Automaton with the Hidden Information

To learn the automaton that must reproduce a chunk structure, different information about the words is used, as it has been seen in chapter 4 where various alphabets have been created for learning NE and NP patterns. Obviously, the used information depends on the specific kind of subsequence we want to detect, and can be for example orthographic, morpho-syntactic, about the position in the sentence, etc. Using the chosen features, the alphabet of the automaton is built as a closed set of symbols, and the words in a sentence are translated into this alphabet. The sentences in the train corpus translated in such a way will be the sequence that we use to learn the automaton via CSSR. The learned automaton will reproduce the sequence behavior in terms of the chosen features.

This is conceptually equivalent to the process performed in chapter 4, but now we need to introduce the hidden information about if a word is part of a determined chunk or not. To do so, we propose to introduce into the alphabet the information of the chunk tag (B, I or O) available in the supervised training corpus. Thus, the correct tag is part of the alphabet symbol that represents each word and is taken into account when building the automaton.

More specifically, we consider two different parts of the alphabet. First of all, there is the part that can be built without any hidden information, that we will call the *visible alphabet*: $\Sigma = \{s^1, s^2, ..., s^k\}$. This alphabet corresponds to the symbols associated to the visible features of the words and does not take into account any hidden information. That

[1] CoNLL 2000 shared task was devoted to text chunking, and CoNLL 2002 shared task approached Named Entity Recognition and Classification

means that we can translate a text into this alphabet even if the chunks in that text are not tagged. The alphabets used in chapter 4 are purely visible alphabets.

Once this alphabet is defined, to introduce the hidden information, we create another alphabet, to which we will refer to as *complet alphabet*, that combines each possible symbol in the visible alphabet with each possible hidden tag: $\Sigma_{BIO} = \{s_B^1, s_I^1, s_O^1, ..., s_B^k, s_I^k, s_O^k\}$. Thus, the alphabet size has been multiplied by the number of hidden tags, in our approach, the complet alphabet is three times bigger than the visible alphabet.

Once this complet alphabet is defined, all the sentences in the training corpus, that contain the correct B-I-O tag for each word, can be translated into this alphabet and CSSR is used to learn an automaton with these data. Then, the learned automaton will encode in its transitions not only the information about the relevant features for the task, but also the information about the probabilities of each kind of word having the B, I or O tag in each determined context. Thus, we can later use this automaton to compute the best path for a sequence and use it to annotate chunks in a new text.

5.2.1 An Example

To better illustrate our proposal to annotate chunks in a sentence using CSSR algorithm, here we present an example of annotating Named Entities. Let's suppose an approach where the features used are the same presented in 4.2. In this case, the alphabet will consist of fifteen symbols, which are the possible combinations of those five visible symbols with the three possible NE tags (G_B, G_I, G_O, S_B, S_I, S_O, M_B, M_I, G_B, G_I, G_O, S_B, S_I, S_O, M_B, M_I, M_O, a_B, a_I, a_O, w_B, w_I, w_O). Each word will be translated into one of these symbols depending on the features it presents.

Once the data are properly translated into the alphabet, the automaton is built using CSSR. Figure 5.2 shows a possible (not real) automaton with this alphabet. In this figure, the probabilities are not shown to facilitate the reading but the automaton learned by CSSR would have a probability assigned to each possible transition with each symbol in the alphabet. Note that there are nonexistent transitions (G_I and S_I never occur, for example). This transitions will have a null probability and will go to a *sink* state.

This is a simple example, where only sequences such as $...w_O w_O M_B a_I M_B w_O...$ or $...w_O G_B M_B w_O...$ are considered. This automaton forbids sequences such as $...w_O M_B a_I M_O...$ that are very infrequent in real language but may still be found in a real training corpus. In a real automaton learned with CSSR much more transitions and states will be present, since the automaton reproduces all the patterns seen in the corpus even the very infrequent ones. Furthermore, as the algorithm builds the causal states depending on the probability distribution for the future, the built automata tend to be bigger because states with similar behavior regarding the NE structure are maintained separately because they have different probability distributions.

5.3 Using the Learned Automaton to Annotate Language Subsequences

When a concrete kind of subsequence in a sentence has to be annotated, the information about the correct chunk tag is not available, so there are several possible alphabet symbols for that word. It is possible to know only the part of the translation that depends on the word or sentence features, so we can know the corresponding visible alphabet symbol (s^i),

Word	**Correct Tag**	**Alphabet Symbol**
Fuentes	O	S_O
del	O	a_O
Vaticano	B	M_B
comentaron	O	w_O
la	O	a_O
salud	O	w_O
del	O	a_O
Papa	B	M_B
de	I	a_I
Roma	I	M_I
.	O	w_O
Navarro-Valls	B	G_B
afirma	O	w_O
que	O	w_O
está	O	w_O
estable	O	w_O
.	O	w_O
El	O	w_O
portavoz	O	w_O
...		

Translation: *"Sources of the Vatican commented the health of the Pope of Rome. Navarro-Valls affirms that he is stable. The spokesman..."*

Figure 5.1: Example of a training sentence and its translation to the chosen alphabet.

but we do not know the part of the symbol that depends on the chunk tag, which is, in fact, what we want to know. Given a word with a visible symbol s^i, there are three possible complete symbols for this word: s^i_B, s^i_I and s^i_O. If we determine which of these three possible symbols suits better the word in its context, we will know which is the best B-I-O tag for that word. So we look for the most likely symbol of the complete alphabet for each word, taking into account the visible part of the symbol that we already know.

To find this most likely symbol for each word in a sentence, a Viterbi algorithm is

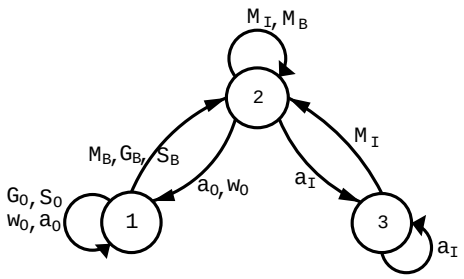

Figure 5.2: Example of an automaton that models text sequences containing simple NEs.

applied. That is, for each word in a sentence, the possible states the automaton could reach if the current word had the tag B, I, or O (that is, if the symbol was s_B^i, s_I^i or s_O^i), and the probabilities of these paths are computed. Then, only the highest probability for each tag is recorded. That means that for each word, the best path for this word having each tag is stored. 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 complete symbols, so of B-I-O tags for each word in the sentence is obtained. There are some forbidden paths, which are those that lead to the OI tag combination. The paths including this combination are pruned out since they include incorrect tag combinations.

Returning to the example presented in 5.2.1, let's see how the automaton presented in figure 5.2 will perform the NE-annotation of a sentence. Most of the time the automaton will stay in the state 1, with input symbols w, as most words will have this symbol. In this state, all w words will be tagged with an O, since the symbols w_I and w_O do not have any transition from this state. When a G symbol comes, the automaton will transit to state 2, supposing that the correct tag for that symbol is B and to state 1, supposing that the correct tag is O. Each of these two paths will have different probability depending on the probabilities assigned to each transition. If after that symbol, a preposition is seen (visible symbol a), the system has three possibilities: go from state 2 to 3 with an I tag, from 2 to 1 with an O tag, or from 1 to 1 also with an O tag. In the last case, only the path with higher probability will be recorded, because only the best path for each B-I-O tag is recorded in each instant. This process goes on until the end of a sentence, when the most likely path is chosen and the tags for each word are recovered.

5.4 Managing Unseen Transitions

When performing the annotation of subsequences in a text, it is possible to find symbol sequences that have not been seen in the training corpus. It can imply that there is no transition available for the studied symbol in the current state. Since there are different histories grouped in the same state, sometimes there will be an available transition determined by another suffix in the state, but if this does not happen, the automaton will fall in a *sink* state, which receives all the unseen transitions. This state can be seen as the state that contains all the unseen suffixes.

When the system transitions to the *sink* state, we are interested on not loosing the studied path, so the probability of the transition from the current state to the *sink* state is not considered zero but smoothed to a small value using Lidstone's law (Manning & Schütze, 1998). Actually, the only sequences that have null probability are those that contain a suffix with the forbidden tag combination "OI", this is, all suffixes of the form $s_O^i s_I^j$.

When the automaton falls in the *sink* state, it can not follow the input sequence using state-to-state transition information because, since the transitions were not seen, they are not defined. Then, it is necessary to define a method to allow the system to continue annotating the text, and to go out of the *sink* state as soon as possible. In this work, we propose two different ways of dealing with the sink state, presented in next sections. The first method consists of waiting for a seen suffix and the second one in looking for suffixes sharing the same ending.

Waiting for a Seen Suffix

This first method propose to keep reading the text input and building all possible suffixes till some of the built suffixes is found in an existing state. To do so, when the automaton falls into *sink* state, the suffix of length l_{max} is built concatenating the last $l_{max} - 1$ symbols with the next symbol from the input, which is the symbol that made the automaton fall into the sink state. This first suffix for sure will not be in any state in the automaton, but then the next symbol (word) of the input is studied creating the new suffix in the same way. Then, a state containing this new suffix is searched over the automaton and, if found, the automaton goes to this state and continues its normal functioning. If not, the process is repeated, getting more symbols from the input sequence, until a state containing the new suffix is found. Note that for each visible symbol taken from the input, three new suffixes are created, one for each possible complete symbol (combination of the visible symbol with the B-I-O tags).

This may cause skipping some part of the input, since the suffixes not found in the automaton are considered to have a smoothed probability so we can not say which is the correct tag for them. This is caused by the fact that the text sequence is considered as a stationary process, and so, when the CSSR-acquired automaton fails, we have to resynchronize it with the input data.

Looking for a Suffix Sharing the Same Ending

Another possibility to fix the problem of falling into the *sink* state is to study the unseen suffix that has made the automaton evolute to the sink state and look for other suffixes sharing the same ending. If the suffixes are long enough, we can consider that shortening a little bit the history taken into account can not lead to a high loose of information while, on the other hand, let the system continue its normal functioning without waiting for a seen suffix.

Note that in this case, there could be more than one state sharing the same ending of the unseen suffixes. Then, the system considers all the possible states and continues all the paths from each state and in the next step only the paths with higher probabilities will be kept.

5.5 Validating the Method

A preliminary experiment to validate the proposed approach was performed. The goal of this preliminary experiment is to see if CSSR is able to learn correctly the patterns created by a hand-written automaton and to evaluate the attained performance when using it to annotate new text.

To do so, we focus on Named Entity Recognition task for Spanish, for which FreeLing analyzer (Carreras et al., 2004; Atserias et al., 2006) has a system that uses a simple hand-built automaton of four states. The alphabet used in this automaton is the same –and encode the same features– than the one presented in 4.2. We use this system to re-annotate both a training and a test corpora, obtaining corpora annotated with a simple and systematic annotation criteria. Note that now we are not looking for the real NEs, but we just want to check if CSSR is able to reproduce the behavior of FreeLing automaton, independently of the correctness of the FreeLing tags.

The corpora we use are those of CoNLL-2002 shared task (Tjong Kim Sang, 2002a) for Spanish, that will be also used and better introduced in chapter 6. This data-set contains a training corpus, one corpus for the development of the system and another one for the evaluation. In this case, we will test our system on both the development and evaluation corpora, since we are not developing a system, just checking the validity of our method. Actually, these experiments could be done with any text, since the manual NE tags are completely ignored at this moment and just the FreeLing tags are taken into account.

Thus, these three corpora with the NEs detected by FreeLing analyzer are the data used to test our proposal for using CSSR at subsequence detection tasks. The vocabulary we use is the same presented in the example 5.2.1 that combines the visible part of the symbol introduced in 4.2 with each possible hidden tag: G_B, G_I, G_O, S_B, S_I, S_O, M_B, M_I, G_B, G_I, G_O, S_B, S_I, S_O, M_B, M_I, M_O, a_B, a_I, a_O, w_B, w_I, w_O. Since the visible part of the alphabet is the same than the one used by FreeLing NER system, the features taken into account are the same and CSSR is expected to be able to learn automata that reproduces correctly the behavior of this hand-build NER system.

Using the training corpus annotated with FreeLing translated into the complete alphabet, two automata were learned using CSSR algorithm: one for $l_{max} = 2$ and one for $l_{max} = 3$. Then, these automata were evaluated over both test corpora (also annotated with FreeLing) with the method explained in 5.3. The system obtained $F_1 = 100\%$ when using $l_{max} = 2$ for both test sets, and using $l_{max} = 3$, $F_1 = 99,83\%$ for the CoNLL-2002 development corpus and $F_1 = 99,98\%$ for the CoNLL-2002 evaluation corpus were obtained. For $l_{max} = 3$, the lost in F_1 is due to some missed NEs. In fact the system obtained 100% precision in both cases, but the recall fell a little bit. This lose in the recall, could be due to the fact that if l_{max} rises, more training data are necessary to generate a correct automaton.

These results prove that CSSR is able to perfectly acquire the behavior of the FreeLing annotation schema underlying the data. Although this is an easy task, since that schema is simple and systematic, it validates the viability of our adaptation of CSSR from stationary-process acquisition to pattern recognition. In next chapter we will present and discus the performance of this schema when applied to real data.

5.6 Conclusions of this Chapter

In this chapter, we have presented our approach to use CSSR in NLP annotating tasks. So far, we have focused on tasks that imply subsequence detection. The proposed approach consists basically of introducing the hidden information about where the subsequences begin and end into the alphabet and learn an automaton containing this information. Then, the automaton can be used to annotate new sentences using a Viterbi algorithm. Also, two methods for dealing with unseen events when performing the annotating step have been presented.

An important issue to take into account is that the fact of introducing the hidden tags into the visible alphabet makes the alphabet three times bigger than the visible alphabet. In chapter 4 we have already discussed that one of the limitations of CSSR is that the amount of data necessary to learn a good automaton grows exponentially with the size of the alphabet. Now, we are multiplying the alphabet size by three, in respect with the experiments performed in chapter 4 when no hidden information was taken into account. This means that the amount of necessary data 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 annotation task under the B-I-O approach.

Next chapter will present the experiments performed with this approach to annotate Named Entities and to detect the sintactic chunks in a text. Then, at the sight of the results we will better discuss the limitations of the proposed method and will study the effect of having a limited amount of data.

CHAPTER 6

Experiments and Results with CSSR in Annotation Tasks

In this chapter, the performed experiments and the obtained results of applying CSSR to annotation tasks under the approach proposed in chapter 5 are presented.

First of all, we summarize in sections 6.1 and 6.2 the different system configurations that we will use and test, as well as the measures and corpora used to determine the performance of each configuration. Then, we explain in detail the experiments and results with the three addressed tasks. The tasks we are interested in are Named Entity Recognition (NER) both in general and medical domain, and Text Chunking.

Some of the experiments with NER in general domain presented here were published in (Padró & Padró, 2005a; Padró & Padró, 2005c). The application of CSSR to biomedical NER was presented in a poster at BioLink'07 (Dowdall et al., 2007) and the Chunking experiments can be found in (Padró & Padró, 2005b).

6.1 Experimental Settings

So far we have explained the method we use and our proposal to apply it to NLP annotation tasks. We have focused on the main ideas behind the method and we have also explained which are the most important parameters for the algorithm. Nevertheless, when applying it to the real tasks, we found some useful variations of the original system and other parameters that also could influence the performance of the system. In next sections we review all these paremeters and enumerate all the different settings that we have used in our experiments.

6.1.1 Maximum Length

As we have already said, one of the main parameters of the CSSR algorithm is the maximum length l_{max} the suffixes can reach. All the experiments will be performed with various l_{max} values in order to study the influence of this parameter.

6.1.2 Hypothesis Test

Another influent issue is the confidence degree of the hypothesis test and the hypothesis test itself. Our first experiments were performed with χ^2 test, and with various values of its confidence degree α. But, as we have said in chapter 4, this test depends on the amount of available data, what leads to the phenomenon that if we give more significance to the data replying the data many times, as in section 4.3.2, the system can improve its performance. In this way, we propose two lines to be studied:

1. In the experiments to tag subsequences of data, apart from using various α values for the hypothesis test, we simulate that we have more data multiplying the χ^2 by a pre-established value β each time we perform the hypothesis test. It can be proved from the χ^2 equation that if β times more data is available, the χ^2 value will be multiplied also by β. This is, then, a simple way to simulate that we repeat the same corpus β times, which is the same to say that we give more statistical reliability to the available data. As discussed in section 4.3.2 this could be expected to be equivalent to modify the confidence degree of the χ^2 test but experiments show that introducing β modifies significantly the behavior of the system obtaining better results than modifying only α. Thus, we see this β value as another parameter that can be tuned to obtain the best possible performance of the system. In fact, giving more weight to the training corpus in this way is equivalent to accept that this corpus is representative of the real population, this is, to accept that the Maximum Likelihood Estimator is valid. If this is true, the results will improve when β is tuned, if not, they will be worse. We will perform several experiments with various β values to see how influent this parameter can be and we will see that this can improve the performance of the system.

2. Given that χ^2 is very dependent on the size of the corpora, and that it can be a poor test when small amount of data is available, we also propose to use another hypothesis test. In this case, we use the Jensen-Shannon divergence (Lin, 1991), that gives the distance between two distributions. Then, we consider that two future probability distributions are different if the distance between them is higher than a determined threshold d. Note that this is not a test, so in this case the parameter to be tuned is not the confidence degree α but the allowed distance between distributions d, which is conceptually different. We are specially interested in using this distance because it will be very useful when introducing ME models into CSSR, as will be presented in chapter 7.

Thus, our experiments can be performed with two different ways of determining which suffixes belong to the same state: the χ^2 hypothesis test, in which case we will tune two parameters (α and β) and the Jensen-Shannon divergence with its parameter d.

The difference between the parameter α for χ^2 test and the parameter d for the Jensen-Shannon test is that α represents a confidence degree for the hypothesis test, that is, the confidence with which two histories can not be considered to have different probability distribution for the future, and can vary from 0 to 1. The bigger α is, the more exigent is the test to consider two distributions not different, what for CSSR means that less histories will be considered to have the same probability distributions for the future so to belong to the same causal states. Thus, the bigger α is, the more states the automaton will have. On the other hand, the Jensen-Shannon divergence can take any value from 0 to ∞, so

the chosen threshold d can be any positive value, and represents the threshold under which two distributions are considered not different. Then, and opposite to the χ^2 test, if the threshold d rises, it means that we are being less exigent to consider two histories to have not different probability distributions, so more histories will be grouped into the same state and the automaton will have less states.

6.1.3 Sink Management

As explained in section 5.4, we propose two different ways of managing the unseen transitions and going out of the sink state that will be tested in the proposed tasks.

The first method consist of skipping symbols from the input until a seen history is found and the automaton can evolute to its state (we will refer to this method as *sink-1*) and the second one looks for suffixes in other states with the same ending than the unseen suffix (*sink-2* method) .

Nevertheless, the experimental results will show that most of the times that the system falls into the sink state, it is not because the current suffix has not been seen in the training corpus but because it has been deleted from the automaton due to the way in what CSSR builds it, as it will be explained in the following section. Thus, many of the times that the system falls into the sink state can be avoided changing a little bit the algorithm.

6.1.4 Adjusting State Deletion

In section 3.3 we thoroughly illustrate how CSSR builds the automaton. Among other steps, we discuss the way in which CSSR determines the recurrent states, which can lead to wrong generalizations (see section 3.3.4). To summarize, CSSR determine recurrent states by studying which states receive transitions from other states. The original CSSR implementation studies only the transitions for the suffixes of length l_{max} -1, since these are the suffixes that can be lengthened, with each symbol, to build an l_{max} suffix. Then, this longer suffix is searched in the automaton, and the state containing it receive the transition from the state containing the short suffix with the given symbol.

In the examples presented in section 3.3, we also test an alternative implementation, that uses all the suffixes in all the states to determine recurrent states. This implies that less states are deleted, loosing generalization power but keeping states that may be useful in some cases.

The first way of determining the recurrent states is more theoretically justified, and is necessary for CSSR to perform generalizations. This is, an automaton that can involve infinite subsequences, will not be generalized correctly if we change this approach. Though the second implementation leads to some loose in CSSR generalization power, as NLP subsequences are always finite, we are interested in considering also the use of this second approach.

In fact, in our experiments, we found that the first approach causes an actual problem, since there are some states that are deleted because they are considered transient, but they contain suffixes that are necessary in the annotation step. Loosing these suffixes makes the system fall into the sink state more often than if these states were not deleted. These states are only deleted if the only transitions studied to determine the recurrent states are those of the suffixes of length $l_{max} - 1$. If the transitions for all suffixes are considered, then more states receive some transitions from long suffixes and are not deleted.

Despite of the fact that, as it is explained in section 3.3, the correct way of building an automaton with CSSR is studying just the shortest suffixes, considering the experimental results we also tried to learn automata that study the recurrent states taking into account all the suffixes. Doing so leads to a system unable to represent systems that can involve infinite subsequences, but as it is not the case of NLP tasks, we consider that the loose in the generalization power of CSSR that considering all the histories may introduce, can be better than removing suffixes from the automaton that may be necessary in the annotation step. In section 6.3 we will study the results obtained with the different implementations and discuss the experimental influence of this issue.

6.1.5 Synthesis

Those are the different settings we propose for our system. To summarize, we have four different ways of building the automata: the two different hypothesis tests combined with the possibility of studying just the short suffixes when determining recurrent state or studying all the suffixes. Also, we have two methods for the annotation step, which are the two different ways of dealing with the sink state.

In total, we can make eight combinations that are the eight different methods that we will test. Each method will be tested with various l_{max} values and with various significance level and β if performing χ^2 test or with various d when using the Jensen-Shannon divergence.

6.2 Evaluation

Next sections will present the concrete experiments and results obtained at each of the NLP tasks approached in this work. All these tasks are subsequence detection tasks, and, as it has been said in section 5.1, we will use the "B-I-O" approach to represent these subsequences.

To determine quantitatively how well our system annotates new text, we will use the typical measures Precision (P), Recall (R) and F_β. Precision measures how many subsequences are correctly tagged over the total number of subsequences annotated by the system (this is, the sum of the correctly annotated sequences and the incorrectly annotated ones). Recall counts the number of correct subsequences over the number of actual subsequences of the corpus. F_β is the harmonic mean of this two measures:

$$\begin{aligned} P &= \frac{\text{Correct Subseq.}}{\text{Annotated Subseq.}} = \frac{\text{Correct Subseq.}}{\text{Correct Subseq.} + \text{Incorrect Subseq.}} \\ R &= \frac{\text{Correct Subseq.}}{\text{Actual Subseq.}} = \frac{\text{Correct Subseq.}}{\text{Correct Subseq.} + \text{Missed Subseq.}} \\ F_\beta &= \frac{(1+\beta^2)P \cdot R}{\beta^2 \cdot P + R} \end{aligned}$$

In our work we will use $\beta = 1$:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

A perfect annotation system would have $P = R = F_1 = 100\%$. All these measures count whole subsequences, if a subsequence is partially marked but is not exact, it is not counted

as a correct subsequence.

For NER in general domain and Chunking, we will use the data of two competitions devoted to those tasks. In both cases, there are three corpora: one for the train, one for the development and one for the evaluation. The training corpus is used to train the automata with various parametrization. Then, the various automata are tested over the development corpus to determine which is the best configuration. Once it has been chosen, this best configuration is tested over the evaluation corpus, in order to have a measure of the performance of the system over a corpus that has not been used to seek for the best parameters. This last result is used to compare the performance of the various systems in the competition. In our case we will give also the results over the evaluation corpus for different maximum lengths, since we think it may be of interest for the reader.

In the case of NER for biomedical domain, we will use ten-fold cross validation, since this is the method used by the systems we want to compare to. Thus, we divide the corpus in ten parts and use nine for the training of the system and one for the test. Then, the process is repeated changing the part that is used for the test until we have ten different results obtained with ten different combinations of train/test corpus. The given performance measures are the average of the results obtained with each combination.

6.3 Named Entity Recognition

This section presents the concrete experiments and the obtained results in the NER task. This task is approached with all the possible system settings presented in previous section.

6.3.1 Data

To perform the experiments in NER task, the data for the CoNLL-2002 shared task (Tjong Kim Sang, 2002a) for Spanish are used. These data contain three corpora: one for the train one for the development of the system and the other one for the evaluation. The amount of data in each corpus and some interesting statistics are shown in table 6.1.

Corpus	Number of words	Number of sentences	Number of NEs	Words per sentence	NE per sentence	Words per NE
Train	264,715	7,263	18,797	36.4	2.6	1.8
Development	52,923	1,639	4,351	32.3	2.7	1.7
Evaluation	51,533	1,364	3,558	37.7	2.6	1.7

Table 6.1: Number and ratios of sentences, words and NEs in each corpus.

6.3.2 Baseline

First of all an experiment using a basic system was performed in order to set a baseline for our work. To do so, the NEs of both test corpora were tagged using FreeLing analyzer (Carreras et al., 2004; Atserias et al., 2006). This system uses a simple hand-built automaton of four states. Since the system is hand-built, in this case the training corpus was not used. The results obtained with FreeLing over the two test corpora are presented in table 6.2.

These results are not very high because the system used is very naive and only recognizes simple Named Entities. Nevertheless it may be useful to see if the automata learned via CSSR are better than this hand-made automaton.

	Precision	**Recall**	F_1
Development	79.83	88.32	83.86
Evaluation	79.14	90.02	84.23

Table 6.2: Obtained results using FreeLing analyzer over the two test corpora.

6.3.3 Alphabet

The alphabet used to build automata using CSSR algorithm to perform Named Entity Recognition was the same used in the example 5.2.1, also used to validate the proposed method (section 5.5): G_B, G_I, G_O, S_B, S_I, S_O, M_B, M_I, M_O, a_B, a_I, a_O, w_B, w_I, w_O.

6.3.4 Experiments and Results

Various experiments with the eight different settings presented in section 6.1 and with different values for the parameters were conducted. First of all, we made the experiments using χ^2 test and using only short suffixes to determine the recurrent states, as it is in the original CSSR implementation. Nevertheless, we found that this system had a strange behavior, specially for long l_{max} (3 or 4) and studying more accurately the error sources, we realized that the problem was due to the fact that the system deleted necessary suffixes because they belonged to recurrent states.

To avoid that problem, we changed the way in which the recurrent states are determined, studying the transitions for all suffixes, as it has been explained in section 6.1.4. This led to a better performance of the system. Though the best results are similar, the second system is much better behaved for long maximum length values and the results depend less on the parameters.

Furthermore, we were interested in testing the use of Jensen-Shannon divergence as an alternative to χ^2 hypothesis test. We repeated the experiments using short and long suffixes to determine the recurrent states. The first approach produced similar results to those of χ^2 test for $l_{max} = 1$ or $l_{max} = 2$. For longer l_{max} values, though the system was less dependant on the parameters, the obtained results were far behind the best results obtained using χ^2. Nevertheless this loose in performance was solved when using all suffixes to determine the recurrent states, obtaining a similar performance to the χ^2 system that uses also all suffixes.

In all cases, also the two sink methods, *sink-1* and *sink-2* were tested. In general, the results obtained with both methods are similar, specially the best results, but the second method makes the system a little less dependent on the parameters, specially when using short suffixes to determine recurrent states.

Next sections present the detailed results for each possible system configuration introduced in section 6.1. This is, using χ^2 test or Jensen Shannon divergence both with short and long suffixes when determining the recurrent states. Also the two sink methods presented in section 6.1.3 are tested in each case.

Using χ^2 Test and Short Suffixes to Determine Recurrent States

This is the first used method because it is the original proposal of Shalizi and Shalizi (2004). In fact, an implementation of CSSR made by Shalizi and Shalizi (2004) is available on the web[1] which offers two different hypothesis test: the Kolmogorov-Smirnov test and the χ^2 test. After performing some experiments with this software in NER task (Padró & Padró, 2005a) it was seen that the results did not change significantly using one or the other test, so we developed our own implementation of CSSR[2], in order to test the other settings presented above.

In this case, there are three parameters that must be tuned to use CSSR: the maximum length (l_{max}), the confidence degree of the test α and the parameter β that we introduce as a way of giving more significance to the available data. Various experiments with different values for these three parameters are performed. The search for best parametrization is performed over the development corpus, and then, this configuration is tested with the evaluation corpus.

Number of States

The first issue we are interested in studying when using CSSR for NER task, is the number of generated states. This number depends on the three parameters mentioned above, so we have many automaton with different sizes. Figure 6.1 shows how the parametrization of the model affects the size of the generated automaton, taking as an example the variability on the number of states when varying α maintaining $\beta = 1$ and also the dependence on β when α is set to 0.01. Other combinations have been also studied but the general behavior is the same showed in this figure.

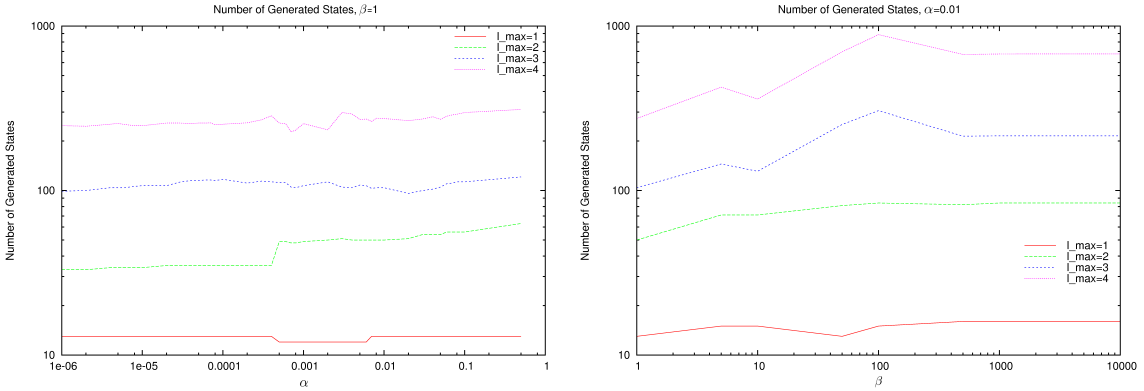

Figure 6.1: Number of states of the learned automata with χ^2 and short suffixes using various l_{max}, α and β values.

From these figures it can be seen that the most influent parameter in the number of generated states is the maximum length of the suffixes. Note that both axis have a logarithmic scale, so the number of states grows exponentially with l_{max}. This is because the number of studied suffixes also grows exponentially with the considered maximum length, since for an alphabet of size k there are $k^{l_{max}}$ possible suffixes of length l_{max}. If our data were created by a perfect automaton, as in the examples studied in section 3.3, increasing

[1] http://bactra.org/CSSR/

[2] Also available under GPL license at http://www.lsi.upc.edu/~mpadro/cssr_en.html

the maximum length would not imply to increase the size of the automaton, since CSSR is able to capture exactly the automaton behind the data, but in our case, the structure behind the data is much more complicated, what means that when using longer maximum lengths, the system is able to capture longer patterns not captured with small l_{max} and the number of states grows because there are many new suffixes with different behavior and different probability distribution taken into account.

Also, it can be seen that the α value changes the number of states of the generated automata but not as much as l_{max}. As it was expected, the larger α is, the greater the number of states will be. β is also influent in the number of states, in fact more than α, what means that modifying β affects more the performed hypothesis test than varying the significance level of the χ^2 test.

Annotation Results

Once the different automata with the various parameter values have been created, they can be used to tag new sentences. At this step, we have two possible methods to deal with the sink state: *sink-1* and *sink-2*.

Figure 6.2 shows the results obtained for the development corpus with the first method for dealing with unseen transitions and figure 6.3 shows the performance when using the second method. In both cases, the left plot shows the dependence on α when setting $\beta = 1$, which is equal to not giving any extra weight to our data, and the right figure shows the dependence on β when α is set to 0.01. If α is modified, the behavior of the system in respect to the parameter β does not change significantly. Otherwise, if β is set to higher values, the system becomes less dependent on α, as shown in figure 6.4

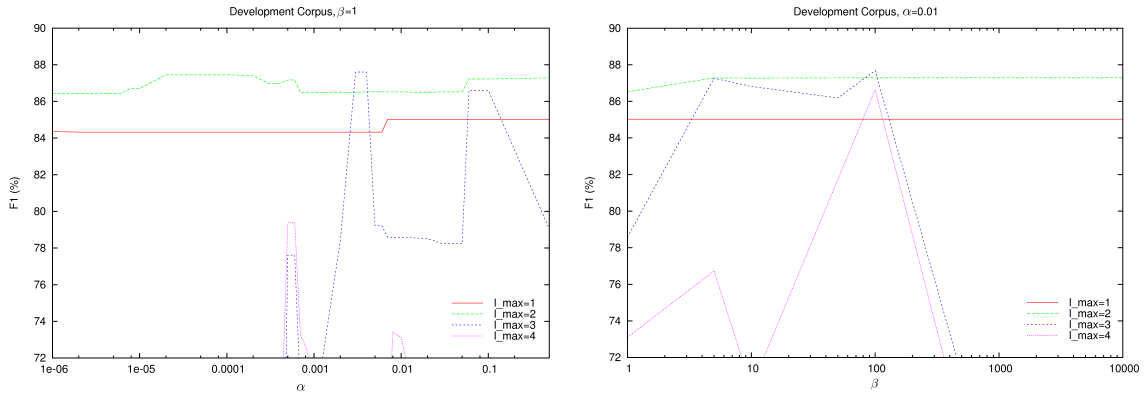

Figure 6.2: F_1 scores obtained when annotating NEs with CSSR, using χ^2 test, short suffixes and *sink-1* method, with various l_{max}, α and β values.

From these figures it can be seen that while for $l_{max} = 1$ and $l_{max} = 2$ the obtained results do not vary very much with the various parameters nor the chosen method to deal with the unseen transitions, for bigger maximum lengths the behavior of the system becomes very unstable, except for $\beta = 100$, when the system is stable respect the parameters for all lengths. Furthermore, in that case, the best performance is obtained with $l_{max} = 3$, while in the other cases this length only performed better than $l_{max} = 2$ for some specific configurations.

The best obtained results, and the configuration that leads to them for *sink-1* method are shown in table 6.3. In this table, also the results of the best system applied over the

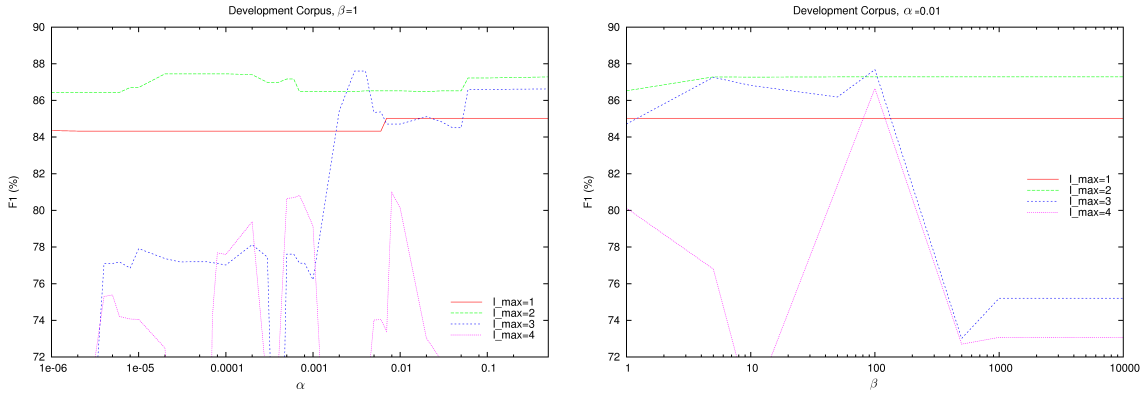

Figure 6.3: F_1 scores obtained when annotating NEs with CSSR, using χ^2 test, short suffixes and *sink-2* method, with various l_{max}, α and β values.

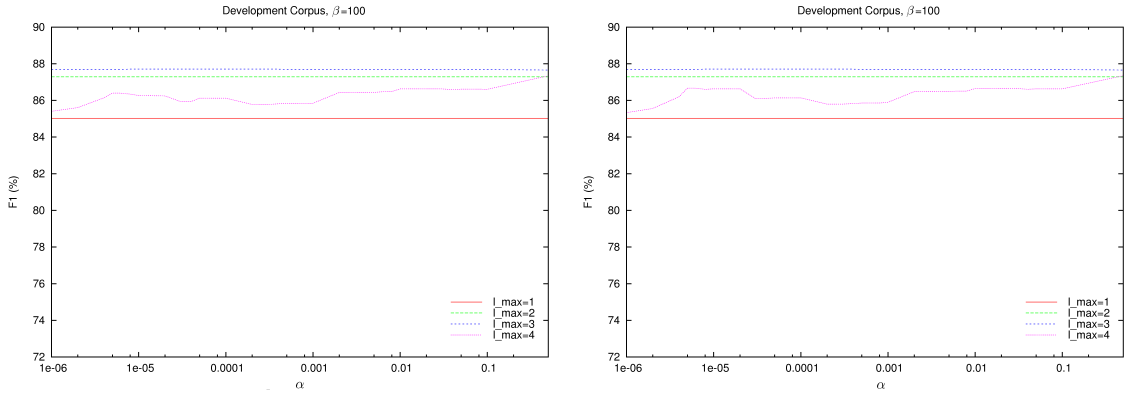

Figure 6.4: F_1 scores obtained when annotating NEs with CSSR, using χ^2 test and short suffixes with $\beta = 100$. The left figure shows the results for *sink-1* and the right one the results for *sink-2*.

evaluation corpus are shown. We maintain a special column for the case of $\beta = 1$ to study the influence of introducing the extra parameter β into the system.

For *sink-2* method the best results are the same with the same configuration, what means that the best obtained results are independent of the method used to deal with unseen transitions but, as we have ready seen, the system performance is more stable with respect to α and β with the second method. In the case of using $\beta = 1$, the results are slightly different respect the ones obtained with *sink-1*, but only for $l_{max} = 4$. In this case, the system obtains $F_1 = 80.99$ for the development corpus with $\alpha = 0.008$ and $F_1 = 82.89$ using the evaluation corpus with this configuration.

From table 6.3 it can be seen that the difference between using $\beta = 1$ and the β that leads to the best result is not significative (at a 95% confidence degree) except for $l_{max} = 4$. In this case, the gain in F_1 when looking for the best β value is of more than 8%, allowing the automaton of $l_{max} = 4$ to be competitive with the other automata, what not happens if we consider just $\beta = 1$.

l_{max}	**Best Parameters** α	β	**Development Corpus** **P**	**R**	F_1	**Evaluation Corpus** **P**	**R**	F_1
With $\beta = 1$								
1	0.07 – 0.1	1	83.66	86.44	85.03	85.34	88.00	86.65
2	0.00002 – 0.0002	1	88.99	86.05	87.45	89.26	87.63	88.44
3	0.03 – 0.04	1	89.18	86.07	**87.60**	89.54	87.83	**88.68**
4	0.0005 – 0.0006	1	84.24	75.07	79.40	85.61	79.90	82.66
With best β								
1	0.07 – 0.1	$1 - 10^6$	83.66	86.44	85.03	85.34	88.00	86.65
2	0.1	10	89.24	86.23	87.71	89.58	87.91	88.74
3	$5 \cdot 10^{-6}$– 10^{-5}	50	89.33	86.26	87.77	89.52	87.91	88.71
4	$5 \cdot 10^{-6}$	10	89.37	86.26	**87.79**	89.54	87.86	**88.69**

Table 6.3: Best F_1 scores obtained for each l_{max}, using χ^2 test, short suffixes and *sink-1* method over the two test corpora.

The best results obtained with l_{max} equal to 2, 3 and 4 are not significantly different neither in the development nor in the evaluation corpus.

Then, we can conclude that parameter β is more influent for large l_{max} values, allowing CSSR to learn much more accurate automata when more significance is given to the data. This is logical, and coherent with the experiments performed in 4.3.2, where generating more data (equivalent to giving more weight to the existent data) allowed CSSR to learn better automata for long maximum lengths while the automata learned with short lengths did not improve significantly. It is logical, because it is for large l_{max} that the lack of data can be a problem, since we have not enough data to perform good statistical tests using χ^2 and giving more weight to the data is a way of avoiding this problem. The best performance is obtained with relatively small β values, between 10 and 50. For β between 50 and 100 in general the performance does not fall significantly but when β is set to larger values it begins to be worse. This means that if we give to much significance to our data, the system learns incorrect automata, since it is giving too much weight to observation for which we do not have enough real evidence.

From the table it also can be seen that it seems that the best configuration of the parameters is quite arbitrary, it is not possible to say that the system perform better with determined α or β, since it depends on the maximum length and, in fact, there are various configurations leading to same results. In fact, α and β are related, since both modify the performed hypothesis test, so it is logical that the best values for β with a determined α are not the same as the best β values for another α. But as we have said before, the larger l_{max} is the more influent are the parameters, specially β being more narrow the intervals in which the best performance is obtained and being bigger the difference between the score obtained with the different configurations.

Sink State

We have already observed that the results for $l_{max} = 3$ and $l_{max} = 4$ have a stranger behavior in their dependence on α and β, and, in general, very worse performance than

the other results. It could be expected to have bad accuracy with large l_{max} values, since the amount of available data may be insufficient to perform good statistics with such long suffixes, but we have seen that the best results obtained with these lengths are competitive with the ones obtained with $l_{max} = 2$. This suggests that there is some other problem, apart from the limited amount of data, that makes the system behave strangely. When studying in detail the error sources, it was seen that partly, the strange behavior is due to the fact that for longer l_{max} values, the system falls more often into the sink state. Figures 6.5 and 6.6 show the number of times the system falls into the sink state depending on the studied parameters for both methods of dealing with unseen suffixes.

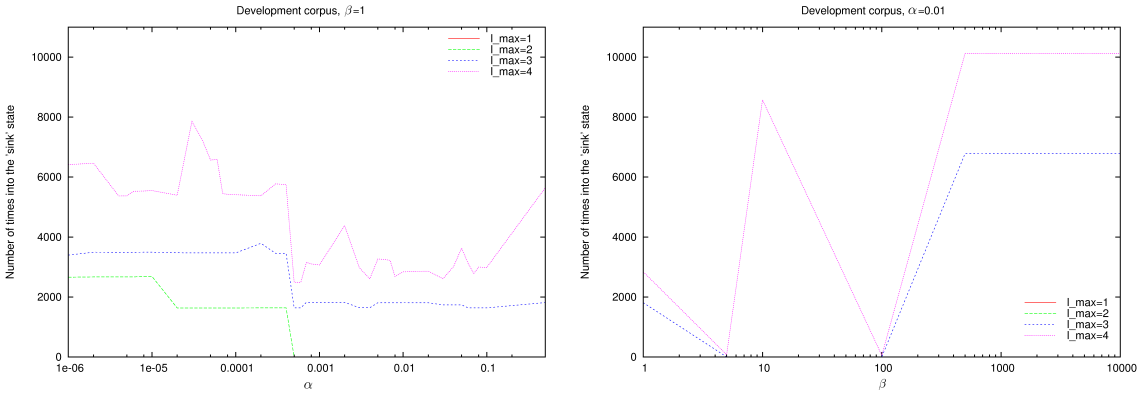

Figure 6.5: Number of times the system fell into the sink state, using χ^2 test, short suffixes and *sink-1* method, with various l_{max}, α and β values.

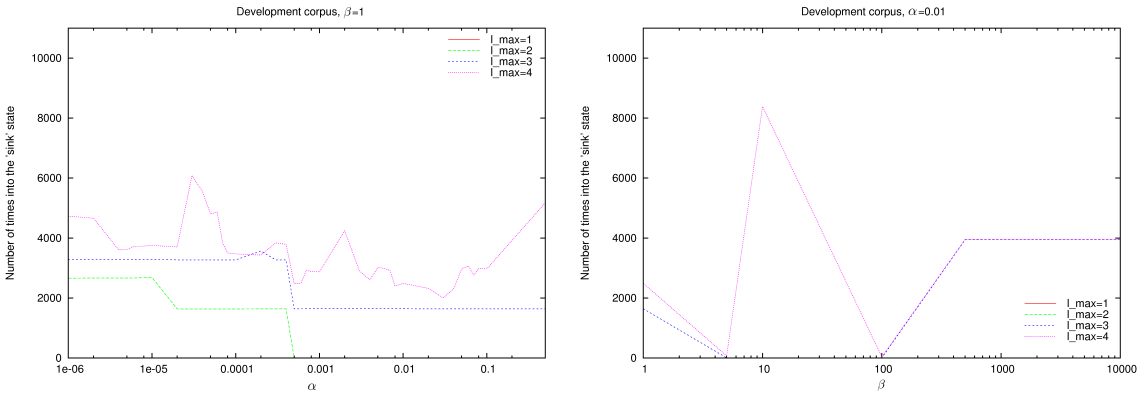

Figure 6.6: Number of times the system fell into the sink state, using χ^2 test, short suffixes and *sink-2* method, with various l_{max}, α and β values.

It can be seen that the behavior for both methods is similar, since the automaton used is the same, so the unseen suffixes will be found in both methods. The difference in the count of sink states is due to the fact that, when the system is not able to go out from the sink state for some consecutive words, it is computed as if all these words caused the system to fall into the sink state, as in fact if the system is still into the sink state it means that we still have an unseen suffix. Note, then, that the second method (figure 6.6) has in general equal or less counts into the sink state than the first method (figure 6.5). It means that *sink-2* method has been able to solve more times the unseen events than *sink-1* method.

For $l_{max} = 1$ the system never falls into the sink state. It is logical, since we are considering suffixes of length 1, and would be very unusual to find an unseen symbol in the training corpus that appears in the test corpus, at least for the simple vocabulary we are using.

It is interesting also to note that the peaks on the number of sink states coincide with the valleys in the performance. Nevertheless, there are configuration with the similar number of sink states that lead to much worse F_1 scores. This is due to the fact that the loose in the performance is not only due to the number of times the system finds an unseen suffix, but also to which are these unseen suffixes. If the system falls into the sink state for a suffix that contains an NE it probably will not tag it properly, so the accuracy would be much worse than if the sink states happen for suffix outside NEs.

To better compare the performance of the two methods for dealing with unseen suffixes, it is important to study the ratio of fixed sink states. This is, the ratio between the number of times the system fell into the sink state and was able to find another suitable state over the total number of times the system went into the sink state. This ratio is shown in figure 6.7 for *sink-1* If the ratio is equal to 1, it means that every time the system fell into a sink state, it has been fixed, so another suitable state has been found.

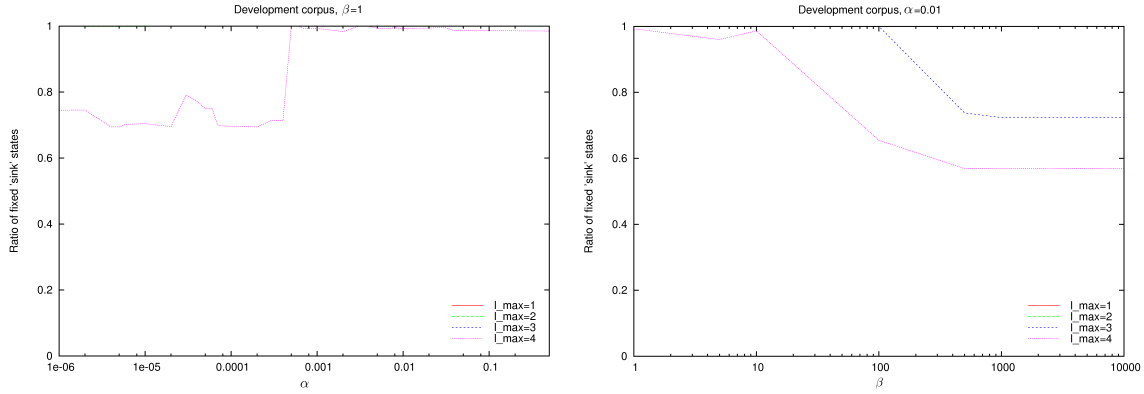

Figure 6.7: Ratio of fixed sink states, using χ^2 test, short suffixes and *sink-1* method, with various l_{max}, α and β values.

The second method *sink-2*, obtains a much better performance on fixing sink states, obtaining a ratio equal to 1 for all lengths when β is smaller than 500, and only loosing some performance for l_{max} =3 and $l_{max} = 4$ for larger values of β. This improvement of the second method could be expected, as finding a suffix with the same ending than the not-found suffix can be easy, specially if we shorten the suffix until it has length 1. This would mean that we are looking for a suffix in another state ending with the same symbol than the current suffix, which is very likely to happen.

Furthermore, what we are interested in is the evaluation of these systems on the NER performance. *Sink-2* method fixes more times the sink state than *sink-1* method, but in the performance figures, it does not seem to significantly affect. What really affects the performance is the fact of falling into the sink state, and neither of the proposed systems can be claimed to solve this correctly.

If we now study the number of times the system falls into the sink state for $\beta = 100$ we will see that the system almost never falls into the sink state. For $l_{max} = 3$ it falls into the sink state a maximum of 2 times and for $l_{max} = 4$ between 20 and 100 times for both sink methods. As shown in figure 6.4, with this β, the system have a much better performance

for all l_{max} and α values, what means that the main source of loosing performance is the fact that the automaton finds unseen suffixes in the test corpora. Furthermore, the fact that with the same length but different parametrization the system fell a different number of times into the sink state, shows that the suffixes that make the automaton fall into this special state are not suffixes unseen in the training corpus but suffixes that has been deleted from the automaton by CSSR. The parameters affect which suffixes are deleted, since the built automata are different and so are the considered transient states.

These suffixes that have been deleted from the automaton, are different suffixes for different parameters, depending on how they are distributed in states at the end of sufficiency step, more or less of them will be deleted when the transient states are determined. For that reason, the performance of the system for large l_{max} values varies so much, since depending not only on how many but also on which suffixes have been removed, the system will find more trouble when annotation new sentences.

For that reason, and once it was seen that the source of these errors was the way of determining recurrent states, we repeated the experiments taking into account all suffixes to study the transient states, in which case less states will be deleted.

Using χ^2 Test and Long Suffixes to Determine Recurrent States

When we refer to using long suffixes to determine recurrent states, we mean that, at the recurrent step, the transitions for all suffixes in each state are studied as explained in section 6.1.4. For the case of suffixes of maximum length, the suffix is shortened by the left before each symbol is attached at the right and the new formed suffix is searched in the automaton. All the states receiving a transition, this is, all states containing at least one suffix that receives a transition from a suffix outside this state, are considered recurrent and so, not deleted. If only the short suffixes are considered to study these transitions, there are less suffixes, so less states receiving transitions and more states are deleted because they are considered transient.

We present the same experiments studied in the previous section, with the same parameters and also with both sink methods, with the only difference that when CSSR determines the recurrent states, all suffixes in each state are used.

Number of States

Figure 6.8 shows the size of the generated automata when using this variation of CSSR. As expected, there the automata generated in this case are a little bit bigger than the ones generated with short suffixes, as when using only short suffixes more states were deleted. Nevertheless, this difference is no comparable to the difference in the automata size when increasing the maximum length.

Annotation Results

Again, once the automata have been created, we can apply them to Named Entity Recognition over the development corpus with both methods to deal with unseen transitions. Figures 6.9 and 6.10 show the performance when using the first and the second method respectively. As in previous section, the left plot shows the dependence on α when setting $\beta = 1$, and the right figure shows the dependence on β when α is set to 0.01.

From these figures it can be seen that the performance of the system is in general more stable when varying the parameters, specially for long maximum lengths, than when using

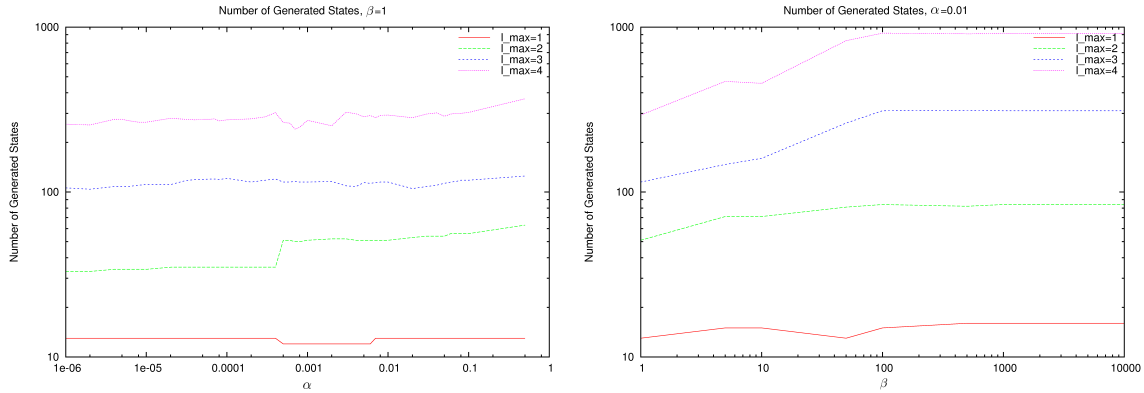

Figure 6.8: Number of states of the learned automata with χ^2 and long suffixes using various l_{max}, α and β values.

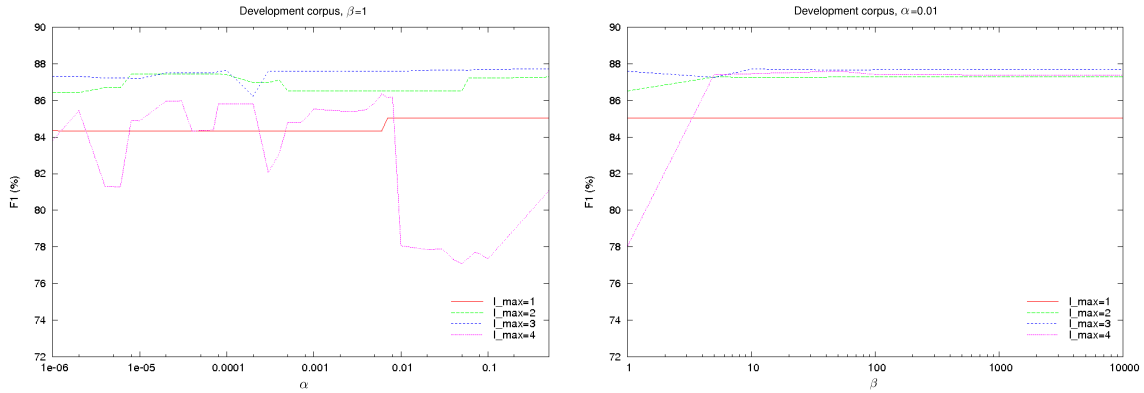

Figure 6.9: F_1 scores obtained when annotating NEs with CSSR, using χ^2 test, long suffixes, and *sink-1* method, with various l_{max}, α and β values.

only the short suffixes to determine the recurrent states. There are still some configurations that lead to a loose in the performance, but the general behavior of the system is much more stable. Also, as when using short suffixes, if β is set to higher values, the system becomes less dependent on α, as shown in figure 6.11. As in the previous case, if we study the number of times the system falls into the sink state with $\beta = 100$ we will see that this is practically zero.

The best results obtained and the correspondant configurations for the first sink method are summarized in table 6.4.

As happened when using only short suffixes, for *sink-2* method the best results are the same with the same configuration than for *sink-1*, except for $l_{max} = 4$. In this case, for $\beta = 1$, the system obtains $F_1 = 86.18$ for the development corpus with $\alpha = 0.01$ and $F_1 = 87.97$ using the evaluation corpus with this configuration. Note that these results are slightly worse than the results obtained with the first sink method. Nevertheless, the best obtained results are the same for both methods, and the second method, as happened when using short suffixes is not so dependent on the parameters.

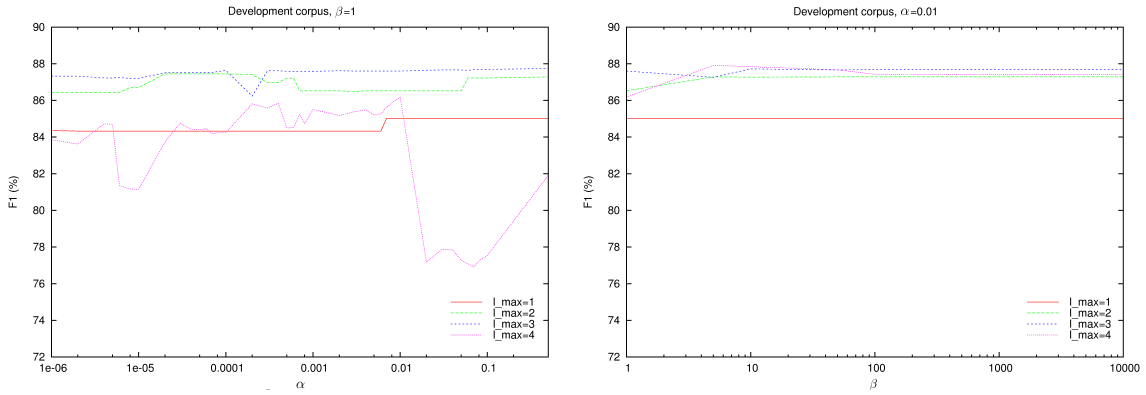

Figure 6.10: F_1 scores obtained when annotating NEs with CSSR, using χ^2 test, long suffixes and *sink-2* method, with various l_{max}, α and β values.

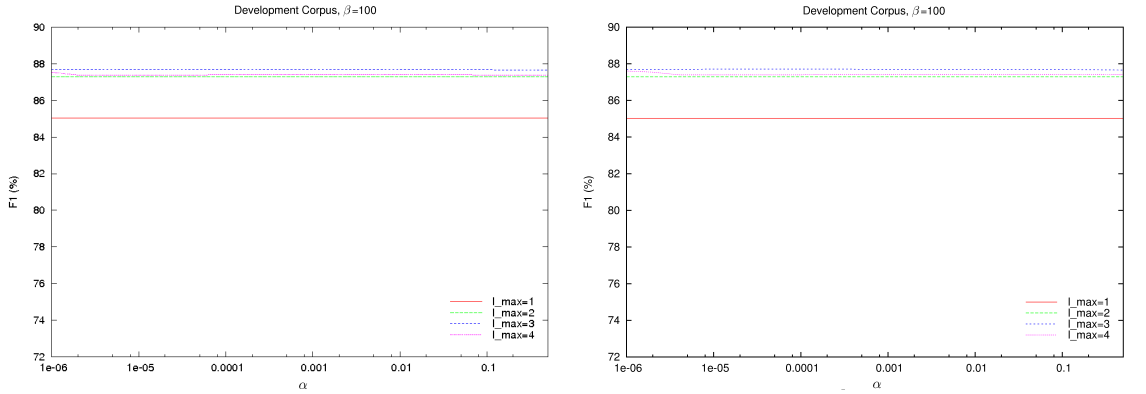

Figure 6.11: F_1 scores obtained when annotating NEs with CSSR, using χ^2 test and long suffixes with $\beta = 100$. The left figure shows the results for *sink-1* and the right one the results for *sink-2*.

Though the figures of the best values are a little bit better when using all suffixes than when using only short suffixes, the results obtained are in fact not significantly different. Nevertheless, for $l_{max} = 4$ and $\beta = 1$ the difference is significative, since when using only short suffixes the performance with these parameters was quite bad. Now, even with $\beta = 1$, the obtained results with $l_{max} = 4$ are not far from the other results.

The main conclusion of these experiments is that including all suffixes is not so good for developing a better system but for building a more robust system, not so dependent on the values of the various parameters that affect which suffixes have been deleted.

Sink State

Now, lets study how this improvement on the behavior of the system depends on the number of times the system fell into the sink state. Figures 6.12 and 6.13 show the number of times the system fell into the sink state depending on the studied parameters for both methods *sink-1* and *sink-2*.

l_{max}	Best Parameters α	β	Development Corpus **P**	**R**	F_1	Evaluation Corpus **P**	**R**	F_1
With $\beta = 1$								
1	0.07 – 0.1	1	83.66	86.44	85.03	85.34	88.00	86.65
2	0.00001 – 0.0002	1	88.99	86.05	87.45	89.26	87.63	88.44
3	0.5	1	89.34	86.19	**87.74**	89.60	87.94	**88.77**
4	0.006	1	88.41	84.39	86.35	89.38	87.55	88.46
With best β								
1	0.07 – 0.1	$1 - 10^6$	83.66	86.44	85.03	85.34	88.00	86.65
2	0.1	10	89.24	86.23	87.71	89.58	87.91	88.74
3	$5 \cdot 10^{-6}$– $2 \cdot 10^{-5}$	50	89.33	86.26	87.77	89.66	88.00	88.82
4	0.006	5	89.34	86.56	**87.93**	89.29	87.89	**88.58**

Table 6.4: Best F_1 scores obtained for each l_{max}, using χ^2 test, long suffixes and *sink-1* method over the two test corpora.

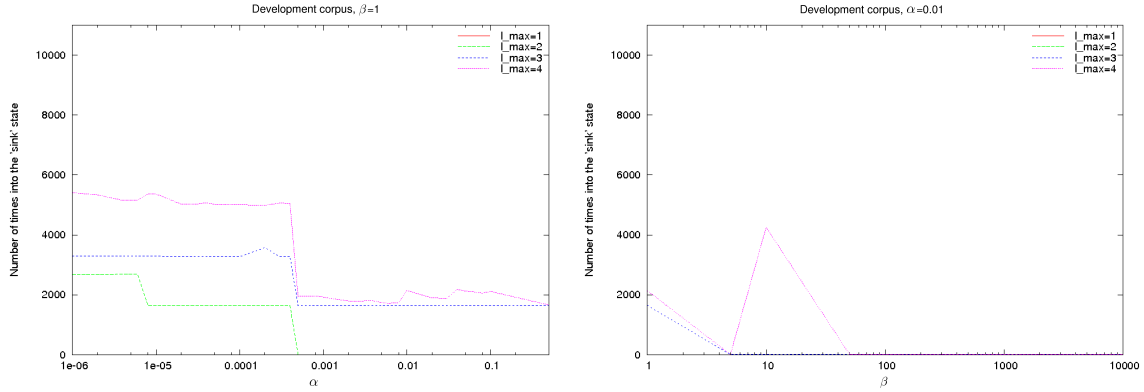

Figure 6.12: Number of times the system fell into the sink state, using χ^2 test, long suffixes and *sink-1* method, with various l_{max}, α and β values.

It can be seen that the number of times the system falls into the sink state are much less than when using only short suffixes to determine recurrent states. This is due to the fact that there are more suffixes in the automaton, as more transitions have been studied and less states have been considered transient.

Figure 6.14 shows the ratio of fixed sink states for *sink-1* method. As in previous section the second method solves more states than the first one, in this case the ratio is 1 for all the cases. Nevertheless, this does not revert on the performance of the system significantly.

In general, considering all the results obtained using χ^2 test, it is clear that this test is very dependent on the amount of data, being very influent the extra weight β that we give to the data, specially for long l_{max}. At the sight of that, we considered the possibility of using another method to compare state distributions, independent of the amount of data to see if the system becomes more robust in that way. For that reason we used Jensen-Shannon divergence to evaluate the distance between two distributions, and repeated all

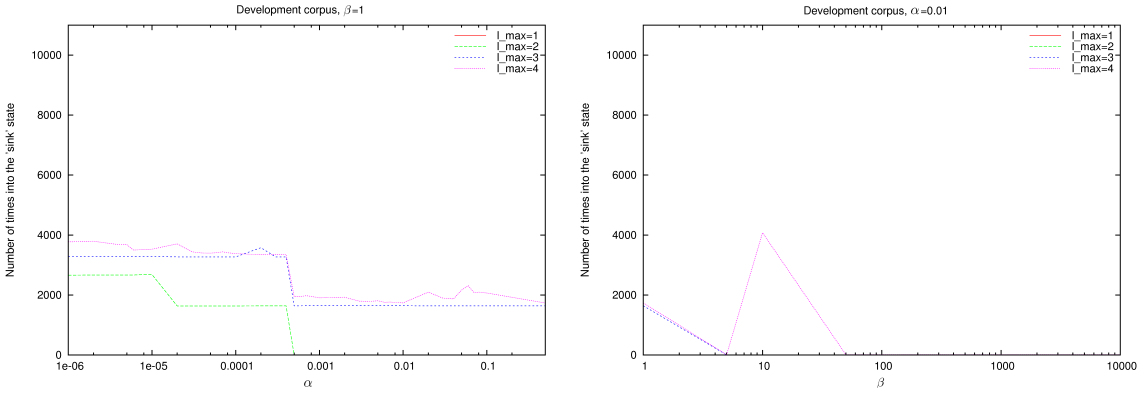

Figure 6.13: Number of times the system fell into the sink state, using χ^2 test, long suffixes and *sink-2* method, with various l_{max}, α and β values.

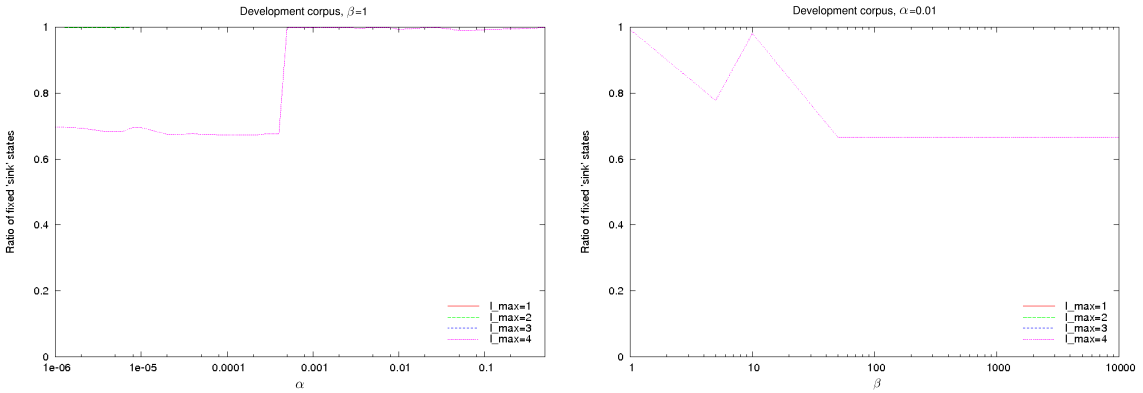

Figure 6.14: Ratio of fixed sink states, using χ^2 test, long suffixes and *sink-1* method, with various l_{max}, α and β values.

the experiments with all possible configurations used with χ^2 test. Next sections present the results obtained using these divergence both with short and long suffixes.

Using Jensen-Shannon Test and Short Suffixes to Determine Recurrent States

Once the χ^2 test has been tested with the various proposed methods, we are also interested in studying the performance of Jensen-Shannon divergence as the hypothesis test. In this case, the test is not dependent on the amount of data, so the parameter β has no influence. The parameter that controls the test is d, the minimum distance over which we consider that two histories have different probability distribution for the future.

Number of States

Using this method, the size of the generated automata is bigger than using χ^2 test, as it can be seen in figure 6.15. Note that in this case we do not have an upper bound to the parameter d but the presented figures show the range in which CSSR performs better.

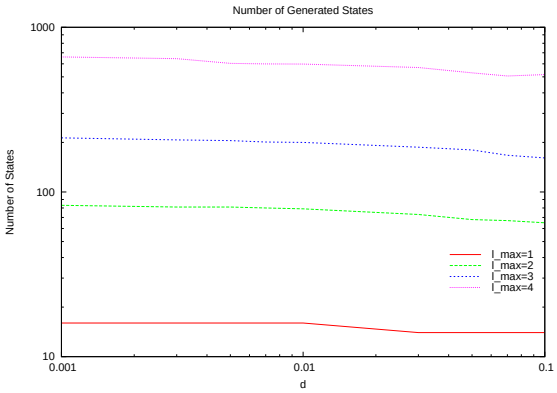

Figure 6.15: Number of states of the learned automata with Jensen-Shannon divergence and short suffixes using various l_{max} and d values.

Annotation Results

When these automata are applied to NER task over the development corpus the results shown in figure 6.16 are obtained. In this figure we can compare both systems for dealing with the sink state (*sink-1* at the left and *sink-2* at the right).

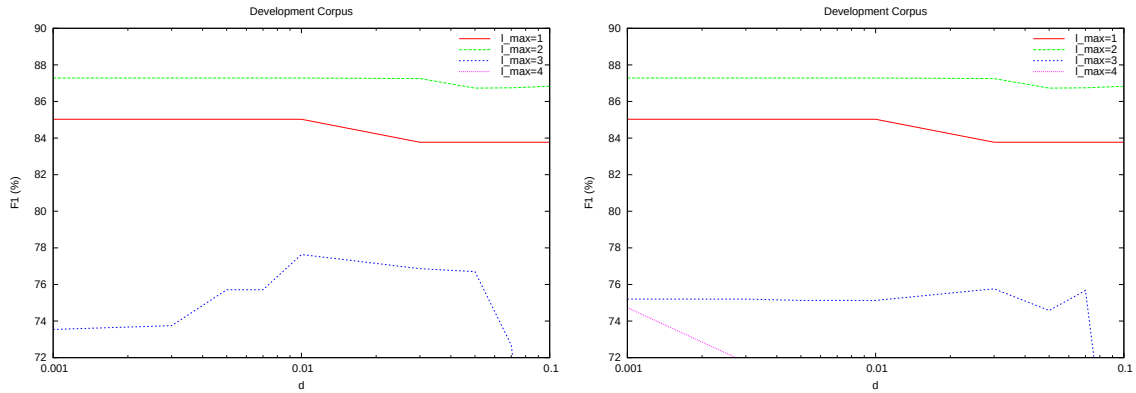

Figure 6.16: F_1 scores obtained when annotating NEs with CSSR, using Jensen-Shannon divergence and short suffixes, with various l_{max} and d. Left hand figure shows the results using *sink-1* method, and the right one shows *sink-2* performance.

In this figure it can be seen that the system is more constant in d than it was in α and β when using χ^2 test. For $l_{max} = 1$ and $l_{max} = 2$, the results obtained are similar to those obtained with χ^2 test, but for longer l_{max} values the results are quite worse than the best results obtained in the previous experiments.

The best results obtained with both sink methods are shown in table 6.5. These results are very similar to those obtained with χ^2 method (for $l_{max} = 1$ and $l_{max} = 2$), and as happened with that test, there is no difference between using *sink-1* or *sink-2* method. For the case of longer maximum lengths, there is a significative difference between the two methods, but the performance in both cases is very worse than using shorter l_{max} or χ^2 test.

l_{max}	d	Development Corpus P	R	F_1	Evaluation Corpus P	R	F_1
***sink-1* method**							
1	10^{-6} – 0.01	83.66	86.44	85.03	85.34	88.00	86.65
2	10^{-6}– 0.01	89.55	85.13	**87.28**	89.63	87.66	**88.63**
3	0.01	77.00	78.28	77.64	78.32	77.26	77.79
4	0.0007	67.36	69.89	68.60	67.67	70.40	69.01
***sink-2* method**							
1	10^{-6}– 0.01	83.66	86.44	85.03	85.34	88.00	86.65
2	10^{-6}– 0.01	89.55	85.13	**87.28**	89.63	87.66	**88.63**
3	0.03	73.92	77.70	75.76	74.87	78.56	76.67
4	0.001	73.60	75.91	74.73	74.57	77.15	75.84

Table 6.5: Best F_1 scores obtained for each l_{max}, using Jensen-Shannon test and short suffixes over the two test corpora.

Sink State

To study the cause of the loose in the performance when using long l_{max} values, the number of times the system falls into the sink state are studied. The behavior of the system for various values of d is very similar, so table 6.6 shows the number of times the system falls into the sink state and the ratio of fixed unseen events just for $d = 0.01$. Both sink methods are represented in this table.

	sink-1			***sink-2***		
l_{max}	**Sink Counts**	**Fixed**	**Ratio**	**Sink Counts**	**Fixed**	**Ratio**
1	0	0	–	0	0	–
2	0	0	–	0	0	–
3	5984	4110	0.69	3941	3268	0.83
4	9779	5643	0.58	4153	3903	0.94

Table 6.6: Number of sink states for each l_{max}, using Jensen-Shannon test and short suffixes.

From this table it can be seen that for length 1 and 2 the system never falls into the sink state, so for that reason the performance is much better than for longer l_{max}, when the system often finds unseen suffixes. On the other hand, it can be seen that the second method for dealing with the sink state solve more times this problem than the first method. For that reason the performance improves for these lengths when using *sink-2* method. Nevertheless this improvement is not sufficient to develop a competitive system with $l_{max} = 3$ or $l_{max} = 4$.

Taking into account that results, and seeing that, as happened with χ^2 test, the main reason for the system falling into the sink state are the states deleted because they are considered transient, we repeated these experiments using all suffixes to determine the transitions that define the recurrent states.

Using Jensen-Shannon Test and Long Suffixes to Determine Recurrent States

We now study the influence of taking into account all suffixes when determining the transient states with the automaton built using Jensen-Shannon divergence.

Number of States

Regarding the automaton size, as happened for χ^2 , the built automata are slightly bigger when all suffixes are considered, since less states are deleted, but the dependence on d and l_{max} is equivalent to figure 6.15.

Annotation Results

If we study the performance of these automata over the development corpus, we obtain F_1 scores shown in figure 6.17. In this case, we only show the results of using *sink-1* method because the figures obtained with *sink-2* are the same.

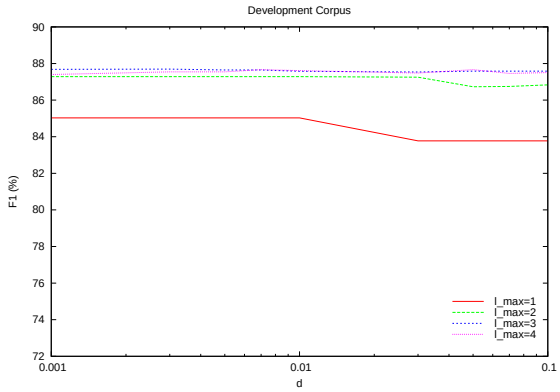

Figure 6.17: F_1 scores obtained when annotating NEs with CSSR, using Jensen-Shannon divergence, long suffixes and *sink-1* method, with various l_{max} and d. The results of using *sink-2* method are exactly the same.

From this figure it is clear that the system performs much better when using all suffixes to determine the recurrent states. Now, the results obtained with $l_{max} = 3$ and $l_{max} = 4$ are comparable, or even slightly better to the results obtained with $l_{max} = 2$.

Table 6.7 shows the best results and the correspondant d values for each maximum length. Again, just the results obtained with *sink-1* method are shown, since they are the same for *sink-2* method.

From this table it can be seen that now the results obtained with long maximum lengths are significantly better than the ones obtained with shorter lengths. Comparing these results with the best results for χ^2 test, it can be seen that for the development corpus they are a little bit worse, but not significantly different with a significance test at a 95% of significance. So the system using Jensen-Shannon divergence is competitive, obtaining also the same results over the evaluation corpus in the best case. Furthermore, this second system is more robust, since it is not so dependent on the parameters, so it can be considered better than the method that uses χ^2 test.

Sink State

In this case, the results obtained with both sink methods are exactly the same. This means that the influence of the sink state is negligible when using all suffixes to determine recurrent

l_{max}	d	Development Corpus P	R	F_1	Evaluation Corpus P	R	F_1
***sink-1* method**							
1	10^{-6}– 0.01	83.66	86.44	85.03	85.34	88.00	86.65
2	10^{-6}– 0.01	89.55	85.13	87.28	89.63	87.66	88.63
3	0.003	89.18	86.26	**87.69**	89.64	88.03	**88.83**
4	0.05	89.29	86.10	87.66	89.24	87.69	88.46

Table 6.7: Best F_1 scores obtained for each l_{max}, using Jensen-Shannon test and long suffixes over the two test corpora with *sink-1* method.

states. Table 6.8 shows the number of times the system falls into the sink state and the fixed times for both methods when using $d = 0.01$. It can be seen from this table that for all maximum lengths the system almost never falls into this special state. For that reason, despite of the fact that for $l_{max} = 4$ the *sink-2* method fixes more sink states, the obtained results over the test corpora are the same for both methods.

	sink-1			***sink-2***		
l_{max}	**Sink Counts**	**Fixed**	**Ratio**	**Sink Counts**	**Fixed**	**Ratio**
1	0	0	–	0	0	–
2	0	0	–	0	0	–
3	2	2	1	2	2	1
4	12	8	0.67	10	10	1

Table 6.8: Number of sink states for each l_{max}, using Jensen-Shannon test and long suffixes.

6.3.5 Comparison with a Markov Model

Another interesting issue to be studied is the performance of the equivalent Markov Model (MM) in the same task with the same data. As we have said, CSSR builds automata that have the same form of an MM but with less states since some suffixes have been grouped together. In fact, if CSSR built an automaton with one state for each possible suffix, this automaton would be the equivalent Markov Model. When we talk of a Markov Model of order n, this means that the considered suffixes have length $n-1$, so the CSSR automaton of $l_{max} = n - 1$ with each suffix in a different state is the n-order Markov Model for the process. Note that we are not talking about Hidden Markov Models, but simple MMs, since CSSR learns automata without any hidden information. As when using CSSR, the unseen events receive a small probability using Lidstone's Law) in order to use them in the annotation step.

In that way, we built this kind of model with the same vocabulary, using the training corpus and tested the obtained automaton over the test corpus. The process is conceptually the same used for CSSR but now the automata do not have the suffixes grouped into states but a state for each suffix of length $n-1$. The experiments where performed with suffixes of length 2, 3 and 4, which are Markov Models of order 3, 4 and 5.

If we apply these different automata to the NER task over both test corpora, the results

shown in table 6.9 are obtained. This table also shows the number of generated states in each case, that is the number of possible suffixes of this length with the current alphabet: $15^{l_{max}}$.

			Development Corpus			Evaluation Corpus		
l_{max}	**MM-order**	**States**	**P**	**R**	F_1	**P**	**R**	F_1
1	bigram	15	83.66	86.44	85.03	85.34	88.00	86.65
2	trigram	255	89.55	85.13	87.28	89.62	87.63	88.62
3	4-gram	4095	89.18	86.23	**87.68**	89.61	88.00	**88.80**
4	5-gram	65535	88.99	85.87	87.40	89.14	87.69	88.41

Table 6.9: Obtained results for each l_{max}, using a Markov Model

The obtained results for the MM over the development corpus are better than the best results obtained with CSSR for each l_{max}, except for $l_{max} = 3$, in which case the difference is not significative. Furthermore, the automata built by CSSR are much smaller than the Markov Model (between 200 and 600 states for $l_{max} = 4$ with CSSR over 60,000 for the MM) so they could be faster to apply. We also tried to build the 6-gram Markov Model with $l_{max} = 5$, that would have 1,048,575 states but such a model could not be loaded into the memory of the PC we used (2Gb of RAM).

At the sight of that, we used CSSR to learn an automaton with $l_{max} = 5$, and in this case it was possible to build it since the number of states is much smaller (about 2,000). Using this automaton to tag NEs in the development and evaluation corpora did not improve the results obtained with $l_{max} = 4$, though the performance was very similar. If $l_{max} = 6$ is used, the automaton has around 4,000 states and the performance starts decreasing significantly with respect to shorter lengths.

6.3.6 Discussion

In last sections, various experiments of using CSSR for NER task have been presented. It has been seen that the main source of error in the CSSR automata is the number of times the system finds an unseen suffix. This depends on how the automaton has been built, since there are some deleted suffixes that may be necessary afterwards. In this line, we have seen that modifying the algorithm in order it studies more suffixes when determining the recurrent states leads to a improvement on the performance.

The problem with the sink state becomes worse when increasing the maximum length, in fact for $l_{max} = 1$ and $l_{max} = 2$ this problem can be ignored, at least for NER task. Nevertheless, we think that it is important to develop a more robust system, not so dependent on the maximum length that can be more useful for future tasks.

In order to ease the discussion of the results, table 6.10 summarizes the best results obtained with each system in NER task. Since the best results are independent from the method used to deal with the sink state, this table shows the results for *sink-1* method. Also, the results obtained with $\beta = 1$ are ignored since they are always behind the results obtained wit best β.

The system that led to the best accuracy, obtained a $F_1 = 87.93\%$ for the development corpus and $F_1 = 88.58\%$ for the evaluation corpus. These results are clearly better than those obtained by FreeLing analyzer (baseline), what means that our system is able to capture more sophisticated patterns than those recognized by the hand-made automaton used by FreeLing.

	χ^2 Test				Jensen Shannon Test				Markov Model	
	Short Suff.		Long Suff.		Short Suff.		Long Suff.			
l_{max}	Devel.	Eval.	Devel.	Eval.	Devel.	Eval.	Devel.	Eval.	Devel.	Eval.
1	85.03	86.65	85.03	86.65	85.03	86.65	85.03	86.65	85.03	86.65
2	87.71	88.74	87.71	88.74	87.28	88.63	87.28	88.63	87.28	88.62
3	87.77	88.71	87.77	88.82	77.64	77.79	87.69	88.83	87.68	88.80
4	87.79	88.69	**87.93**	**88.58**	68.60	69.01	87.66	88.46	87.40	88.41

Table 6.10: Best F_1 scores obtained for each l_{max}, with the different possible configurations and *sink-1* method over the two test corpora.

These results are not significantly different than the results obtained with the best Markov Model ($F_1 = 87.68$ for the development corpus and $F_1 = 88.80$ for the evaluation) but are obtained with a much smaller automaton. This means that CSSR keeps all the properties of Markov Models adding the advantage of building a much smaller automaton, what makes it a faster system to be used.

Our results can be also compared with the winner system of CoNLL-2002 shared task (Carreras et al., 2002). This system was developed with the same training and testing data and performs the NE recognition and classification separately, so it is possible to compare our system with the part that performs the NE recognition.

That system obtained an F_1 of 91.66% for the Spanish development corpus and a 92.91% for the test corpus. These results are higher than the results presented in this work, which was expected since the feature set used by that system is much richer (bag of words, disambiguated PoS tag, many orthographic features, etc.) than the used in our experiments.

Furthermore, it is possible to apply the NEC system used by Carreras et al. (2002) to the output of our NE detector. Doing so over our best results yields to an $F_1 = 75.58\%$, which would situate our system in the sixth position in CoNLL-2002 ranking table for complete NER systems in Spanish as shown in table 6.11 (Tjong Kim Sang, 2002a).

System	Precision	Recall	F_1
Carreras et al. (2002)	81.38	81.40	81.39
Florian (2002)	78.70	79.40	79.05
Cucerzan and Yarowsky (2002)	78.19	76.14	77.15
Wu et al. (2002)	75.85	77.38	76.61
Burger et al. (2002)	74.19	77.44	75.78
Tjong Kim Sang (2002b)	76.00	75.55	75.78
CSSR	**76.19**	**75.00**	**75.58**
Patrick et al. (2002)	74.32	73.52	73.92
Jansche (2002)	74.03	73.76	73.89
Malouf (2002b)	73.93	73.39	73.66
Tsukamoto et al. (2002)	69.04	74.12	71.49
Black and Vasilakopoulos (2002)	68.78	66.24	67.49
McNamee and Mayfield (2002)	56.28	66.51	60.97
Baseline	26.27	56.48	35.86

Table 6.11: Results presented by the participants of the CoNLL-2002 shared task compared to the results obtained by CSSR NER-system.

The obtained results are considered promising, since the used features are very simple and the system is fast and robust. The possibility of using a more complex alphabet encoding more features remain to be studied, but it is important to take into account that when using a bigger alphabet, more data will be necessary to build correct automata.

6.4 Biomedical Named Entity Recognition

Considering the results on using CSSR to detect Named Entities in general domain, some experiments on applying it to detect Biomedical Named Entities in medical text were performed. In this case, the relevant features are different, since the biomedical terms depend on other characteristics, but the task is conceptually equivalent to detecting NEs in a text. As in NE recognition, here just the part of identifying terms is performed, for the classification step more features would be needed. Despite the simplicity of the approach, preliminary results in the identification step are just slightly behind those of comparable systems, which may be considered promising.

This work was done in collaboration with Bill Keller and James Dowdall in the University of Sussex.

6.4.1 Data

To perform the experiments in Biomedical NE identification, GENIA corpus version 3.0 (Kim et al., 2003) is used. This corpus contains 2,000 MEDLINE abstracts and almost 90,000 Biomedical NEs. Table 6.12 shows the statistics of this corpus.

GENIA corpus in numbers
18,545 sentences
436,967 words
89,862 Biomedical NEs
23.6 words/sentence
4.8 NE/sentence
2.3 words/NE

Table 6.12: Statistics of the GENIA 3.0 corpus.

Note that the frequency of NE occurrence is bigger than for NER in general domain though the sentences are smaller. Also the number of words per NE is bigger than for NER in general domain.

6.4.2 Alphabet

To use CSSR for terminology extraction, it is necessary to develop an alphabet that captures the most relevant features of the task. So it is necessary to determine what features are important to determine term structure.

In our approach, one of the features taken into account to create the alphabet is the PoS tag of the word (basically if it is a noun or an adjective). These two PoS tags are specially interesting because they are the PoS of the words that will form the medical terms (in combination with conjunction, articles, ...).

Another crucial feature for this task is to determine if a word is likely to appear inside a term or not. There will be clearly medical words, that will almost always belong to a term, some non-medical words, that will be usually outside any term and words that could be inside or outside a term depending typically on their position respect other words. For that reason, we defined a division in three sets of kinds of words: one for the words that are very likely to be inside a term (words that appear more frequently as a part of a term or in medical domain texts) what we will call *term-like words*, words that are probably outside a term (words that appear more frequently outside any term or in general domain texts), these will be *non-term-like words* and words that can appear both inside or outside a term or in medical and non-medical texts (*ambiguous words*). The way of defining and determining these three sets, will be discussed in next section.

To better capture the pattern of the terms, there are some relevant kinds of words that are mapped into special symbols in the alphabet without taking into account the other features. One of those groups are the *functional words*, containing articles, prepositions and conjunctions which can appear inside terms linking other kind of words. Another one are the words with orthographic peculiarities (alphabetic characters mixed with numbers, punctuation marks inside the word, capitalized and non-capitalized characters mixed or words totally capitalized), which will often be part of a term. Thus, the visible alphabet used is the following:

- **PW:** Orthographically peculiar word.
- **TN:** Term-like noun (without peculiarities).
- **NN:** Non-term-like noun (without peculiarities).
- **GN:** Other noun (without peculiarities).
- **TA:** Term-like adjective (without peculiarities).
- **NA:** Non-term-like adjective (without peculiarities).
- **GA:** Other adjective (without peculiarities).
- **f:** Functional word (prepositions, articles, conjunctions)
- **w:** Other.

Note that, as in NER task, these classes are disjoint, so each word belongs just to one of them, having only one possible alphabet symbol.

Going back to the example used in section 2.3.2, the words of that sentence will be mapped into the visible and complete alphabet as shown in figure 6.18. Note that the considered *term-like*, *non-term-like* or *ambiguous* words may vary depending on how these groups are determined and which thresholds are used.

Term-like and Non-term-like Words

In order to determine if a non-peculiar noun or adjective is to be considered *term-like*, *non-term-like* or *ambiguous*, we developed two different ways to measure how likely is a word to be medical, so probably a term.

Word	Visible Symbol	Complete Symbol
IL-2	PW	PW_B
gene	GN	TN_I
expression	GN	GN_I
and	f	f_O
NF-kappa	PW	PW_B
B	TA	TA_I
activation	GN	GN_I
through	w	w_O
CD28	PW	PW_B
requiresreactive	TA	TA_O
oxygen	NN	NN_O
production	NN	NN_O
by	f	f_O
5-lipoxygenase	PW	PW_B
.	w	w_O

Figure 6.18: Example of a training sentence and its translation to visible and complete alphabets.

The first measure was built comparing the probability of seeing a determined word in a medical corpus, M (in our approach, the GENIA corpus[3] (Kim et al., 2003)) with the probability of seeing this word in a general domain corpus G (BNC[4]). Computing this ratio is equivalent to compare the counts of this word in each corpora and ponderate this ratio with the size of the corpora:

$$R = \frac{P(w|M)}{P(w|G)} = \frac{counts(w, M)}{counts(w, G)} \frac{size(G)}{size(M)}$$

The second measure implemented to determine if a word is *term-like* or *non-term-like*, took into account just the information available in the training corpus. In this case, the ratio was computed counting how many times each word has been seen inside a term in the training corpus over how many times it has been seen outside a term:

$$R = \frac{counts(w, term)}{counts(w, nonterm)}$$

In both cases if the ratio R is much greater than 1, this will probably be a *term-like* word, since it means that the word appears more often in a medical domain than in a general domain, or more often inside a term than outside. If R is much smaller than 1 this means that this word appears more frequently in a general domain than in a medical domain or that it appears more frequently outside a term in the training corpus. In this case, it will be a *non-term-like* word. Finally, if the ratio is close to 1, the word appears with a similar probability in both domains, or outside/inside a term, so there is not enough evidence to classify this word so it will be in an *ambiguous* word. The words in the test corpus that

[3] http://www-tsujii.is.s.u-tokyo.ac.jp/~genia/
[4] http://www.natcorp.ox.ac.uk/

have not been seen in the training corpus are also mapped into this last set since there is no evidence to consider them *term-like* or *non-term-like*.

To perform the classification in these three sets, two thresholds for R are defined: one over which a word will be classified as *term-like word* and another one under which the word will be a *non-term-like word*. If R is between these two thresholds, the word will be classified as *ambiguous*. Lets say sup is the superior threshold for R and inf is the inferior limit, then word classification will be:

$$\begin{array}{lcl} \text{if } R < inf & \rightarrow & \text{non-term-like word} \\ \text{if } inf < R < sup & \rightarrow & \text{ambiguous word} \\ \text{if } sup < R & \rightarrow & \text{term-like word} \end{array}$$

The performance of the system is expected to depend strongly on these thresholds that determine the three areas used to classify the words. These thresholds are empirically determined performing the experiments with many different threshold values. Studying the performance of the system, it was seen that the second measure led to better results than the first one. For that reason, the results presented in this work are those obtained with this second measure.

6.4.3 Experiments and Results

Several experiments to study the influence of the various parameters and to seek for the best configuration were performed. The experiments are conducted with 10-fold cross-validation with the GENIA 3.0 corpus.

As explained in section 6.4.2, two of the main parameters for this task will be the thresholds to determine if a word is *term-like* or not. The other important parameters are the parameters of the algorithm: the maximum length of the histories taken into account (l_{max}) and the significance level of the hypothesis test. If χ^2 test is used to perform this hypothesis test, also the parameter β has to be tuned.

The experiments were originally performed using χ^2 test and using only short suffixes to determine the transient states. In this first experiments, β parameter was set to 1, since this is the configuration of the original implementation of the algorithm. Secondly we repeated the experiments with both χ^2 test and Jensen Shannon divergence when using all suffixes to determine recurrent states. In this case, when using χ^2 test, β parameter was also tuned. We did not perform the experiments with Jensen Shannon divergence, nor the tuning of β for χ^2 test when using short suffixes to determine recurrent states because we have already seen for NER task that best results are obtained using all suffixes. All the experiments are performed using *sink-1* method.

The best values for precision, recall, and F_1 obtained for various maximum lengths and the values of the parameters that led to these results are presented in the following tables. Table 6.13 shows the results when using χ^2 test and short suffixes to determine recurrent states. In this case, β is set to 1.

In table 6.14 the results obtained with χ^2 test and using all suffixes to determine recurrent states are shown. This table presents both the results obtained using $\beta = 1$ and with the best β.

Finally, table 6.15 shows the best obtained results when using Jensen Shannon divergence. In this case, also all suffixes are used to determine recurrent states.

l_{max}	Parameters				Results		
	α	β	inf	sup	P	R	F_1
With $\beta = 1$							
1	0.1	1	0.8	9.8	71.81	73.34	72.58
2	0.1	1	1.1	6.1	71.19	75.23	**73.15**
3	0.1	1	4.1	4.7	69.54	72.91	71.18
4	0.1	1	0.04	0.06	50.64	68.38	58.19

Table 6.13: Best results obtained for each l_{max} using χ^2 test and short suffixes to determine recurrent states with $\beta = 1$

l_{max}	Parameters				Results		
	α	β	inf	sup	P	R	F_1
With $\beta = 1$							
1	0.1	1	0.8	6.8	71.83	73.44	72.63
2	0.1	1	0.7	6.0	71.25	75.31	**73.22**
3	0.05	1	0.6	6.0	69.78	72.95	71.33
4	0.1	1	0.9	6.8	69.06	71.06	70.05
With best β							
1	0.1	100	0.8	6.5	71.84	73.47	72.65
2	0.1	100	0.6	6.0	71.29	75.30	**73.24**
3	0.1	100	0.6	6.0	70.71	73.14	71.90
4	0.1	100	0.8	6.8	69.58	70.76	70.17

Table 6.14: Best results obtained for each l_{max} using χ^2 test and all suffixes to determine recurrent states

6.4.4 Discussion

From all the various system configurations that have been tested, the best results are obtained when using χ^2 test and all suffixes to determine the recurrent states. Nevertheless, the improvement introduced by using all suffixes and tuning β parameter is not significative for the best system (obtained with $l_{max} = 2$). As for NER task, the introduction of more suffixes to study the recurrent states was specially useful to develop better automata with large maximum lengths but in this case the results obtained can not compete with those attained with $l_{max} = 2$.

Our results can be compared directly to (Lin et al., 2004) and (Lee et al., 2004), which also consider term recognition over GENIA 3.0 using 10-fold cross-validation. The system of Lin et al. (2004) obtains $F1 = 57.4\%$ using Maximum Entropy Models, but improves to $F1 = 76.9\%$ using post-processing techniques to expand the limits of NEs depending on the boundary words. The system of (Lee et al., 2004) reports $F1 = 79.2\%$ using SVMs and $F1 = 79.9\%$ when also applying post-processing techniques to extend the boundaries of NEs based in dictionaries.

Our best system obtains $F1 = 73.24\%$ without using any post-processing techniques,

l_{max}	Parameters			Results		
	d	inf	sup	P	R	F_1
With $\beta = 1$						
1	0.05	1.3	6.4	71.8	73.42	72.60
2	0.1	1.0	6.0	71.22	75.29	**73.20**
3	0.1	1.0	5.9	71.46	72.15	71.80
4	0.05	1.1	6.8	69.06	71.06	70.05

Table 6.15: Best results obtained for each l_{max} using Jensen Shannon divergence and all suffixes to determine recurrent states

and with few features taken into account, which may be considered competitive to state of the art systems.

Detecting exactly the boundaries of NEs is a very difficult issue in this task, being an important error source. For that reason we think that using a post-processing technique to expand term boundaries if necessary could improve the obtained results significantly, as in (Lin et al., 2004) system.

6.5 Text Chunking

In this section, the performed experiments in detecting various syntactic chunks in a text and the obtained results are presented.

6.5.1 Data

For these experiments, the data for the CoNLL-2000 shared task (Tjong Kim Sang & Buchholz, 2000) are used. These data contain three corpora: one for the train, one for the development of the system and one for the evaluation. There are eleven different chunk types. The sizes of these three corpora are presented in table 6.16. Table 6.17 shows the number of chunks of each kind in the training data.

Corpus	Number of words	Number of chunks
Train	211,794	192,762
Development	40,053	36,087
Evaluation	47,406	43,210

Table 6.16: Number of words and chunks in each corpus.

6.5.2 Alphabet

The alphabet used to train the automata are the Part of Speech (PoS) tags available in the CoNLL-2000 corpora. The total number of different tags is 44, what means that CSSR alphabet will have 132 symbols (each PoS tag combined with the B, I or O tag). In these experiments, the same vocabulary is used for all the chunk types.

Chunk Type	Counts	Proportion
Noun Phrase (NP)	55,081	51%
Verb Phrase (VP)	21,467	20%
Prepositional Phrase (PP)	21,281	20%
Adverb Phrase (ADVP)	4,227	4%
Subordinated Clause (SBAR)	2,207	2%
Adjective Phrase (ADJP)	2,060	2%
Particles (PRT)	556	1%
Conjunction Phrase (CONJP)	56	0%
Interjection (INTJ)	31	0%
List Maker (LST)	10	0%
Unlike Coordinated Phrase (UCP)	2	0%

Table 6.17: Number of occurrences of each kind of chunk in the training corpus.

6.5.3 Experiments and Results

For chunking task, we will focus on the use of CSSR with χ^2 hypothesis test and all suffixes to determine the recurrent states. We also performed the experiments using only short suffixes, but the results were much worse (Padró & Padró, 2005b) and, as for NER, we found troubles when some necessary suffixes were deleted from the automaton when determining recurrent states. Since when using all suffixes to determine the recurrent states, the method used to manage the transitions into the sink state is not very influent, in these experiments we will use *sink-1* method (see section 6.1.3). The experiments with Jensen-Shannon divergence have been also performed and are summarized at the end of this section.

As for NER, we performed several experiments with various values of the parameters influencing the χ^2 hypothesis test: significance level α, maximum length l_{max} and extra data weight β. This experiments are performed over the development corpus and the best obtained systems are tested over the evaluation corpus.

To perform the annotation of the various kind of chunks, an automaton is learned using CSSR for each phrase type. Each automata is expected to reproduce the pattern of each chunk kind, and following the same approach used in NER, we can use it to tag new text. In this way, each automaton is an individual chunker that detects a different kind of chunk in a text. To build a complete chunker, it is necessary to look for the best way to apply and combine all the automata over the development corpus in order to obtain the best performance. In this work we will present two simple approaches to combine the various automata.

Before combining the various automata, we are interested in studying the performance of each chunker separately, paying a special attention to the behavior of the system when varying the parameters. For that reason, in next section we will present in detail the results obtained in NP detection, with the study of the variability on β. Since the dependency on the parameters is similar for all the automata, just the best results of the other chunkers will be summarized in that section.

Independent Chunkers

As we have said, various values of α, β and l_{max} were tested over the development corpus to build the best possible automata for each kind of chunk. First of all we will focus on the study of the results for NP chunker.

Figure 6.19 shows the variation on β of the F_1 score with various maximum lengths. Since the behavior when varying α is similar we will present the results with α set to 0.1.

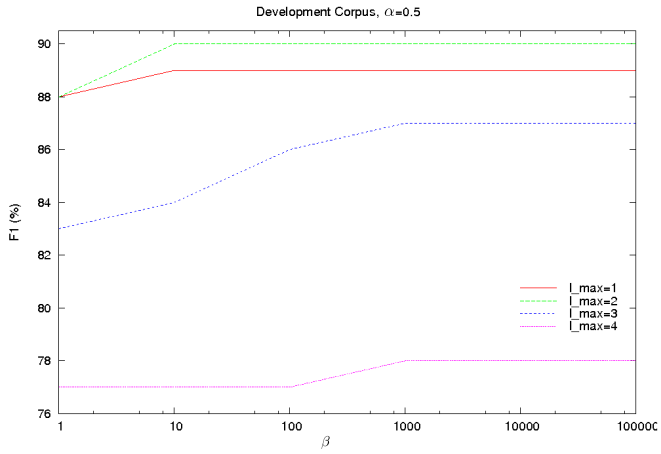

Figure 6.19: F_1 score obtained when tagging NPs with CSSR, using χ^2 test, long suffixes and *sink-1* method, with various l_{max} and β values and with $\alpha = 0.5$.

From this figure it can be seen that introducing the parameter β improves significantly the performance of the system, specially for higher maximum lengths. The maximum is normally obtained for β around 10,000. Nevertheless, only for $l_{max} = 2$ introducing this parameter makes a better system that the best system for $\beta = 1$, which is obtained with $l_{max} = 1$. For longer maximum lengths, the system is not able to learn a good automaton because for this task the alphabet is much bigger than for NER, so much more data would be necessary to learn good automata with maximum length 3 or 4. Nevertheless, introducing the parameter β leads to an important improvement of the performance of the automata with higher maximum lengths.

The best scores obtained with the various maximum lengths when varying β, compared with the values when β is set to 1, and the correspondent α values are shown in table 6.18.

Regarding the other kind of chunks, the behavior on β is similar, obtaining best results if this parameter is used. Table 6.19 summarizes the best obtained results over both the development and evaluation corpora with each separate chunker. First part of the table shows the results for $\beta = 1$, and the second one for best β.

From this table it can be seen that tuning the β parameter improves the performance of all chunkers, though this improvement is only significative (at a 95% confidence degree) for the case of NP, SBAR and ADJP chunkers. In general, introducing this parameter makes the system less dependent on α as happened for NER.

Complete Chunking

In this section, we present the experiments performed and the obtained results when annotating the various kind of chunks in a sentence. In our work only the seven more frequent

l_{max}	**Best Parameters**		**Development Corpus**			**Evaluation Corpus**		
	α	β	P	R	F_1	P	R	F_1
With $\beta = 1$								
1	0.3	1	89.47	88.75	**89.11**	89.09	88.56	**88.83**
2	0.5	1	88.37	89.09	88.73	87.99	89.19	88.59
3	0.7	1	83.23	84.91	84.06	83.90	85.23	84.56
4	0.7	1	75.55	78.98	77.23	75.97	78.68	77.30
With best β								
1	0.001 – 0.7	$100 - 10^6$	90.45	89.42	89.93	89,84	88.58	89.20
2	0.001 – 0.7	10,000	90.48	90.25	**90.37**	91.27	91.05	**91.16**
3	0.001 – 0.1	10,000	87.25	87.85	87.54	88.19	88.46	88.33
4	0.1 – 0.7	10,000	77.34	80.05	78.67	77.39	80.12	78.73

Table 6.18: Best F_1 scores obtained with NP chunker, for each l_{max}, using χ^2 test, long suffixes and *sink-1* method over the two test corpora.

chunks are studied, but introducing the other chunk types would not make any conceptual difference.

As we have said, in this work we implemented two simple approaches to combine the various automata. The first one combines directly the predictions of each independent chunker. The second one combines the chunkers sequentially, taking into account the presence of other chunks in both the learning and the annotating steps. These methods and the results obtained with them are presented in next sections.

Combining Independent Chunkers

The first approach consists of tagging the corpus separately with each automaton, independently of the predictions of the other automata. Then, the chunk tags deduced by the various automata are combined by a simple priority system: when a word is tagged as part of a chunk by two different automata (i.e. this word is recognized as part of two different chunk types), the system always gives priority to the prediction of the chunker that obtained best individual performance and, therefore, for the words that belong to the same chunk according to the prediction of the chosen automaton.

To perform these combination of the various chunkers, the systems that obtained best results over the development corpus (presented in table 6.19) are used. This combination leads to the results presented in table 6.20. In this table, the precision, recall and F_1 are presented for both the development and evaluation corpus, when using $\beta = 1$ or the best β value. The results for each kind of chunk are those corresponding to the combined systems, not to the individual chunker. In this table, the chunks are in the order in which they are applied, this is, the first one is the best individual chunker, so its prediction will win over the others in case of conflict.

Although these results are not very good, they are in the range of the lower systems presented in CoNLL-2000 shared task, and our system takes into account very few information. Note that in this case, introducing the parameter β leads to an important improvement in the overall results.

Chunk Type	Best Parameters l_{max}	α	β	Development Corpus P	R	F_1	Evaluation Corpus P	R	F_1
With $\beta = 1$									
VP	1	0.001 – 0.03	1	90.59	90.56	90.57	91.74	91.56	91.65
PP	1	0.03	1	87.03	93.81	90.30	87.85	95.16	91.36
NP	1	0.3	1	89.55	88.86	89.20	89.19	88.70	88.94
ADVP	1	0.001	1	69.46	70.88	70.16	77.63	67.32	72.11
ADJP	1	0.005	1	67.95	55.73	61.23	66.58	59.13	62.64
SBAR	2	0.7	1	63.98	29.31	40.20	72.32	23.93	35.96
PRT	1	0.07	1	67.24	23.78	35.14	52.38	20.75	29.73
With best β									
VP	1	0.001 – 0.7	100 – 10^6	91.26	91.09	91.18	91.80	91.84	91.82
PP	1	0.001 – 0.7	100 – 10^6	87.10	93.84	90.34	87.94	95.20	91.43
NP	2	0.001 – 0.7	1000 – 10^6	90.68	90.47	90.57	91.45	91.24	91.34
ADVP	1	0.001 – 0.7	100 – 10^6	69.44	72.54	70.96	77.68	67.90	72.46
ADJP	1	0.001 – 0.7	100	67.28	57.75	62.15	66.42	62.33	64.31
SBAR	2	0.7	100	63.02	29.80	40.47	67.79	26.36	37.95
PRT	1	0.001 – 0.7	100 – 10^6	65.62	25.61	36.84	51.16	20.75	29.53

Table 6.19: Best results obtained separately by each chunker for both test corpora with $\beta = 1$ and best β.

After using this simple way to combine various chunkers, we developed a second system a little bit more informed that takes into account the predictions of previous chunk types to perform the tagging task.

Combining Chunkers Sequentially

In this section, the second approach to combine the various chunkers is presented. In this case, the automata were not built and applied to the task separately, as if they were independent, but applied sequentially, each one taking into account the information produced by the previous chunkers. To do so, it is necessary to design previously the architecture of the whole system, deciding in which order the chunkers will be applied. Once it is decided, all the chunkers must be trained again in order to learn automata that take into account the output of the previous steps. So each chunker is built via CSSR, with a vocabulary that includes the same alphabet introduced above plus the information about previously tagged chunks. That means that, for example, if the chunker NP is applied after VP and PP, the existent VP and PP chunks will be tagged with a special symbol in the vocabulary. CSSR will learn that the words having this special symbol will not be ever considered part of the currently studied chunk, since they are already considered part of another chunk type.

The chosen order to apply this method is, as when combining independent chunkers, the order of better performance when the systems are applied separately. This is: VP, PP, NP, ADVP, ADJP, SBAR, PRT. In this case, since the automata are learned with a different vocabulary, the search for the best parameters has to be performed again.

Table 6.21 shows the best obtained results when combining the various chunkers in that order, and the parameters that lead to this performance. Again, we present the results

Chunk Type	**Development Corpus**			**Evaluation Corpus**		
	P	R	F_1	P	R	F_1
With $\beta = 1$						
VP	90.59	90.56	90.57	91.74	91.56	91.65
PP	87.17	93.73	90.33	88.06	95.05	91.42
NP	89.82	88.71	89.26	89.45	88.55	89.00
ADVP	70.19	70.37	70.28	78.69	66.51	72.09
ADJP	69.88	50.56	58.67	70.93	55.71	62.40
SBAR	75.19	24.63	37.11	79.05	21.87	34.26
PRT	68.42	23.78	35.29	56.76	19.81	29.37
Overall	88.17	86.58	**87.37**	88.87	87.19	**88.02**
With best β						
VP	91.26	91.09	91.18	91.80	91.84	91.82
PP	87.22	93.76	90.37	88.12	95.12	91.48
NP	91.32	89.90	90.61	92.02	90.66	91.34
ADVP	71.03	70.75	70.89	79.42	66.86	72.60
ADJP	72.29	51.01	59.82	73.87	56.16	63.81
SBAR	73.38	25.12	37.43	76.77	22.24	34.49
PRT	70.00	25.61	37.50	55.26	19.81	29.17
Overall	89.16	87.36	**88.25**	90.32	88.42	**89.36**

Table 6.20: Results obtained combining the predictions of the various kind of chunkers choosing always the best chunker.

when using $\beta = 1$ and the results with best β.

In this case, F_1 grows respect our first approach, though this improvement is not significant if we compare the two systems with $\beta = 1$ and with best β value. What really improves the performance of the system is the tuning of β parameter.

6.5.4 Discussion

Table 6.22 shows the results obtained by the various systems in the CoNLL-2000 shared task and the position that the presented CSSR approaches would have had. The first approach to combine the individual chunkers is labelled "CSSR-1" and the second one, that combines chunkers sequentially "CSSR-2". In both cases, the system with $\beta = 1$ and with best β are presented. The systems marked with an asterisk are systems that did not participate in CoNLL-2000 shared task but reported this results later using the same corpora, so are also comparable with CoNLL-2000 systems and with our results.

Note that, though tuning β parameter improves significantly the performance of the system, the position that we would have obtained in CoNLL-2000 shared task, would have been the same for all cases. This position is in the lower positions, so the obtained results with CSSR in Chunking task are not as good as those obtained for NER, comparatively with other state-of-the-art systems. Nevertheless, the system uses very few information and is very fast, so being competitive with other systems is not a bad result.

The comparison with (Thollard & Clark, 2002) is specially interesting for us, since this

Chunk Type	Best Parameters l_{max}	α	β	Development Corpus P	R	F_1	Evaluation Corpus P	R	F_1
With $\beta = 1$									
VP	1	0.01	1	91.59	90.56	90.57	91.74	91.56	91.65
PP	1	0.3	1	87.55	93.87	90.60	85.99	94.91	90.23
NP	2	0.3	1	88.98	89.03	89.00	90.58	90.05	90.32
ADVP	2	0.3	1	72.33	72.75	72.54	74.88	69.52	72.10
ADJP	2	0.3	1	63.72	60.96	62.31	61.54	65.75	63.58
SBAR	1	0.3	1	48.35	24.67	32.67	55.06	25.42	34.78
PRT	1	0.3	1	40.54	28.30	33.33	33.33	29.25	31.16
Overall				87.54	87.61	**87.58**	88.03	88.37	**88.20**
With best β									
VP	1	0.3	1000	91.26	91.09	91.18	91.80	91.84	91.82
PP	2	0.3	100	87.60	93.71	90.49	88.72	95.06	91.76
NP	2	0.3	100	91.55	89.94	90.75	91.70	91.71	91.70
ADVP	1	0.3	100	73.45	73.13	73.29	74.59	77.26	75.90
ADJP	2	0.3	100	64.23	62.45	63.33	64.05	60.08	62.00
SBAR	2	0.3	100	49.23	23.02	31.37	64.25	21.50	32.21
PRT	2	0.3	100	31.58	21.82	25.81	36.26	31.13	31.50
Overall				88.01	87.88	**88.40**	89.58	89.49	**89.54**

Table 6.21: Results obtained combining the various chunkers sequentially.

system uses Probabilistic Suffix Trees, that have some common properties with CSSR. The obtained results with that system are similar to the results obtained with CSSR but it uses some more information, such as lexical information. This opens the idea that introducing this kind of information in the same way into CSSR, the results could be improved, since Thollard and Clark (2002) report a significant improvement when using this information.

All the experiments presented in this section were repeated using Jensen Shannon divergence to build the causal states. In this case, also all suffixes are used to determine recurrent states. In general, the obtained results are very similar or a little bit behind those obtained with χ^2 test when using the best β, though the difference in general is not significative. Thus, Jensen Shannon divergence has proved to be also a good method to determine whether two histories belong to the same causal state for Chunking task and taking into account that there is one less parameter to tune to obtain this results, it is worth considering its use when using CSSR.

6.6 Conclusions of this Chapter

In this chapter, the experiments and results obtained when using CSSR for various NLP annotation tasks have been presented. Different parametrization and different implementations of the system have been tested, studying the effects of the parameters. For NER in general domains and in biomedical texts, a system competitive with other state-of-the-art systems has been developed, with the advantage that the approach is very simple and the

System	Precision	Recall	F_1
Zhang et al. (2001)*	94.29	94.01	94.13
Kudo and Matsumoto (2001)*	93.89	93.92	93.91
Carreras and Màrquez (2003)*	94.19	93.29	93.74
Kudo and Matsumoto (2000)	93.45	93.51	93.48
van Halteren (2000)	93.13	93.51	93.32
Tjong Kim Sang (2000)	94.04	91.00	92.50
Zhou et al. (2000)	91.99	92.25	92.12
Déjean (2000)	91.87	92.31	92.09
Koeling (2000)	92.08	91.86	91.97
Osborne (2000)	91.65	92.23	91.94
Veenstra and van den Bosch (2000)	91.05	92.03	91.54
Pla et al. (2000)	90.63	89.65	90.14
CSSR-2 (Best β)	**89.58**	**89.49**	**89.54**
CSSR-1 (Best β)	**90.32**	**88.42**	**89.36**
Thollard and Clark (2002)*	*not reported*		89.00
CSSR-2 ($\beta = 1$)	**88.03**	**88.37**	**88.20**
CSSR-1 ($\beta = 1$)	**88.87**	**87.19**	**88.02**
Johansson (2000)	86.24	88.25	87.23
Vilain and Day (2000)	88.82	82.91	85.76
Baseline	72.58	82.14	77.07

Table 6.22: Results presented by the participants of the CoNLL-2000 shared task compared to the results obtained by both CSSR Chunking systems.

annotation step fast. For Chunking task, the results are comparable with last systems in CoNLL-2000 competition, so to develop a competitive system, more research in this line must be done.

One interesting contribution of this chapter is the modification of CSSR algorithm to take into account all suffixes when determining recurrent states. This leads to a loose on the generalization power of the algorithm, but it has been experimentally proven to be useful. The results obtained with this alternative implementation are much less dependent on the parameters so good systems can be more easily developed.

In general, the results obtained in these experiments are considered promising, since the approach used is very simple and the results are comparable to state-of-the-art systems. Nevertheless, to improve the performance of the system it would be necessary to introduce more information into the system, what must be done introducing it into the alphabet. Then, the alphabet size would increase and 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 Chunking, where the alphabet is big and lots of data have to be processed.

For that reason, we are interested in looking for some other ways of introducing more information into the system without increasing the size of the alphabet as not enough data would be available to build automata with statistical reliability. In this way, our proposal is to introduce all the features not encoded in the alphabet via conditional models. So, in the following chapters we propose and test two different ways of combining CSSR and Maximum Entropy Models.

CHAPTER 7

Extending CSSR with ME models

In previous chapters, it has been seen that using CSSR for NLP tasks leads to reasonably good results, specially taking into account the simplicity of the considered features. The problem is that the proposed approach is rather limited, since all information we want to take into account has to be encoded in the alphabet, so if we want to introduce more sophisticated features into the system to try to improve the performance, the alphabet size will grow. But since the amount of necessary data to build a correct automaton grows exponentially with the alphabet size, it is important not to increase the alphabet size beyond the limit established by the available data.

For that reason, the improvement of the performance of the system is tied to introducing more information into the system without increasing the alphabet size. In this line, we propose to combine CSSR algorithm with conditional models that can take into account more complicated information. Thus, the generative model of CSSR would be converted into a conditional model. Two examples of conditional models that could be used in this way are Conditional Random Fields (Lafferty et al., 2001; Wallach, 2004) or Maximum Entropy (ME) Models (Berger et al., 1996; Ratnaparkhi, 1997).

Convining conditional models with CSSR we expect to take advantage of the power of conditional models, for example the ability to take many features into account and also to use the good properties of CSSR such as it builds markovian automata that are minimal, so we can encode the same information than a Markov Model encodes but with much less memory.

We focus on the use of ME models with CSSR because McCallum et al. (2000) already approached a way of combining these models with Hidden Markov Models, named Maximum Entropy Markov Models (MEMM) obtaining good results in the field of Information Extraction. Since CSSR builds automata similar to Markov Models, we think that it is interesting to try to combine it with ME models in a similar way than MEMMs do. Nevertheless, our proposal can not be directly compared to MEMMs since they use Hidden Markov Models and our approach uses CSSR automata that do not directly include hidden information.

This chapter first of all introduces the Maximum Entropy Models framework (section

7.1) and then presents, in section 7.2, the proposed approaches to combine ME models and CSSR. We will present two different systems and a baseline that will be tested in chapter 8 in NER task.

7.1 Maximum Entropy Models

Maximum Entropy (ME) models (Berger et al., 1996; Ratnaparkhi, 1997) offer a way to estimate the probability of a determined linguistic class given a linguistic context. The intuitive idea behind these models is to look for the most uniform probability distribution that satisfies all the constraints (context). This is equivalent to say that the model fulfills the constraints but makes the fewest possible assumptions. This uniformity of the distribution can be measured with the entropy, the higher the entropy, the more uniform the distribution. Thus, the Maximum Entropy models are constructed in order that the computed probability maximizes the entropy of the model, while keeping it consistent with the observation.

Consider A the set of all possible classes and B the set of all possible contexts. Then we can define $a \in A$, $b \in B$ and the probability of a in context b will be $p(a|b)$. When these probabilities are estimated from data, we denote $\tilde{p}(a, b)$ the estimated probability for the pair (a, b).

7.1.1 Representing Context and Constraints

To represent the evidence, this is, the context of an observation, all relevant information is encoded into features, that take into account the useful information of the system. For each task, various features will be relevant. Each feature is a binary function of the current class and the context $f(a, b)$ that is set to one when certain values of a and b occur and is zero in any other case. When $f(a, b) = 1$, we will say that the feature is *active*.

Thus, all the relevant information useful to model the system must be translated into feature functions creating a feature set f_i. To ensure that the model takes into account this information, the expected value $\tilde{p}(f)$ that the model assigns to the corresponding feature function f is constrained with the f value for each (a, b) pair:

$$\tilde{p}(f) = \sum_{a \in A,\, b \in B} \tilde{p}(a, b) f(a, b) = \sum_{a \in A,\, b \in B} \tilde{p}(b) p(a|b) f(a, b)$$

where $\tilde{p}(b)$ is the empirical distribution of b in the training sample. Then, the constraint our model has to fulfill is that its estimated probability $\tilde{p}(f)$ is equal to the real probability $p(f)$. There will be various models $\tilde{p}(a|b)$ consistent with these constraints, but following the *Principle of Maximum Entropy* (Jaynes, 1957; Good, 1963) we look for the model $p^*(a|b)$ that maximizes the entropy $H(p)$:

$$H(p) \equiv - \sum_{a \in A,\, b \in B} \tilde{p}(b) p(a|b) \log p(a|b)$$

remaining consistent with the evidence. Thus the selected model will be

$$p^* = \text{argmax}_p H(p)$$

7.1.2 Parametric Form

Given a real corpus and a set of features, the task of determining which is the model that satisfies the constraints and maximizes the entropy is not trivial. The most usual approach is to use the method of Lagrange multipliers to estimate this best model. Here we will summarize the basic ideas of this technique, for a more detailed explanation see (Berger et al., 1996; Ratnaparkhi, 1997; Della Pietra et al., 1997).

The basic idea is that each feature f_i has a Lagrange multiplier λ_i associated called *weight*. Then we can define a Lagrangian function $\Lambda(p, \lambda)$ that must be maximized to maximize the entropy. Then, it can be demonstrated that the model that maximizes the entropy is a function of the features and its weights:

$$p_\lambda(a|b) = \frac{1}{Z_\lambda(b)} \exp\left(\sum_i \lambda_i f_i(a, b)\right)$$

where $Z_\lambda(b)$ is a normalization constant that mantains the probability distribution normalized and is computed as

$$Z_\lambda(b) = \sum_a \exp\left(\sum_i \lambda_i f_i(a, b)\right)$$

This way of computing $p(a|b)$ is called *parametric form* and is used to compute probabilities once the various weights λ for each feature and each class are estimated. There is a variety of algorithms to estimate λ values such as Generalized Iterative Scaling (Darroch & Ratcliff, 1972), Improved Iterative Scaling (Della Pietra et al., 1997) or Limited Memory BFGS (Daumé III, 2004). This last algorithm is the one that will be used in this work. For a comparison of some of these algorithms, see (Malouf, 2002a).

Given a data set, what these algorithms do is to estimate the weights of each feature for each class that maximize the entropy fulfilling the constraints. Once these weights have been determined, they can be used to estimate the probability of any class in any context which is, in fact, what is needed to process new examples.

7.2 Introducing ME models into CSSR

As we have said, our main goal is to introduce into CSSR information that is not encoded in the alphabet. In this way, the algorithm should be able to build more informed automaton without suffering the problems of increasing the alphabet size while having a limited amount of data.

As discussed before, there are various conditional models that would be suitable for the proposed approach, but in this work we focus on the use of Maximum Entropy models to compute the probability distribution of the future. The classes of ME models are the symbols of the complete alphabet used with CSSR, and they define the possible transitions of each state in the automaton. The relevant information associated to each word is encoded in a variety of features, and ME models are used to compute the probability distribution of the next symbol given the current active features.

For example, considering the alphabet used for NER in section 6.3, with CSSR the probability to be computed had the form of $P(M_B|S_B a_I w_O)$, but now the probability will

be computed as $P(M_B|h)$ where h is a history including the symbols and relevant features of last words.

7.2.1 Methodology

In order to study the performance of ME models in the approached NLP tasks and of their combination with CSSR, we present two different approaches to combine these two algorithms: ME-over-CSSR and ME-CSSR. Also, a baseline method (Plain-ME) will be used, that uses ME models but does not combine them with CSSR.

All these methods have in common that they need an ME model learned taking into account the relevant features of the task. Also, all of them will use the symbols of the complete alphabet as the classes of the model.

To build ME models, it is necessary, first of all, to define the feature set relevant for each concrete task. Once it has been done, the features for each word in each corpora and the corresponding class (which corresponds to a symbol of the complete alphabet) are extracted. In our work, we do that using *libfries-0.93*, which is part of FreeLing analyzer (Carreras et al., 2004; Atserias et al., 2006)[1] and based on (Cumby & Roth, 2003). Once the corpora are encoded in that way, the training corpus is used to learn the Maximum Entropy model, this is, the weights for each feature and each class. In our case, this is performed with *MEGA Model Optimization Package* (Daumé III, 2004)[2].

In next sections we describe the two approaches used in this work to combine ME models and CSSR, as well as the baseline method.

7.2.2 Plain ME

This first approach will be used as a baseline of ME models performance since, in fact, it does not use CSSR in any way.

The idea of this first method it to use the learned ME models to compute the probability of each word in test corpus of having each possible class. It has to be taken into account that there is a known part of the symbol (e.g. we know if it is G, M, w, etc.), so the possible classes will be three, one for each possible B, I, O tag. For example, if we know that the word has the visible symbol M, the probability of seeing this word in this context with the complet symbol M_B, M_I and M_O will be computed using the ME model.

Then, with these probabilities computed, the most likely sequence of tags is computed using the Viterbi algorithm, following a similar approach to the one used by Ratnaparkhi (1996).

7.2.3 ME-over-CSSR

This is a first proposal to combine CSSR and ME models without changing the algorithm itself. This method is fast and simple, and we use it to see if introducing more information into the system with ME models leads to better results than using both techniques separately.

[1] FreeLing: Software and information available at http://www.lsi.upc.edu/~nlp/freeling

[2] MEGAM: Software and information available at http://www.cs.utah.edu/~hal/megam

Building the Automaton

The method consists of using CSSR to learn an automaton as in chapter 5, using a simple alphabet that includes the hidden information. So in the automaton building step, ME models are not used. The model is used only during the annotation task, to refine the transition probabilities.

Annotating New Text

To annotate new sentences with ME-over-CSSR, a Viterbi algorithm using CSSR learned automaton is used, as explained in section 5.3. But in this case, the transition probability for a word with a symbol a is computed as a linear combination of the transition probability for the current state s given by CSSR automaton and the probability computed with the ME model taking into account the features of the studied word ($F = \{f_1, f_2, ...f_n\}$):

$$P(a|s, F) = \mu \cdot P_{CSSR}(a|s) + (1 - \mu) \cdot P_{ME}(a|F)$$

In this way, we combine the power of the ME models, that are expected to give more accurate probabilities given the real context, and the information encoded in the automaton, that can, for example, forbid some transition between states or provide information not captured by the ME models. Note that this way of computing $P(a|s, F)$ includes the two limit cases: if $\mu = 0$, we are taking into account just the probability of the ME model, so we are not using the automaton at all. This will be equivalent to perform directly the Viterbi algorithm using ME models as in the method "Plain ME". On the other hand, if $\mu = 1$, just the probability of the state is used, so it is equivalent to use CSSR without ME models.

Managing Unseen Transitions

As for normal CSSR, when performing the annotation task, there can be symbol combinations that has not been seen in the training corpus and the automaton would fall into a *sink* state. Then, it is necessary to implement some method to evolute to a known state and continue tagging the sentence normally.

For ME-over-CSSR we have used the same methods for dealing with the sink state presented in section 5.4. The difference between this process for CSSR and for ME-over-CSSR is that in the first case, while the system is in the sink state there is no information about the probabilities each possible symbol can have. Nevertheless, when using ME-over-CSSR, if we are in the sink state we still can compute the probability for each symbol given the context using the ME model so we have more information to tag correctly the unknown word that when using simple CSSR. This makes the problem of falling into the sink state less important and the difference between the two sink methods tested for NER not significant.

7.2.4 ME-CSSR

In this approach, CSSR is modified to include the use of ME models in both the step of building the automaton and the annotation task. The main idea of this approach is based on generalizing the concept of history. Instead of considering histories as sequences of alphabet symbols corresponding to the last l_{max} words, we define histories as sets of relevant information about the last l_{max} words. Thus, histories can be encoded as collections of features of the words in a window of size l_{max}. Then, causal states can still be defined as

sets of histories with the same distribution for the future and can be calculated following the structure of CSSR.

In this way, all the information encoded in the features is taken into account when building the automaton and the automaton is expected to better capture the patterns of sequences since it has more information.

Building the automaton with ME-CSSR

Once the histories are extended to be sets of features rather than simple suffixes, the algorithm for building the automaton is conceptually similar to the original CSSR but with some variations:

- As in CSSR, the transitions between states are labelled with the complete symbol alphabet (including the hidden tag), as it is necessary to have a closed and reduced number of transitions. Then, the probability distribution of a history is $P(a|x^-)$ where a is a symbol of the complete alphabet and x^- is a history represented as a set of features.

- Probability distributions for each history are computed using Maximum Entropy models, instead of Maximum Likelihood as original CSSR does. To do so it is necessary to extract the feature set for each word in the training corpus and compute the ME models before executing ME-CSSR. With this models ME-CSSR can compute the probability distribution of the histories and compare it with the distribution of the various causal states using a hypothesis test such as χ^2 test or a distribution distance such as Jensen Shannon distance (Lin, 1991).

- In Sufficiency step, CSSR builds iteratively all possible suffixes and compares their probability distributions with existing causal states. ME-CSSR can not do that as the considered features may not be a closed set. Then, sufficiency is performed studying all histories that have been seen in the training corpus, from length 1 to l_{max} .

- The automaton built using ME-CSSR is not deterministic, but this is not a problem for annotation tasks, since Viterbi algorithm can explore all possible paths and choose the best one. The causal state machine is not deterministic not only because two histories with the same probability distribution for the future can evolute to different states with a given symbol, but also because there can be multiple transitions for a concrete history. For example, consider the history formed by the features of three consecutive words ($l_{max} = 3$):

$$
\begin{aligned}
w1: \ & F_1 = \{f_1^1 \ f_1^2 \ f_1^3 ...\} \\
w2: \ & F_2 = \{f_2^1 \ f_2^2 \ f_2^3 ...\} \\
w3: \ & F_3 = \{f_3^1 \ f_3^2 \ f_3^3 ...\}
\end{aligned}
$$

  The history h_1 is formed by all the features of the last three words: $F_1F_2F_3$, and with a given symbol a will evolute to $F_2F_3F_4$, where F_4 is the set of features for the next word, including the symbol a. Note that the first word disappears because the maximum considered length is 3. The only thing that we know about the history receiving the transition is that its last symbol must be a, but there can be different

valid values of all other features for last position F_4. Thus there can be various existing histories in various states receiving the transitions of history h_1 with symbol a, and it is not possible to make it deterministic by splitting histories in different states as CSSR does since a single history makes the automaton indeterministic. Even a state with just one history would have different possible transitions for each symbol.

Annotation Tasks with ME-CSSR

When the annotation task is to be performed with the learned automaton, the same approach presented in section 5.3 is applied. Thus, the alphabet used for the transitions contains the hidden information and a Viterbi algorithm is applied to determine the most likely symbol sequence which is the best B-I-O tag sequence.

In this case, the difference is that the states and their transitions already encode information about more complicated features. Nevertheless, it is still interesting to use not only the probabilities for the states but also the real features of the word we are tagging to compute the probability of seeing each symbol. In this way, we take maximum advantage of the learned ME models and the information encoded in them. So, in the annotation task the ME model is also used to compute the probability of seeing each symbol (class) given the current context.

As for ME-over-CSSR, to obtain the probability of a determined symbol given a word with its context and being in a state of the automaton, the probability given by the ME model is combined (with a weight μ) with the probability given by the state.

In this way, we have a global transition probability that takes into account both the information of the ME model and of CSSR. The idea is that there can be information encoded in the automaton that can not be encoded in the ME models, and vice versa. Again, if $\mu = 0$, the system will perform equivalently to the method "Plain ME" and if $\mu = 1$, only the transition probabilities of the automaton will be used, tough in this case, this will not be equivalent to CSSR as the states are formed by more complicated histories.

Several experiments with various values of μ will be performed to see the behavior of the annotation system in the case of both methods.

Managing Unseen Transitions

In this case, to solve the problem of falling into the *sink* state, it is not possible to use the same approach used for the basic CSSR or for ME-over-CSSR, where the basic idea was to search for similar suffixes in other states or to wait for a known suffixes, as, if the feature set is complicated enough, there will be lots of unseen histories and waiting for a history equal to some history present in the automaton could take a long time.

Thus, we propose to compare the probability distribution of the current (and unseen) history, computed using the ME model, with all the states in the automaton. This is the same procedure that ME-CSSR does when building the automaton to determine which state each history belongs to. Then, the system evolves to those states that pass the test with the same confidence degree used to learn the automaton.

If no state passes the test, the path is continued using only the ME computed probability and the same operation is performed with next word until the system finds a suitable state for the new history.

7.3 Conclusions of this Chapter

At the sight of the results presented in chapter 6, it was seen that the approach proposed to apply CSSR to annotation NLP tasks had some limitations due to the limited amount of available data and to the complicated nature of such tasks. For that reason, two methods to combine CSSR algorithm with ME models have been devised and presented in this chapter. Next chapter presents the performed experiments with these two methods and with the baseline method, in NER task. In the future, other tasks will be also approached with these methods.

CHAPTER 8

Experiments and Results with ME-extended CSSR

The two methods and the baseline presented in chapter 7 have been tested in the task of Named Entity Recognition (NER).

Regarding ME-CSSR, first experiments were performed using χ^2 hypothesis test, and some preliminary results were presented in FSMNLP'07 workshop (Padró & Padró, 2007a). Nevertheless, these preliminary results, and the study of the error sources showed that this test is not suitable when using ME-CSSR, as we will explain in section 8.3. For that reason we used Jensen Shannon divergence to compare future distributions. The experiments and results obtained with this system, performed with more complicated features are presented in this chapter, and are under review for publication.

8.1 Data and Alphabet

As we have said, the data and alphabet used for NER task with ME-extended CSSR are the same used for the task with CSSR alone. This is, the data are those of CoNLL-2002 shared task (Tjong Kim Sang, 2002a), that have a train corpus and two test corpora and the visible alphabet has 5 symbols which combined with the B-I-O tags lead to a 15 symbol complete alphabet.

8.2 Features

To use the different proposed methods to combine ME and CSSR, it is necessary to define the relevant features for the task.

For the development step, different combinations of some relevant features were tested over the development corpus. For each feature set, the features that appear less are filtered, testing also various filter levels to obtain the best possible model. Furthermore, each model is tested using different parametrization that can affect the performance of the method. The parameters to be tuned for the methods that use ME models and CSSR are: the maximum

length of the histories, the significance degree of the hypothesis test and the ponderation value for the annotation step μ.

Many combinations of various features were used and tested with these methods, here we summarize the features that led to best results. The features we use include information about the words in the sentence, their orthography, their Part of Speech tag and their BIO tag.

All the best feature sets include a basic set of features that we will call *Basic* formed by: word, lemma, PoS tag, sentence-end mark, visible symbol, complete symbol and BIO-tag. Note that for the last two features this information is available in the training corpus, which is necessary to build the models, but not in the test corpus. So, when performing the annotation step, the feature used for the hidden information is the one predicted by the algorithm for the currently studied path.

The other features used in the best models are:

- *BoW, BoL, BoP*: size l_{max} bags of words (BoW), lemmas (BoL) or Part of Speech (BoP).
- *Cap*: Capitalized word.
- *AllCaps*: All letters capitalized.
- *FuncW*: Functional word (words that often appear inside NEs such as conjunctions and prepositions)
- *Num*: Word containing numbers.
- *VisSuf*: Suffix of the last three visible symbols. Can be included for the last word (VisSuf-1) or for the last two words (VisSuf-2).

The five combinations of these features that obtained best performance were:

- **F1** = {*Basic, VisSuf-1*}
- **F2** = {*Basic, VisSuf-1, VisSuf-2*}
- **F3** = {*Basic, VisSuf-1, Cap, AllCaps, FuncW, Num*}
- **F4** = {*Basic, VisSuf-1, VisSuf-2, Cap, AllCaps, FuncW, Num, BoW, BoP*}
- **F5** = {*Basic, VisSuf-1, Cap, AllCaps, FuncW, Num, BoW, BoP, BoL*}

All these features, except the visible suffix are taken into account for each word in a window of size l_{max} to the left of the current word. To maintain the idea of histories it is necessary to consider the same maximum length for all features which will be the length used by CSSR to learn the automaton. All feature sets also include the known part of the symbol (e.g. m or M) and the PoS tag of the current word.

The effects of taking into account various lengths for different features, of introducing features of future words, and how to combine it with CSSR algorithm, remains to be studied.

8.3 Experimental Results

As we have said, first experiments with ME-CSSR were performed using χ^2 hypothesis test, as original CSSR does, but the obtained results were really behind the ones obtained with the other systems, and a study of the error source showed that one of the main problems was the use of χ^2 test. This test is highly dependent on the amount of data, being a poor test if there is few evidence, and this is our case when using ME-CSSR, since when introducing features into the system, there are many more different histories so less occurrences of them. In fact, we realized that when performing the comparison between a determined history and the set of ME-CSSR already created states with χ^2 test, the test would pass for more than one state, often with all the states, so the decision of which state this new history has to belong to was quite arbitrary.

For that reason, we dismissed χ^2 test and focused on the use of Jensen-Shannon divergence to determine if two distribution probabilities are different or not, since some preliminary experiments showed that it was much less ambiguous. Furthermore, in the experiments of NER with simple CSSR, we obtained good results with this divergence so it has proven to be useful for causal state machine reconstruction.

For ME-over-CSSR, since the automaton used is built with original CSSR, the comparison between future distribution can be performed either with χ^2 test or using Jensen-Shannon divergence. Furthermore, any of the configuration explained in 6.3 can be used. We choose to use χ^2 test taking into account all suffixes to determine the recurrent states, since this is the configuration that led to the best results.

The various feature sets presented in section 8.2 were tested over the development corpus with the two proposed methods and with the baseline method, all with different parametrization. The parameters that can vary for each feature set are the threshold frequency under which the features are filtered, and for the systems that use CSSR the ponderation value for the annotation step (μ), the Jensen-Shannon threshold distance (d) to perform the probability distribution comparison (or the significance level of χ^2 test if we use ME-over-CSSR with this test) and the maximum length of the histories (l_{max}).

Table 8.1 shows the best F_1 scores obtained for the various feature sets over the development and the evaluation corpora. The maximum lengths used are 2 and 3, since for $l_{max} = 4$, there are too many possible histories and the system is not able to build an automaton.

The table shows the best results in each case, obtained with various parameters. The best parametrization is determined using the development corpus and then, the system with these parameters is used to annotate the evaluation corpus. The best configuration depends on the chosen features and on the maximum length. Nevertheless, for some parameters a general behavior independent of the features can be observed.

Table 8.1 also shows the parameter μ that led to each result, since it is useful to see how much the systems use the probabilities given by ME models with respect to those of the CSSR automaton. It has been observed that optimal values for μ are around 0.75 for ME-over-CSSR and between 0.25 and 0.5 for ME-CSSR. That means that for the first method, the automaton transitions have an important weight, improving the results of ME models alone when used, but for ME-CSSR, the system obtains better results when giving equal or more weight to the ME models than to the automaton built with ME probabilities.

Regarding the other parameters, for ME-CSSR in all cases the best performance is obtained with $d = 0.01$. The filter level of the features seems to be quite arbitrary, being

	Plain ME		ME-over-CSSR			ME-CSSR			CSSR	
	Devel.	**Eval.**	μ	**Devel.**	**Eval.**	μ	**Devel.**	**Eval.**	**Devel.**	**Eval.**
					$l_{max} = 2$					
F1	86.74	88.04	0.75	88.31	89.73	0.25	87.38	88.66		
F2	86.87	88.70	0.75	88.32	89.64	0.25	87.32	88.27		
F3	86.49	88.56	0.80	88.28	89.51	0.50	86.97	88.32	87.71	88.74
F4	**87.60**	**88.56**	0.80	88.35	89.47	0.50	87.73	87.28		
F5	86.54	88.38	0.70	87.44	89.82	0.25	86.51	86.03		
					$l_{max} = 3$					
F1	86.32	88.11	0.75	88.03	89.02	0.50	86.78	87.80		
F2	87.18	88.93	0.75	87.33	90.05	0.50	86.78	87.82		
F3	86.22	88.77	0.75	88.02	90.12	0.50	84.22	86.39	87.77	88.82
F4	87.20	88.37	0.70	87.81	88.85	0.50	**88.06**	**88.12**		
F5	86.74	88.68	0.75	**88.42**	**90.29**	0.60	86.51	85.90		
					$l_{max} = 4$					
									87.93	**88.58**

Table 8.1: Best obtained results with the various feature sets and the various systems over the development and test corpora for $l_{max} = 2$ and 3.

very influent on the final performance but depending its best value on the feature set and on the maximum length.

In this table, we also show the best results obtained with CSSR alone (section 6.3), in order to compare them with the ones obtained with the various methods that use ME models. We include the results of $l_{max} = 4$ since they are the best results obtained with CSSR.

8.3.1 ME-CSSR in more detail

Since ME-CSSR is conceptually more complicated and introduces an important variation respect CSSR, we are interested in studying it in more detail. To study the automata that this algorithm generates, table 8.2 presents the number of histories, number of states and the average number of transitions per symbol of each automaton. The average number of transitions is computed taking into account, for each state, the symbols that have one or more transitions, since depending on the state, there are lots of symbols that have no transition. We see this value as a way to measure to which extend the automata is indeterministic. The range of the number of transitions goes from 1 to around 50, but low values are much more frequent.

The figures in this table correspond to those of the automata that led to best results for each feature set, shown in table 8.1. If different automata are chosen, the figures change significantly, specially if the automata build with various d threshold are studied. For all feature sets, the best performance was obtained with $d = 0.01$. If this value is increased, the number of states falls, since the system consider more histories equivalent. If it is decreased, more states are generated. These variations are about one order of magnitude for each order

Features	Histories	States	Transitions
$l_{max} = 2$			
F1	220,956	110	6.5
F2	244,153	147	5.9
F3	221,054	132	5.9
F4	472,393	150	6.4
F5	472,149	190	5.9
$l_{max} = 3$			
F1	355,454	162	5.0
F2	374,632	232	4.6
F3	358,499	211	5.1
F4	483,582	247	5.3
F5	481,498	280	5.1

Table 8.2: Number of histories, states and average transitions per symbol for ME-CSSR automata of $l_{max} = 2$ and 3.

in d: if for $d = 0.01$ the number of states is of order 100, for $d = 0.1$ it is of order 10 and for $d = 0.001$ of order 1000.

These figures show that ME-CSSR is able to group histories into states, building relatively small automata (taking into account the amount of different histories), and also that the automata have a reduced number of transitions for each symbol, what is important for the use of the automata. If all states could evolve to almost all other states, performing the annotation step would be computationally very expensive.

8.3.2 Discussion

From the obtained results it can be seen that ME-over-CSSR system leads to better results than using plain ME models with the Viterbi algorithm (baseline) or ME-CSSR. In general, ME-CSSR system slightly improves the performance of the baseline system though the improvement introduced by ME-over-CSSR is more important. Both ME-over-CSSR and ME-CSSR obtain their best results with $l_{max} = 3$, though the difference with the systems built with $l_{max} = 2$ is not always significative.

The best results obtained using only CSSR were $F_1 = 87.93\%$ for the development corpus and $F_1 = 88.58\%$ for the evaluation corpus, with $l_{max} = 4$. This means that ME-over-CSSR outperforms CSSR alone but ME-CSSR does not, obtaining similar but lower results than simple CSSR.

As we did in section 6.3, the obtained results can also be compared with the systems that competed in the CoNLL-2002 shared task. The task was devoted to NE recognition and classification, and our system just performs the first step, so a system that performs NE classification is applied to the output of our system. The classifier we use is the one used by the winner system of CoNLL-2002 shared task (Carreras et al., 2002) and doing so with the system that obtains best results over the development corpus (ME-over-CSSR with $l_{max} = 3$ and F5 set) leads to a $F_1 = 76.79\%$ over the evaluation corpus. This result is not

significantly different from the system that classified 4th (Wu et al., 2002), that reported $F_1 = 76,61\%$ for the evaluation corpus.

8.4 Conclusions of this Chapter

In this chapter, we have presented the obtained results when applying the methods that combine ME and CSSR to NER task. Also a baseline has been tested. Summarizing, we can say that, from the two presented systems, ME-over-CSSR is much simpler and leads to better results than ME-CSSR and the baseline. The second system, ME-CSSR, theoretically more sophisticated, improves slightly the baseline results but is always behind ME-over-CSSR.

The main interest of developing these systems that combine ME and CSSR was to improve the results obtained with CSSR alone. This goal is fulfilled only by ME-over-CSSR in which case the improvement is significant and the system can compete with the 4th system of CoNLL-2002 shared task.

We consider that the obtained results with ME-over-CSSR system are very interesting, since this method improves the CSSR performance combining it with simple ME models. In this way, we developed a fast and simple system that performs competitive NER taking advantage of the CSSR ability to build minimal automata and of the ME capacity to compute probability distributions taking into account many features.

Nevertheless, the fact that ME-CSSR obtains lower results than ME-over-CSSR and only improves the baseline slightly is quite surprising, since it was expected that, being a more informed system that combines the power of ME models and CSSR, the results would be better. This limited performance can be due to the fact that the amount of available data is not enough for CSSR to build correct automata, even if we are using a good method for sparse data such as ME models.

In this line, a point requesting further study is the trade-off between the data-sparseness caused by the fact of viewing histories as feature sets. Since the richer feature set we use, the less occurrences we will have of each particular history, the CSSR algorithm will have less evidence to accurately build the causal states. On the other hand, richer feature sets should produce better ME models, which can compensate this lack of evidence.

CHAPTER 9

Conclusions and Future Directions

To conclude, in this chapter we summarize the main contributions of this dissertation as well as the publications related to them. Also, we state some future research lines that can follow this work.

9.1 Summary and Conclusions

In this thesis, we have presented the application of a finite automata acquisition algorithm to some NLP tasks. This algorithm (CSSR) builds the causal state machine of a process from sequential data. This kind of machines have many interesting properties: they are deterministic, markovian and are the minimal representation of a process. In fact, the automata built with CSSR have the form of Markov Models with several suffixes grouped into each state, building a much smaller automaton than a Markov Model for a given process.

In order to use this algorithm for NLP tasks, first of all we have studied a variety of aspects regarding the algorithm and its behavior when learning sentence patterns. Secondly, we have exposed an approach to introduce hidden information into the algorithm and then apply it to NER in general and medical domain and Chunking. Finally, and considering the limitations that this approach has, we have proposed a method to combine CSSR with ME models in order to build a system that takes into account more information.

9.1.1 Main Contributions

In the following points we summarize the contributions of this work, which we differentiate in three main lines:

1. **Study of the ability of CSSR to capture text patterns.**

   We have performed some preliminary experiments in order to study the algorithm behavior and its dependency on the various parameters. In this step, we have focused on the study of the learned automata when the input for the system are data drawn from

natural language texts, were some features have been highlighted. These automata are expected to reproduce the patterns of a text in terms of the selected features. First of all we have selected features related to Named Entities, in order to see how CSSR represents a text with this kind of entities, and secondly we have studied the ability of the algorithm to capture the patterns of Noun Phrases in terms of the PoS tags of each word.

These experiments have shown that CSSR is able to learn correctly these kind of patterns, though its performance is very dependent on the amount of available data.

2. **Applying CSSR to NLP annotation tasks.**

We have proposed an approach to introduce the hidden information, necessary for the annotation tasks, into the alphabet used by the algorithm. Then, these information is taken into account when the automaton is built and can be afterwards used to annotate new text using a Viterbi algorithm. The main advantage of this technique in comparison, for example, with a Markov Model is that the automata involved are much smaller so it is faster to annotate new text. Also, models with longer suffixes can be built if enough data is available since the number of generated states will be much smaller than the equivalent Markov Model and thus more computationally tractable.

There are three main contributions our research introduces in the field of applying CSSR to annotation task:

(a) The idea of introducing the hidden tags into the visible alphabet, building a complete alphabet and performing a Viterbi algorithm to use the automaton to annotate new sentences.

(b) Two methods for dealing with unseen transitions have been presented. As we are performing annotation tasks with CSSR automata, there can be suffixes appearing in the test corpus that have not been seen in the training corpus. Then, some method to deal with these unseen events has to be developed.

(c) We also have proposed and tested a modification of the algorithm in the way it determines recurrent states. This modification theoretically leads to a loose in the generalization power of CSSR but suits better the approached NLP tasks.

With these different proposed methods to apply CSSR to NLP annotation tasks, we performed several experiments:

(a) The method has been validated by applying it to learn sentence patterns from a corpus annotated with a simple hand-made automaton, and checking that CSSR is able to exactly reproduce its behavior. The results show that CSSR is able to annotate perfectly this corpus.

(b) This approach has been applied to NER, with a deep study of the influence of the various parameters of the algorithm and of some different proposed implementations. It has been shown that CSSR can build automata that give pretty good results in this task.

(c) The system has been also applied to Biomedical Named Entity Recognition, using a very simple approach. The obtained results are comparable to other state-of-the-art systems, though some post-processing techniques should be implemented to improve the performance of CSSR in this task.

(d) Finally, the proposed approach has been used for Chunking task. In this case, various automata must be built to annotate each different kind of chunk and then their predictions must be combined. We have proposed and tested two different methods for combining the various chunkers obtaining not as good results as for NER when compared to other systems, but still being comparable with state-of-the-art systems.

3. **Combining CSSR and Maximum Entropy Models.**

Once the proposed approach to use CSSR for NLP tasks has been tested in various tasks, and seeing that to improve the performance of the system it is necessary to introduce more information into it, we have proposed to combine CSSR algorithm with ME models. The goal of this combination is to take advantage of the good properties of CSSR (for example the ability of building minimal automata that represent a process) but combine it with the power of ME models, that can compute probability distributions taking into account lots of features, what CSSR can not do.

In this line, we have proposed two approaches to combine these two methods. These methods have been tested in NER task and compared with a baseline that used only CSSR. The developed systems and the obtained results can be summarized as follows:

(a) First approach (ME-over-CSSR) combines CSSR and ME models only in the annotation step, using the automata built with normal CSSR and the ME probabilities to predict the most likely tag for each word in a sentence. The results obtained with this system are quite good, beating both the baseline that uses ME models and the results obtained with CSSR alone. That means that the combination of CSSR and ME models really makes a more informed system.

(b) The second approach (ME-CSSR) uses ME models both in the automaton building step and in the annotation step. This is a more complex proposal, that introduces modifications into the algorithm with the goal of building directly more informed automata rather than using the original CSSR automata in combination with ME models. Nevertheless, the obtained results are worse than those of ME-over-CSSR, and do not improve those of CSSR alone. Regarding the baseline, ME-CSSR slightly outperforms it.

To conclude this dissertation, we can say that CSSR algorithm is a powerful algorithm that can learn the patterns of sentences in terms of some selected features, capturing the behavior of NPs or NEs. It also can be satisfactorily applied to NER task, both in general and medical domain but obtains not so competitive results in Chunking task. CSSR performs better as smaller is the vocabulary, since to learn good automata with big alphabets, huge amount of data would be necessary.

In fact, this is one of the main limitations of the algorithm: it needs and important amount of data to perform good statistics and this ties the possibility of improving its performance by taking into account more features and increasing the vocabulary size. For that reason we have devised two methods to combine CSSR with ME models obtaining good results with the first one but not improving CSSR alone for the second one. This opens a door to continue the research in this line, since it has been seen that combining ME and CSSR leads to a better system that using only CSSR, but there are still many issues to be studied.

9.1.2 Related Publications

The work presented in this PhD thesis has been published in various conferences. Here we summarize these publications:

Study of CSSR ability to Learn Language Patterns

- Padró, M., & Padró, L. (2007b). Studying CSSR Algorithm Applicability on NLP Tasks. *Procesamiento del Lenguaje Natural*, *39*, 89–96.

Applying CSSR to NLP Annotating Tasks

- Padró, M., & Padró, L. (2005a). Applying a Finite Automata Acquisition Algorithm to Named Entity Recognition. *Proceedings of 5th International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP'05)*. Helsinki, Finland.
- Padró, M., & Padró, L. (2005b). Approaching Sequential NLP Tasks with an Automata Acquisition Algorithm. *Proceedings of International Conference on Recent Advances in NLP (RANLP'05)*. Bulgaria.
- Padró, M., & Padró, L. (2005c). A Named Entity Recognition System Based on a Finite Automata Acquisition Algorithm. *Procesamiento del Lenguaje Natural*, *35*, 319–326.
- Dowdall, J., Keller, B., Padró, L., & Padró, M. (2007). An Automata Based Approach to Biomedical Named Entity Identification. *Annual Meeting of Proceedings of the Annual Meeting of the ISMB BioLINK Special Interest Group on Text Data Mining*. Vienna, Austria.

Extending CSSR algorithm with ME models

- Padró, M., & Padró, L. (2007a). ME-CSSR: an extension of CSSR using maximum entropy models. *Proceedings of the 2007 Conference on Finite-State Methods for NLP (FSMNLP)*. Potsdam, Germany.

9.2 Future Work

Further work that can be done following the work presented in this dissertation can be divided in two main lines: the first one following the proposed approach to apply CSSR to NLP annotation tasks and the second one regarding the combination of CSSR and ME models. In both directions other applications can be developed and also there is still work to be done regarding the implementation of the system and the tunning of the parameters. In next sections we briefly present some relevant future lines following these two directions.

9.2.1 CSSR for annotation tasks

Modifications of the System

We propose to modify the use of CSSR to make it more similar to a Hidden Markov Model (HMM), rather than using it as a visible Markov Model as done so far. The idea is to define the alphabet of CSSR as the possible hidden information, as an HMM does, and let CSSR learn an automaton reproducing the patterns of this information. This automaton will be similar to the hidden part of a HMM but with much less states. Once the automaton is built, emission probabilities could be added to each state.

Note that this would be a way of building much small HMMs, so probably models involving longer histories than HMMS could be built. Nevertheless, for the kind of tasks approached in this PhD, to build HMMs is not a good approach, since there are just three hidden tags and the states will have too few information.

Other Applications

One of the applications we are interested in using CSSR is PoS tagging. This task can not be directly approached using the proposed method to combine the visible part of the alphabet with the hidden tags since the visible information usually used in this task are the words in a sentence and using them as a visible alphabet would be impossible. Then, it should be approached using the system proposed in previous section.

In this case, the alphabet of CSSR would be built by the PoS tags and CSSR would learn an automata reproducing the patterns of these tags. Then, the possible emissions (that for the case of PoS tagging are the words) for each state will be added and its probabilities computed.

Another point requesting further study is the combination of CSSR with some post-processing techniques, specially for the case of Biomedical NER. Such techniques have been used in other systems, leading to a significative improvement of the results.

9.2.2 CSSR combined with ME models

Modifications of the Systems

Some interesting points requesting further study to improve the systems that combine ME models and CSSR are:

1. The use of different lengths for different features. In this way, just the most relevant features will be taken into account for higher maximum lengths, allowing the use of longer l_{max} values without exploiting the number of considered histories, as happens with the current implementation.

2. The possibility of introducing features for future words into the feature set for both ME-over-CSSR and ME-CSSR. This can be done without problem for ME-over-CSSR system, since it uses the features independently of the histories of the automaton, but it has to be studied carefully for the case of ME-CSSR, where histories include the features of last l_{max} words. This means that introducing features about future words will modify the concept of history a little bit, since the future features for one word will be past features for another word that can be in the same history. For that reason this possibility has to be studied carefully specially for ME-CSSR.

3. Regarding ME-CSSR, we are also interested on studying why the performance is lower than what we expected (often close to the baseline) and looking for some ways to improve it.

4. To improve ME-CSSR performance, one possibility is to use ME models and CSSR together to build a system more similar to a Hidden Markov Model, as it has been proposed in section 9.2.1, but for tasks with B-I-O tags. The idea that we propose is to set the vocabulary to the three hidden tags $\Sigma = \{B, I, O\}$. This would be a too poor vocabulary for original CSSR, but since ME-CSSR builds states taking into account all the features that we want to encode independently of the alphabet, the final automata could be rich enough. Thus, the automaton will have sets of histories that would build the states, the transition between states labeled by a B-I-O tag and a visible part that would be the visible part of the alphabet. This is a modification of the proposed ME-CSSR that separates the visible part of the alphabet and the hidden tags, and we think that it could be useful in the learning step since the vocabulary size (the classes of the ME model) will be much smaller so the estimated probabilities better.

Other Applications

For the case of the two systems that combine ME models and CSSR, the same applications to NLP task proposed with the original CSSR can be approached. In this way, the systems could be applied to tasks such as Biomedical NER and Chunking since CSSR led to reasonably good results but needed more information to improve them, or to PoS tagging as explained in previous section. It would be interesting to see if, as for NER, the system that uses ME models is better than the one that uses only CSSR.

Also, since combining CSSR with ME models allows us to introduce more information into the system, it could be applied to more complex tasks such as NE classification both in general and Biomedical domains, what can not be done with original CSSR because of the limitation in the alphabet size.

Bibliography

Aberdeen, J., Burger, J. D., Day, D., Hirschman, L., Robinson, P., & ain, M. V. (1995). MITRE*: Description of the alembic system used for MUC-6*, 141–155. Proceedings of the 6th Messsage Understanding Conference, MUC-6. Columbia, Maryland: Morgan Kaufmann Publishers, Inc.

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

Alfonseca, E., & Manandhar, S. (2002). An unsupervised method for general named entity recognition and automated concept discovery. *Proceedings of the first International Conference of Global WordNet Association*. Mysore, India.

Ando, R. K., & Zhang, T. (2005). A high-performance semi-supervised learning method for text chunking. *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (pp. 1–9). Morristown, NJ, USA: Association for Computational Linguistics.

Aone, C., Halverson, L., Hampton, T., & Ramos-Santacruz, M. (1998). SRA: Description of the IE2 system used for MUC. *Proceedings of the 7th Messsage Understanding Conference, MUC-7*. Fairfax, Virginia.

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., Kameyama, M., Kehler, A., Martin, D., Myers, K., & Tyson, M. (1995). SRI *international* FASTUS *system muc-6 test results and analysis*, 237–248. Proceedings of the 6th Messsage Understanding Conference. Columbia, Maryland: Morgan Kaufmann Publishers, Inc.

Asahara, M., & Matsumoto, Y. (2003). Japanese named entity extraction with redundant morphological analysis. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, NAACL '03* (pp. 8–15). Morristown, NJ, USA: Association for Computational Linguistics.

Atserias, J., Casas, B., Comelles, E., González, M., Padró, L., & Padró, M. (2006). FreeLing 1.3: Syntactic and semantic services in an open-source NLP library. *Proceedings of the fifth*

international conference on Language Resources and Evaluation (LREC 2006). Genoa, Italy. http://www.lsi.upc.edu/ nlp/freeling.

Baker, J. (1979). Trainable grammars for speech recognition. *Speech communication papers presented at the 97th Meeting of the Acoustical Society* (pp. 547–550).

Bender, O., Och, F. J., & Ney, H. (2003). Maximum entropy models for named entity recognition. *Proceedings of the seventh conference on Natural Language Learning at HLT-NAACL 2003, CoNLL-2003* (pp. 148–151). Morristown, NJ, USA: Association for Computational Linguistics.

Bengio, Y. (1996). Markovian models for sequential data. *Technical Report 1049, Dept. IRO*. Université de Montréal.

Bengio, Y., & Frasconi, P. (1995). An input output HMM architecture. *Advances in Neural Information Processing Systems* (pp. 427–434). The MIT Press.

Berger, A., Pietra, S. D., & Pietra, V. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*, 39–71.

Bikel, D. M., Miller, S., Schwartz, R., & Weischedel, R. (1997). Nymble: A high performance learning name-finder. *Proceedings of the 5th Conference on Applied Natural Language Processing, ANLP*. Washington DC.

Black, W. J., Rinaldi, F., & Mowatt, D. (1998). FACILE: Description of the ne system used for muc-7. *Proceedings of the 7th Message Understanding Conference, MUC-7*. Fairfax, Virginia.

Black, W. J., & Vasilakopoulos, A. (2002). Language-independent named entity classification by modified transformation-based learning and by decisi on tree induction. *Proceedings of CoNLL-200* (pp. 159–162). Taipei, Taiwan.

Borthwick, A., Sterling, J., Agichtein, E., & Grishman, R. (1998). NYU: Description of the MENE named entity system as used in muc-7. *Proceedings of the 7th Message Understanding Conference, MUC-7*. Fairfax, Virginia.

Borthwick, A. E. (1999). *A maximum entropy approach to named entity recognition*. Doctoral dissertation, New York University, New York, NY, USA. Adviser-Ralph Grishman.

Boschetti, F. (2007). Mapping the complexity of ecological models. *Ecological Complexity*, *5*, 37–47.

Bottou, L. (1991). *Une approche théorique de l'apprentissage connexionniste: Applications à la reconnaissance de la parole*. Doctoral dissertation, Université de Paris XI, Orsay, France.

Brill, E. (1995). Transformation–based error–driven learning and natural language processing: A case study in part–of–speech tagging. *Computational Linguistics*, *21*, 543–565.

Buchholz, S., Veenstra, J., & Daelemans, W. (1999). Cascaded grammatical relation assignment. *In Proceedings of EMNLP/VLC-99* (pp. 239–246). University of Maryland, USA.

Budi, I., & Bressan, S. (2003). Association rules mining for name entity recognition. *Proceedings of the Fourth International Conference on Web Information Systems Engineering, WISE '03*. Washington, DC, USA: IEEE Computer Society.

Buhlmann, P., & Wyner, A. J. (1999). Variable length markov chains. *The Annals of Statistics, 27* (pp. 480–513).

Burger, J. D., Henderson, J. C., & Morgan, W. T. (2002). Statistical named entity recognizer adaptation. *Proceedings of CoNLL-2002* (pp. 163–166). Taipei, Taiwan.

Cabré, T., Condamines, A., & Sanjuan, F. I. (2005). Introduction: Application-driven terminology engineering. *Terminology*, *11*, 1–20.

Carreras, X., Chao, I., Padró, L., & Padró, M. (2004). Freeling: An open-source suite of language analyzers. *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*. Lisbon, Portugal.

Carreras, X., & Màrquez, L. (2003). Phrase recognition by filtering and ranking with perceptrons. *Proceedings of the International Conference on Recent Advances on Natural Language Processing (RANLP'03)*. Borovets, Bulgaria.

Carreras, X., Màrquez, L., & Padró, L. (2002). Named entity extraction using adaboost. *Proceedings of CoNLL Shared Task* (pp. 167–170). Taipei, Taiwan.

Carreras, X., Màrquez, L., & Padró, L. (2003a). Learning a perceptron-based named entity chunker via online recognition feedback. *CoNLL 2003 Shared Task Contribution*. Edmonton, Canada.

Carreras, X., Màrquez, L., & Padró, L. (2003b). A simple named entity extractor using adaboost. *CoNLL 2003 Shared Task Contribution*. Edmonton, Canada.

Castellví, M. T. C., Bagot, R. E., & Palatresi, J. V. (2001). Automatic term detection: A review of current systems. In D. Bourigault, C. Jacquemin and M.-C. L'Homme (Eds.), *Recent advances in computational terminology*, 53–88. Amsterdam/Philadelphia: John Benjamins.

Charniak, E. (2000). Bllip 1987-89 wsj corpus release 1. *Linguistic Data Consortium*. Philadelphia.

Chieu, H. L., & Ng, H. T. (2002). Named entity recognition: A maximum entropy approach using global information. *COLING*.

Chieu, H. L., & Ng, H. T. (2003). Named entity recognition with a maximum entropy approach. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003, CoNLL-2003* (pp. 160–163). Morristown, NJ, USA: Association for Computational Linguistics.

Chinchor, N. A. (1998). Overview of muc-7/met-2. *Proceedings of the 7th Message Understanding Conference,MUC-7*. Fairfax, Virginia.

Chinchor, N. A., Robinson, P., & Brown, E. (1998). HUB-4 named entity task definition. *Proceedings of the DARPA Broadcast News Workshop*. Herndon, VA.

Chung, T. M. (2003). A corpus comparison approach for terminology extraction. *Terminology*, *9*, 221–246(26).

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

Cimiano, P., & Völker, J. (2005). Towards large-scale, open-domain and ontology-based named entity classification. *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP'05)* (pp. 166–172). Borovets, Bulgaria: INCOMA Ltd.

Clarke, R. W., Freeman, M. P., & Watkins, N. W. (2003). The application of computational mechanics to the analysis of geomagnetic data. *Physical Review E*, *67*, 016203.

Cointet, J.-P., Faure, E., & Roth, C. (2007). Intertemporal topic correlations in online media. *Proc. ICWSM Intl Conf Weblogs Social Media*. Boulder, CO, USA.

Collins, M. (2002). Ranking algorithms for named-entity extraction: boosting and the voted perceptron. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics,ACL '02* (pp. 489–496). Morristown, NJ, USA: Association for Computational Linguistics.

Collins, M., & Singer, Y. (1999). Unsupervised models for named entity classification. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Crutchfield, J. P. (1994). The calculi of emergence: computation, dynamics and induction. *Proceedings of the NATO advanced research workshop and EGS topical workshop on Chaotic advection, tracer dynamics and turbulent dispersion* (pp. 11–54). New York, NY, USA: Elsevier North-Holland, Inc.

Crutchfield, J. P., & Young, K. (1989). Inferring statistical complexity. *Physical Review Letters*, *63*, 105–108.

Crutchfield, J. P., & Young, K. (1990). Computation at the onset of chaos. *In Wojciech H. Zurek, editor, Complexity, Entropy, and the Physics of Information, volume 8 of Santa Fe Institute Studies in the Sciences of Complexity, pages 223-269*. Massachusetts.

Cucchiarelli, A., & Velardi, P. (2001). Unsupervised named entity recognition using syntactic and semantic contextual evidence. *Computational Linguistics*, *27*, 123–131.

Cucerzan, S., & Yarowsky, D. (1999). Language independent named entity recognition combining morphological and contextual evidence. *Proceedings of the 1999 Joint SIGDAT Conference on EMNLP and VLC* (pp. 90–99).

Cucerzan, S., & Yarowsky, D. (2002). Language independent ner using a unified model of internal and contextual evidence. *Proceedings of CoNLL-2002* (pp. 171–174). Taipei, Taiwan.

Cumby, C., & Roth, D. (2003). Feature extraction languages for propositionalized relational learning. *IJCAI'03 Workshop on Learning Statistical Models from Relational Data*. Acapulco, Mexico.

Curran, J. R., & Clark, S. (2003). Language independent ner using a maximum entropy tagger. *Proceedings of the seventh conference on Natural Language Learning at HLT-NAACL 2003, CoNLL-2003* (pp. 164–167). Morristown, NJ, USA: Association for Computational Linguistics.

Darroch, J. N., & Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, *43*, 1470–1480.

Daumé III, H. (2004). Notes on CG and LM-BFGS optimization of logistic regression. Paper and implementation available at `http://www.cs.utah.edu/~hal/megam/`.

Déjean, H. (2000). Learning syntactic structures with xml. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 133–135). Lisbon, Portugal.

Della Pietra, S., Della Pietra, V. J., & Lafferty, J. D. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*, 380–393.

Dietterich, T. G. (2002). Machine learning for sequential data: A review. *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition* (pp. 15–30). London, UK: Springer-Verlag.

Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S., & Weischedel, R. (2004). The Automatic Content Extraction (ACE) Program–Tasks, Data, and Evaluation. *Proceedings of LREC 2004*, 837–840.

Dowdall, J., Keller, B., Padró, L., & Padró, M. (2007). An automata based approach to biomedical named entity identification. *Annual Meeting of Proceedings of the Annual Meeting of the ISMB BioLINK Special Interest Group on Text Data Mining*. Vienna, Austria.

Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., & Yates, A. (2005). Unsupervised named-entity extraction from the web: an experimental study. *Artificial Intelligence*, *165*, 91–134.

Evans, R. (2003). A framework for named entity recognition in the open domain. *Proceedings of Recent Advances in Natural Language Processing, RANLP'03* (pp. 137 – 144). Borovetz, Bulgaria.

Finkel, J., Dingare, S., Nguyen, H., Nissim, M., Sinclair, G., & Manning, C. (2004). Exploiting context for biomedical entity recognition: From syntax to the web. *Proceedings of the Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*.

Florian, R. (2002). Named entity recognition as a house of cards: Classifier stacking. *Proceedings of CoNLL-2002* (pp. 175–178). Taipei, Taiwan.

Florian, R., Ittycheriah, A., Jing, H., & Zhang, T. (2003). Named entity recognition through classifier combination. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003* (pp. 168–171). Morristown, NJ, USA: Association for Computational Linguistics.

Galil, Z., & Giancarlo, R. (1988). Data structures and algorithms for approximate string matching. *Complexity*, *4*, 33–72.

Garcia, P., Cano, A., & Ruiz, J. (2000). A comparative study of two algorithms for automata identification. *Proceedingf of the 5th International Colloquium on Grammatical Inference, ICGI 2000*.

Garcia, P., Vidal, E., & Casacuberta, F. (1987). Local languages, the sucessor method, and a step towards a general methodology for the inference of regular grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI*, *9(6)*, 841–845.

Gold, E. M. (1978). Complexity of automaton identification from given data. *Information and Control*, *37(3)*, 302–320.

Good, I. J. (1963). Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables. *Annals of Mathematical Statistics*, *34*, 911–934.

Grishman, R., & Sundheim, B. (1996). Message understanding conference-6: a brief history. *Proceedings of the 16th conference on Computational linguistics* (pp. 466–471). Morristown, NJ, USA: Association for Computational Linguistics.

GuoDong, Z., & Jian, S. (2004). Exploring deep knowledge resources in biomedical name recognition. *COLING 2004 International Joint workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP) 2004* (pp. 99–102). Geneva, Switzerland: COLING.

Hammerton, J. (2003). Named entity recognition with long short-term memory. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003, CoNLL-2003* (pp. 172–175). Morristown, NJ, USA: Association for Computational Linguistics.

Hanson, J. E. (1993). *Computational mechanics of cellular automata*. Doctoral dissertation, University of California, Berkeley.

Hendrickx, I., & van den Bosch, A. (2003). Memory-based one-step named-entity recognition: effects of seed list features, classifier stacking, and unannotated data. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003* (pp. 176–179). Morristown, NJ, USA: Association for Computational Linguistics.

Holmes, M. P., & Isbell, C. L. (2006). Looping suffix tree-based inference of partially observable hidden state. *ICML '06: Proceedings of the 23rd international conference on Machine learning* (pp. 409–416). New York, NY, USA: ACM Press.

Humphreys, K., Gaizauskas, R., Huyck, C., Mitchell, B., Cunningham, H., & Wilks, Y. (1998). University of Sheffield: Description of the LaSIE-II system and used for MUC-7. *Proc. MUC-7*.

Jacquemin, C., & Bourigault, D. (2003). Term extraction and automatic indexing. In R. Mitkov (Ed.), *Handbook of computational linguistics*, 599–615. Oxford.

Jansche, M. (2002). Named entity extraction with conditional markov models and classifiers. *Proceedings of CoNLL-2002* (pp. 179–182). Taipei, Taiwan.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical Review*, *106*, 620–630.

Jelinek, F. (1990). Self-organized language modeling for speech recognition. In A. Waibel and K.-F. Lee (Eds.), *Readings in speech recognition*, 450–506. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Ji, H., & Grishman, R. (2006). Data selection in semi-supervised learning for name tagging. *Proceedings of the Workshop on Information Extraction Beyond The Document in ACL'06* (pp. 48–55). Sydney, Australia: Association for Computational Linguistics.

Johansson, C. (2000). A context sensitive maximum likelihood approach to chunking. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 136–138). Lisbon, Portugal.

Kemeny, J. G., & Snell, J. L. (1976). *Finite markov chains*. Undergraduate Texts in Mathematics. New York: Springer. 2nd edition.

Kemeny, J. G., Snell, J. L., & Knapp, A. W. (1976). *Denumerable markov chains*. Graduate Texts in Mathematics. New York: Springer. 2nd edition.

Kim, J. D., Ohta, T., Tateisi, Y., & Tsujii, J. (2003). GENIA corpus a semantically annotated corpus for bio-textmining. *BioInformatics*, *19*, i180–i182.

Kim, J. D., Ohta, T., Tsuruoka, Y., Tateisi, Y., & Collier, N. (2004). Introduction to the bio-entity recognition task at JNLPBA. *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA), Geneva, Switzerland* (pp. 70–75). held in conjunction with COLING'2004.

Kim, J.-H., Kang, I.-H., & Choi, K.-S. (2002). Unsupervised named entity classification models and their ensembles. *Proceedings of the 19th international conference on Computational linguistics* (pp. 1–7). Morristown, NJ, USA: Association for Computational Linguistics.

Klein, D., Smarr, J., Nguyen, H., & Manning, C. D. (2003). Named entity recognition with character-level models. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003* (pp. 180–183). Morristown, NJ, USA: Association for Computational Linguistics.

Koeling, R. (2000). Chunking with maximum entropy models. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 139–141). Lisbon, Portugal.

Krupka, G. R., & Hausman, K. (1998). Isoquest, inc.: Description of the netowl^{TM} extractor system as used for muc-7. *Proceedings of the 7th Message Understanding Conference, MUC-7*. Fairfax, Virginia.

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

Kudo, T., & Matsumoto, Y. (2001). Chunking with support vector machines. *Proceedings of NAACL-2001.*

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Eighteenth International Conference on Machine Learning (ICML-2001)* (pp. 282–289). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Lari, K., & Young, S. J. (1990). The estimation of stochastic context-gree grammars using the inside-outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Lari, K., & Young, S. J. (1991). Application of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, *5*, 237–257.

Lauritzen, S. L. (1996). *Graphical models.* Oxford University Press.

Lee, K.-J., Hwang, Y.-S., Kim, S., & Rim, H.-C. (2004). Biomedical named entity recognition using two-phase model based on svms. *J. of Biomedical Informatics*, *37*, 436–447.

Lee, Y.-S., & Wu, Y.-C. (2007). A robust multilingual portable phrase chunking system. *Expert Systems with Applications*, *33*, 590–599.

L'Homme, M.-C., Heid, U., & Sager, J. C. (2003). Terminology during the past decade (1994-2004). *Terminology*, *9*, 151–161.

Li, H., Webster, J., & Kit, C. (2003a). Transductive hmm based chinese text chunking. *Proceeding of International Conference on Natural Language Processing and Knowledge Engineering* (pp. 257–262).

Li, H., Zhu, J., & Yao, T. (2004). Svm based chinese text chunking. *Journal of Chinese Information Processing*, *18*, 1–7.

Li, S., Liu, Q., & Yang, Z. (2003b). Chunking based on maximum entropy. *Chinese Journal of Computer*, *25*, 1734–1738.

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

Lin, Y.-F., Tsai, T.-H., Chou, W.-C., Wu, K.-P., Sung, T.-Y., & Hsu, W.-L. (2004). A maximum entropy approach to biomedical named entity recognition. *BIOKDD* (pp. 56–61).

Malouf, R. (2002a). A comparison of algorithms for maximum entropy parameter estimation. *CoNLL-2002* (pp. 49–55). Taipei, Taiwan.

Malouf, R. (2002b). Markov models for language-independent named entity recognition. *Proceedings of CoNLL-2002* (pp. 187–190). Taipei, Taiwan.

Manning, C. D., & Schütze, H. (1998). *Foundations of statistical natural language processing.* The MIT Press.

Mansouri, A., Affendey, L. S., & Mamat, A. (2008a). Named entity recognition approaches. *International Journal of Computer Science and Network Security, IJCSNS*, *8*, 339–344.

Mansouri, A., Affendey, L. S., & Mamat, A. (2008b). Named entity recognition using a new fuzzy support vector machine. *International Journal of Computer Science and Network Security, IJCSNS*, *8*, 320–325.

Mayfield, J., McNamee, P., & Piatko, C. (2003). Named entity recognition using hundreds of thousands of features. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003* (pp. 184–187). Morristown, NJ, USA: Association for Computational Linguistics.

McCallum, A. (1995a). Instance-based utile distinctions for reinforcement learning with hidden state. *ICML* (pp. 387–395).

McCallum, A. (1995b). *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation, Department of Computer Science, University of Rochester.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proc. 17th International Conf. on Machine Learning* (pp. 591–598). Morgan Kaufmann, San Francisco, CA.

McCallum, A., & Li, W. (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003, CoNLL-2003* (pp. 188–191). Morristown, NJ, USA: Association for Computational Linguistics.

McNamee, P., & Mayfield, J. (2002). Entity extraction without language-specific resources. *Proceedings of CoNLL-2002* (pp. 183–186). Taipei, Taiwan.

Meulder, F. D., & Daelemans, W. (2003). Memory-based named entity recognition using unannotated data. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003* (pp. 208–211). Morristown, NJ, USA: Association for Computational Linguistics.

Mikheev, A., Grover, C., & Moens, M. (1998). Description of the LTG system used for muc-7. *Proceedings of the 7th Message Understanding Conference, MUC-7*. Fairfax, Virginia.

Mikheev, A., Moens, M., & Grover, C. (1999). Named entity recognition without gazetteers. *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics, EACL'99* (pp. 1–8). Morristown, NJ, USA: Association for Computational Linguistics.

Munro, R., Ler, D., & Patrick, J. (2003). Meta-learning orthographic and contextual models for language independent named entity recognition. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003, CoNLL-2003* (pp. 192–195). Morristown, NJ, USA: Association for Computational Linguistics.

Murphy, K. P. (1996). Passively learning finite automata. *Technical Report 96-04-017*. Santa Fe Institute.

Nadas, A. (1984). Estimation of probabilities in the language model of the ibm speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *4*, 859–861.

Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Journal of Linguisticae Investigationes*, *30*.

Nakagawa, H., & Mori, T. (2003). Automatic term recognition based on statistics of compound nouns and their components. *Terminology*, *9*, 201–219.

N.Prieto (1995). *Aprendizaje de modelos semánticos para sistemas de comprensión del discurso continuo*. Doctoral dissertation, Universidad Politécnica de Valencia. Advisor: Dr. E.Vidal.

Oncina, J. M., & Garcia, P. (1991). Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis, volume 1 of Series in Machine Perception and Artificial Intelligence* (pp. 49–61). World Scientific.

Osborne, M. (2000). Shallow parsing as part-of-speech tagging. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 145–147). Lisbon, Portugal.

Padró, M., & Padró, L. (2005a). Applying a finite automata acquisition algorithm to named entity recognition. *Proceedings of 5th International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP'05)*. Helsinki, Finland.

Padró, M., & Padró, L. (2005b). Approaching sequential NLP tasks with an automata acquisition algorithm. *Proceedings of International Conference on Recent Advances in NLP (RANLP'05)*. Bulgaria.

Padró, M., & Padró, L. (2005c). A named entity recognition system based on a finite automata acquisition algorithm. *Procesamiento del Lenguaje Natural*, *35*, 319–326.

Padró, M., & Padró, L. (2007a). ME-CSSR: an extension of CSSR using maximum entropy models. *Proceedings of the 2007 Conference on Finite-State Methods for NLP (FSMNLP)*. Potsdam, Germany.

Padró, M., & Padró, L. (2007b). Studying CSSR algorithm applicability on NLP tasks. *Procesamiento del Lenguaje Natural*, *39*, 89–96.

Park, J., won Lee, J., Jo, H.-H., Yang, J.-S., & Moon, H.-T. (2006). Complexity and entropy density analysis of the korean stock market. *JCIS*.

Park, S.-B., & Zhang, B.-T. (2003). Text chunking by combining hand-crafted rules and memory-based learning. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, ACL'03* (pp. 497–504). Morristown, NJ, USA: Association for Computational Linguistics.

Patrick, J., Whitelaw, C., & Munro, R. (2002). Slinerc: The sydney language-independent named entity recogniser and classifier. *Proceedings of CoNLL-2002* (pp. 199–202). Taipei, Taiwan.

Perry, N., & Binder, P. M. (1999). Finite statistical complexity for sofic systems. *Physical Review E*, *60*, 459–463.

Pla, F. (2000). *Etiquetado léxico y análisis sintáctico superficial basado en modelos estadísticos.* Doctoral dissertation, Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València.

Pla, F., Molina, A., & Prieto, N. (2000). Improving Chunking by means of Lexical-Contextual Information in Statistical Language Models. *Proceedings of ConNLL–2000.* Lisbon, Portugal.

Ponomareva, N., Pla, F., Molina, A., & Rosso, P. (2007a). Biomedical named entity recognition: A poor knowledge hmm-based approach. *NLDB* (pp. 382–387).

Ponomareva, N., Rosso, P., Pla, F., & Molina, A. (2007b). Conditional random fields vs. hidden markov models in a biomedical named entity recognition task. *In Proceedings of RANLP 2007.* Borovets, Bulgaria.

Rabiner, L. R. (1990). *A tutorial on hidden markov models and selected applications in speech recognition.* Readings in Speech Recognition (eds. A. Waibel, K. F. Lee). San Mateo, CA: Morgan Kaufmann Publishers, Inc.

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

Ratnaparkhi, A. (1996). A maximum entropy part–of–speech tagger. *Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing, EMNLP.*

Ratnaparkhi, A. (1997). *A simple introduction to maximum entropy models for natural language processing* Technical Report 97-08. Institute for Research in Cognitive Science, University of Pennsylvania.

Rau, L. F. (1991). Extracting company names from text. *Seventh IEEE Conference on Artificial Intelligence Applications* (pp. 29–32).

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

Riloff, E., & Jones, R. (1999). Learning dictionaries for information extraction by multi-level bootstrapping. *Proceedings of National Conference on Artificial Intelligence* (pp. 474–479).

Rinaldi, F., Dowdall, J., Schneider, G., & Persidis, A. (2004a). Answering questions in the genomics domain. *Proceedings of the ACL04 Workshop on Question Answering in Restricted Domains.* Barcelona, Spain.

Rinaldi, F., Hess, M., Dowdall, J., Molla, D., & Schwitter, R. (2004b). Question answering in terminology-rich technical domains. In M. T. Maybury (Ed.), *New directions in question answering*, 71–82. AAAI Press.

Rissanen, J. (1983). A universal data compression system. *IEEE Transactions in Information Theory, IT-29:656-664.*

Ron, D., Singer, Y., & Tishby, N. (1994a). Learning probabilistic automata with variable memory length. *Computational Learing Theory* (pp. 35–46).

Ron, D., Singer, Y., & Tishby, N. (1994b). The power of amnesia. *Advances in Neural Information Processing Systems* (pp. 176–183). Morgan Kaufmann Publishers, Inc.

Rulot, H. (1992). *Ecgi: Un algoritmo de inferencia gramatical basado en la corrección de errores*. Doctoral dissertation, Universitat de València. Advisor(s): Dr. E. Vidal and Dr. F. Casacuberta.

Rulot, H., & Vidal, E. (1987). Modelling (sub)string-length based constraints through a grammatical inference method. In *Pattern recognition theory and applications*, 451–459. Springer-Verlag.

Sanjuan, E., Dowdall, J., Ibekwe-Sanjuan, F., & Rinaldi, F. (2005). A symbolic approach to automatic multiword term structuring. *Computer Speech and Language*, *19*, 524–542.

Santos, D., Seco, N., Cardoso, N., & Vilela, R. (2006). HAREM: an Advanced NER Evaluation Contest for Portuguese. *Proceedings of LREC'2006* (pp. 1986–1991). Genoa, Italy.

Schütze, H., & Singer, Y. (1994). Part-of-speech tagging using a variable memory markov model. *Proceedings of ACL 32'nd*.

Segarra, E. (1993). *Una aproximación inductiva a la comprensión del discurso continuo*. Doctoral dissertation, Universidad Politécnica de Valencia.

Sekine, S. (1998). Nyu: Description of the japanese ne system used for Met-2. *Proceedings of the 7th Message Understanding Conference, MUC-7.*. Fairfax, Virginia.

Sekine, S., & Isahara, H. (2000). Irex: Ir and ie evaluation project in japanese. *2nd International Conference on Language Resources and Evaluation (LREC) 2000* (pp. 1475–1480).

Shalizi, C. R. (2001). *Causal architecture, complexity, and self-organization in time series and cellular automata*. Doctoral dissertation, University of Michigan.

Shalizi, C. R., & Crutchfield, J. P. (2001). Computational mechanics: pattern, prediction strucutre and simplicity. *Journal of Statistical Physics*, *104*, 817–879.

Shalizi, C. R., & Shalizi, K. L. (2004). Blind construction of optimal nonlinear recursive predictors for discrete sequences. *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference*.

Shalizi, C. R., Shalizi, K. L., & Crutchfield, J. P. (2002). An algorithm for pattern discovery in time series. *SFI Working Paper 02-10-060*.

Shalizi, C. R., Shalizi, K. L., & Haslinger, R. (2004). Quantifying self-organization with optimal predictors. *Physical Review Letters*, *93*.

Shinyama, Y., & Sekine, S. (2004). Named entity discovery using comparable news articles. *Proceedings of the 20th international conference on Computational Linguistics, COLING '04*. Morristown, NJ, USA: Association for Computational Linguistics.

Spasic, S., Ananiadou, S., McNaught, J., & Kumar, A. (2005). Text mining and ontologies in biomedicine: Making sense of raw text. *Briefings in Bioinformatics*.

Stolcke, A., & Omohundro, S. (1993). Hidden markov model induction by bayesian model merging. *In Stephen Jose Hanson, J. D. Gocwn, and C. Lee Giles, editors, Advances in Neural Information Processing Systems, volume 5, pages 11-18. Morgan Kaufmann, San Mateo, California*.

Thollard, F., & Clark, A. (2002). Shallow parsing using probabilistic grammatical inference. *ICGI '02: Proceedings of the 6th International Colloquium on Grammatical Inference* (pp. 269–282). London, UK: Springer-Verlag.

Tino, P., & Dorner, G. (2001). Predicting the future of discrete sequences from fractal representations of the past. *Machine Learning*, *45*, 187–217.

Tjong Kim Sang, E. F. (2000). Text chunking by system combination. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 151–153). Lisbon, Portugal.

Tjong Kim Sang, E. F. (2002a). Introduction to the conll-2002 shared task: Language-independent named entity recognition. *Proceedings of CoNLL-2002* (pp. 155–158). Taipei, Taiwan.

Tjong Kim Sang, E. F. (2002b). Memory-based named entity recognition. *Proceedings of CoNLL-2002* (pp. 203–206). Taipei, Taiwan.

Tjong Kim Sang, E. F., & Buchholz, S. (2000). Introduction to the conll-2000 shared task: Chunking. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 127–132). Lisbon, Portugal.

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. *Proceedings of CoNLL-2003* (pp. 142–147). Edmonton, Canada.

Tjong Kim Sang, E. F., & Veenstra, J. (1999). Representing text chunks. *Proceedings of EACL'99* (pp. 173–179). Bergen, Norway.

Trakhtenbrot, B., & Barzdin, Y. (1973). *Finite automata: Behaviour and synthesis*. North Holland Publishing Company.

Tsukamoto, K., Mitsuishi, Y., & Sassano, M. (2002). Learning with multiple stacking for named entity recognition. *Proceedings of CoNLL-2002* (pp. 191–194). Taipei, Taiwan.

Upper, D. R. (1997). *Theory and algorithms for hidden markov models and generalized hidden markov models*. Doctoral dissertation, University of California at Berkeley.

van Halteren, H. (2000). Chunking with wpdv models. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 154–156). Lisbon, Portugal.

Varn, D. P., & Crutchfield, J. P. (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. *Nikos Fakotakis (ed), Machine learning in human language technology, workshop at ACAI 99*. Chania, Greece.

Veenstra, J., & van den Bosch, A. (2000). Single-classifier memory-based phrase chunking. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 157–159). Lisbon, Portugal.

Vidal, E., Thollard, F., C. de la Higuera, F. C., & Carrasco, R. (2005a). Probabilistic finite-state machines - part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*, 1013–1025.

Vidal, E., Thollard, F., C. de la Higuera, F. C., & Carrasco, R. (2005b). Probabilistic finite-state machines - part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*, 1025–1039.

Vilain, M., & Day, D. (2000). Phrase parsing with rule sequence processors: an application to the shared conll task. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 160–162). Lisbon, Portugal.

Wallach, H. M. (2004). *Conditional random fields: An introduction* (Technical Report MS-CIS-04-21). Department of Computer and Information Science, University of Pennsylvania.

Weeds, J., Dowdall, J., Schneider, G., Keller, B., & Weir, D. (2005). Using distributional similarity to organise biomedical terminology. *Terminology*, *1*, 107 – 141.

Weinberger, M. J., Lempel, A., & Ziv, J. (1992). A sequential algorithm for the universal coding of finite memory sources. *IEEE Transactions on Information Theory*, *38*, 1002–1014.

Weinberger, M. J., Rissanen, J., & Feder, M. (1995). A universal finite memory source. *IEEE Transactions on Information Theory*, *41*, 643–652.

Weischedel, R. (1995). BBN*: Description of the* PLUM *system as used for muc-6*, 55–69. Proceedings of the 6th Messsage Understanding Conference. Columbia, Maryland: Morgan Kaufmann Publishers, Inc.

Whitelaw, C., & Patrick, J. (2003). Evaluating corpora for named entity recognition using character-level features. *Australian Conference on Artificial Intelligence* (pp. 910–921). Springer.

Willems, F., Shtarkov, Y., & Tjalkens, T. (1995). The context-tree weighting method:basic properties. *IEEE Transactions in Information Theory, IT-41:653-664.*

Wu, D., Ngai, G., & Carpuat, M. (2003). A stacked, voted, stacked model for named entity recognition. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003, CoNLL-2003* (pp. 200–203). Morristown, NJ, USA: Association for Computational Linguistics.

Wu, D., Ngai, G., Carpuat, M., Larsen, J., & Yang, Y. (2002). Boosting for named entity recognition. *Proceedings of CoNLL-2002* (pp. 195–198). Taipei, Taiwan.

Wu, Y.-C., Chang, C.-H., & Lee, Y.-S. (2006). A general and multi-lingual phrase chunking model based on masking method. *CICLing* (pp. 144–155).

Yu, S., Bai, S., & Wu, P. (1998). Description of the kent ridge digital labs system used for muc-7. *Proceedings of the 7th Message Understanding Conference, MUC-7*. Fairfax, Virginia.

Zhang, T., Damerau, F., & Johnson, D. (2001). Text chunking using regularized winnow. *Meeting of the Association for Computational Linguistics* (pp. 539–546).

Zhang, T., Damerau, F., & Johnson, D. (2002). Text chunking based on a generalization of winnow. *J. Mach. Learn. Res.*, *2*, 615–637.

Zhang, T., & Johnson, D. (2003). A robust risk minimization based named entity recognition system. *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003, CoNLL-2003* (pp. 204–207). Morristown, NJ, USA: Association for Computational Linguistics.

Zhou, G., & Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (pp. 473–480). Morristown, NJ, USA: Association for Computational Linguistics.

Zhou, G., Su, J., & Tey, T. (2000). Hybrid text chunking. *Proceedings of CoNLL-2000 and LLL-2000* (pp. 163–166). Lisbon, Portugal.