Pontifícia Universidade Católica
do Rio de Janeiro

Eraldo Luís Rezende Fernandes

Entropy Guided Feature Generation for Structure Learning

Tese de Doutorado

Thesis presented to the Programa de Pós-Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Doutor.

Advisor: Prof. Ruy Luiz Milidiú

Rio de Janeiro
September 2012

Pontifícia Universidade Católica
do Rio de Janeiro

Eraldo Luís Rezende Fernandes

Entropy Guided Feature Generation for Structure Learning

Thesis presented to the Programa de Pós-Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Doutor.

Prof. Ruy Luiz Milidiú
Advisor
Departamento de Informática — PUC–Rio

Prof. Marcus Vinicius Soledade Poggi de Aragão
Departamento de Informática — PUC–Rio

Prof. Valmir Carneiro Barbosa
UFRJ

Prof. Cícero Nogueira dos Santos
IBM Research

Prof. Daniel Schwabe
Departamento de Informática — PUC–Rio

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico — PUC–Rio

Rio de Janeiro — September 6, 2012

Eraldo Luís Rezende Fernandes

Graduated from the Universidade Federal de Mato Grosso do Sul in Ciência da Computação and obtained a degree of Mestre em Informática at PUC–Rio. He is a lecturer and a researcher at the Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Bibliographic data

Fernandes, Eraldo Luís Rezende

Entropy Guided Feature Generation for Structure Learning / Eraldo Luís Rezende Fernandes ; advisor: Ruy Luiz Milidiú. — 2012.
93 f. : il. ; 30 cm

Tese (Doutorado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2012.
Inclui bibliografia

1. Informática – Teses. 2. Aprendizado de Estruturas. 3. Geração de Atributos. 4. Entropia. 5. Processamento de Linguagem Natural. I. Milidiú, Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

First of all, I am thankful for all my family which has always supported me. We have had amazing moments during so many meetings, and it is always my pleasure to be with them. Specially, I would like to thank my parents, Eraldo and Sônia, my sister, Luciana, and my little brother, Lúciu. They constitute my ultimate foundation for all aspects of my life.

I am unable to express in words all my love and gratitude for my dear wife, Valéria. She standed by me during the most difficult and sad moments of my PhD course. And, more importantly, we are always enjoying each other and having fun with the simplest things.

I also want to thank my sisters-in-law, Vanessa and Patrícia, and my brother-in-law, Márcio.

I will never forget the great moments I have spent in the lab. Thank you very much, all *Learnlings*. Specially, I am glad for having closely worked with Carlos, Cícero, Eduardo, Leandro and William.

I am thankful to my advisor, Ruy, who have taught me so many things regarding different aspects of my life.

I thank all my friends from Barcelona, where I spent one great year at Yahoo! Research Lab. Specially, I need to thank my supervisor and friend, Ulf, and all the *Berlineros*.

Finally, I need to thank my colleagues from IFG for all the support. Specially, José Antônio, his family and Sérgio Henrique.

Abstract

Fernandes, Eraldo Luís Rezende; Milidiú, Ruy Luiz. **Entropy Guided Feature Generation for Structure Learning**. Rio de Janeiro, 2012. 93p. DSc Thesis — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Structure learning consists in learning a mapping from inputs to structured outputs by means of a sample of correct input-output pairs. Many important problems fit into this setting. Natural language processing provides several tasks that can be formulated and solved as structure learning problems. Dependency parsing, for instance, involves the prediction of a tree underlying a sentence. Feature generation is an important subtask of structure learning which, usually, is partially solved by a domain expert that builds complex discriminative feature templates by conjoining the available basic features. This is a limited and expensive way to generate features and is recognized as a modeling bottleneck.

In this work, we propose an automatic feature generation method for structure learning problems. This method is entropy guided since it generates complex features based on the conditional entropy of local output variables given the available input features. We experimentally compare the proposed method with two important alternative feature generation methods, namely manual template generation and polynomial kernel methods. Our experimental findings indicate that the proposed method is more attractive than both alternatives. It is much cheaper than manual templates and computationally faster than kernel methods. Additionally, it is simpler to control its generalization performance than with kernel methods.

We evaluate our method on nine datasets involving five natural language processing tasks and four languages. The resulting systems present state-of-the-art comparable performances and, particularly on part-of-speech tagging, text chunking, quotation extraction and coreference resolution, remarkably achieve the best known performances on different languages like Arabic, Chinese, English, and Portuguese. Furthermore, our coreference resolution systems achieve the very first place on the Conference on Computational Natural Language Learning 2012 Shared Task. The competing systems were ranked by the mean score over three languages: Arabic, Chinese and English. Our approach obtained the best performances among all competitors for all the three languages.

Our feature generation method naturally extends the general structure learning framework and is not restricted to natural language processing tasks.

Keywords

Structure Learning. Feature Generation. Entropy. Natural Language Processing.

Resumo

Fernandes, Eraldo Luís Rezende; Milidiú, Ruy Luiz. **Geração de Atributos Guiada por Entropia para Aprendizado de Estruturas**. Rio de Janeiro, 2012. 93p. Tese de Doutorado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Aprendizado de estruturas consiste em aprender um mapeamento de variáveis de entrada para saídas estruturadas a partir de exemplos de pares entrada-saída. Vários problemas importantes podem ser modelados desta maneira. O processamento de linguagem natural provê diversas tarefas que podem ser formuladas e solucionadas através do aprendizado de estruturas. Por exemplo, parsing de dependência envolve o reconhecimento de uma *árvore* implícita em uma frase. Geração de atributos é uma sub-tarefa importante do aprendizado de estruturas. Geralmente, esta sub-tarefa é realizada por um especialista que constrói gabaritos de atributos complexos e discriminativos através da combinação dos atributos básicos disponíveis na entrada. Esta é uma forma limitada e cara para geração de atributos e é reconhecida como um gargalo de modelagem.

Neste trabalho, propomos um método automático para geração de atributos para problemas de aprendizado de estruturas. Este método é *guiado por entropia* já que é baseado na entropia condicional de variáveis locais de saída dados os atributos básicos. Comparamos experimentalmente o método proposto com dois métodos alternativos para geração de atributos: geração manual e métodos de kernel polinomial. Nossos resultados mostram que o método de geração de atributos guiado por entropia é superior aos dois métodos alternativos em diferentes aspectos. Nosso método é muito mais barato do que o método manual e computacionalmente mais rápido que o método baseado em kernel. Adicionalmente, ele permite o controle do seu poder de generalização mais facilmente do que métodos de kernel.

Nós avaliamos nosso método em nove datasets envolvendo cinco tarefas de linguística computacional e quatro idiomas. Os sistemas desenvolvidos apresentam resultados comparáveis aos melhores sistemas atualmente e, particularmente para etiquetagem morfossintática, identificação de sintagmas, extração de citações e resolução de coreferência, obtêm os melhores resultados conhecidos para diferentes idiomas como Árabe, Chinês, Inglês e Português. Adicionalmente, nosso sistema de resolução de coreferência obteve o primeiro lugar na competição *Conference on Computational Natural Language Learning 2012 Shared Task*. O sistema vencedor foi determinado pela média de desempenho em três idiomas: Árabe, Chinês e Inglês. Nosso sistema obteve o melhor desempenho nos três idiomas avaliados.

Nosso método de geração de atributos estende naturalmente o framework de aprendizado de estruturas e não está restrito a tarefas de processamento de linguagem natural.

Palavras–chave

Aprendizado de Estruturas. Geração de Atributos. Entropia. Processamento de Linguagem Natural.

Contents

List of Figures

List of Tables

1
Introduction

Machine learning (ML) is a very active research field whose main objective is to learn a prediction function from a given set of examples. In the last decades, ML has been successfully applied in many fields, such as natural language processing, information extraction, computer vision, and computational biology. There are several learning paradigms, but in this work we focus on *supervised* machine learning. In this case, the training examples comprise inputs and their correct outputs.

A classic ML problem is binary classification, in which the prediction output is binary. Here, the learned model discriminates between two classes of examples. However, many important problems involve the prediction of a structure. In these cases, the prediction output comprises many variables with complex interdependencies. Natural language processing (NLP) includes many *structure learning* (SL) problems, such as dependency parsing (DP), part-of-speech (POS) tagging, quotation extraction and coreference resolution. Dependency parsing is to identify a *tree* underlying a given sentence. In POS tagging, for a given input sentence, the prediction output is a *sequence* of tags. In quotation extraction, an input document is *segmented* into non-overlapping quotes that, additionally, are associated with their authors. Given a document with mentions to entities from the real world – like people, companies and places – coreference resolution consists in *clustering* mentions that are references to the same entity.

1.1
Ad Hoc Approaches

Many approaches to solve structured problems are based on complex ML systems that combine several basic classifiers. In order to consider the interdependencies among output variables, the basic classifiers are trained and applied by *ad hoc* strategies that pass information from one classifier to another during training and enforce constraints on the prediction outputs of the basic classifiers. One of the most basic structured problems is multiclass classification. This is a generalization of the binary classification problem

where one needs to discriminate among $K > 2$ classes of interest. There are two common approaches to perform this task by decomposing it on independent binary classification problems. The *one-vs-all* approach trains K binary classifiers, where the k-th classifier, for $k \in \{1, \ldots, K\}$, is trained to discriminate the instances of class k from instances of all other classes. To classify an unseen example, all K classifiers are applied and a rule is used to choose only one class. Another approach to multiclass classification is the *one-vs-one* technique. In this approach, $K(K-1)/2$ binary classifiers are trained in order to discriminate between each pair of classes. To classify an unseen example, a voting scheme is used to enforce the unique-class constraint.

RelHunter (Fernandes et al., 2010b,c) is a general method for relation extraction from text that is based on task decomposition and entropy-guided transformation learning (ETL) (Milidiú et al., 2008; dos Santos and Milidiú, 2009b). Besides the fact that ETL considers complex output dependencies, it is tailored for sequential outputs and thus can not directly consider arbitrary structures. In quotation extraction, for instance, RelHunter trains two ETL models to identify tokens that start or end a quote. Then, another ETL model is trained to discriminate which pairs of start-end tokens are correct quotes and, additionally, to associate them with their authors. A task specific heuristic is later applied to discard overlapping quotes. In Fernandes et al. (2010c), RelHunter is further applied to text chunking, clause identification, hedge detection, and dependency parsing.

Dependency parsing (DP) (Buchholz and Marsi, 2006) is to identify the words that syntactically modify other words in a given sentence. This dependency structure corresponds to a rooted tree whose nodes are the sentence words. In Figure 1.1, we show a dependency tree example. As usual,

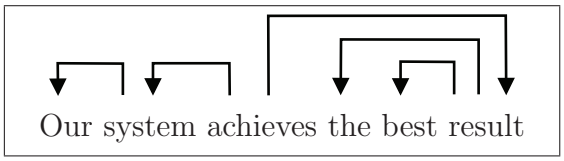


Figure 1.1: Dependency tree example.

in this example, the root node is the main verb of the sentence. An edge (i, j) connects the i-th token to the j-th token and indicates that the latter modifies the former. Additionally, the i-th token is called the *head* token and the j-th token is called the *modifier*. Nivre et al. (2006) propose a dependency parser that relies on a left-to-right deterministic parsing algorithm. Four support vector machines (SVM) are trained to predict parsing actions given features that represent the parser history. Previous word predictions are incorporated in order to consider the interdependency between predictions. Milidiú et al.

(2009) cast DP as a token classification task by using a specific tagging style that provides good generalization. In this tagging style, a modifier token class uniquely identifies its head token. A modifier tag is given by the concatenation of three values: the head token POS tag; how many tokens with the same POS of the head there are between the modifier and its head; and if the head is to the left or to the right of the modifier token. Then, an ETL model is directly trained on the given sentences, along with their features, to predict these specific token tags. This approach achieves good performance but has limitations to generalize on complex sentences with several clauses, since these instances can involve arbitrarily long distances, even with this relative distance.

These ad hoc approaches are developed on a per-task basis and do not provide a general neither principled design pattern. Thus, they are not directly generalized to arbitrary structures and, moreover, most of them have no theoretical guarantees regarding their performances.

1.2 Linear Discriminative Models

In this work, we are interested in *linear discriminative* methods that, regarding binary classification, for instance, learn the parameters of the following linear discriminant function

$$s(\boldsymbol{x}; \boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}) \rangle = \sum_{m=1,\ldots,M} w_m \cdot \phi_m(\boldsymbol{x}), \tag{1-1}$$

where $\boldsymbol{\Phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_M(\boldsymbol{x}))$ is a vector of M real-valued feature functions, or simply *features*, that represents the input $\boldsymbol{x}$; $\boldsymbol{w} = (w_1, \ldots, w_M)$ is the parameter vector – also called *model*, which is estimated from examples; and $\langle \cdot, \cdot \rangle$ is the scalar product operator. Then, the prediction function for a given model $\boldsymbol{w}$ is simply

$$F(\boldsymbol{x}; \boldsymbol{w}) = \begin{cases} +1 & \text{if } s(\boldsymbol{x}; \boldsymbol{w}) \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{1-2}$$

The learning problem is then to estimate $\boldsymbol{w}$ from a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$ comprising correct input-output pairs $(\boldsymbol{x}, y)$, such that $\boldsymbol{x}$ is an input and $y \in \{-1, +1\}$ is the corresponding binary output.

There are plenty of training algorithms for binary classification that follow the *empirical risk minimization* (ERM) principle (Vapnik, 1998). The empirical risk of a model $\boldsymbol{w}$ on the training set $\mathcal{D}$ is given by

$$\mathcal{R}(\mathcal{D}, \boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathbf{1}[y \neq F(\boldsymbol{x}; \boldsymbol{w})], \tag{1-3}$$

where $\mathbf{1}[p]$ is equal to 1 if p is true and 0 otherwise. In words, the empirical risk is the number of misclassified examples in the training data.

The binary perceptron (Rosenblatt, 1957) is an online algorithm that starts from a null model $\boldsymbol{w} = \mathbf{0}$ and iteratively updates its parameters. In Figure 1.2, we present the pseudo-code of this algorithm. At each iteration,

$\boldsymbol{w} \leftarrow \mathbf{0}$
while no convergence
  for each $(\boldsymbol{x}, y) \in \mathcal{D}$
    $\hat{y} \leftarrow F(\boldsymbol{x}; \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \left(\frac{y-\hat{y}}{2}\right) \cdot \boldsymbol{\Phi}(\boldsymbol{x})$
return $\boldsymbol{w}$

Figure 1.2: Binary perceptron algorithm.

the perceptron draws a training example $(\boldsymbol{x}, y)$ and performs two actions: a prediction $\hat{y}$ is obtained by applying the current model $\boldsymbol{w}$ and, in case of a prediction error, the model is updated towards the misclassified example. Observe that, for positive examples ($y = +1$), the misclassified example features $\boldsymbol{\Phi}(\boldsymbol{x})$ are summed to the model parameters. And, for negative examples ($y = -1$), the features are subtracted from the model parameters. If the predicted output is correct ($\hat{y} = y$), the model $\boldsymbol{w}$ is not updated. This algorithm is proved to converge to a zero-risk solution, if one exists (Novikoff, 1962). Other training algorithms like stochastic gradient descent or support vector machines can also be employed with similar performance guarantees.

Weston and Watkins (1998) propose a generalization of the binary linear discriminant approach for multiclass classification problems. Their approach learns K linear discriminant functions, one for each class. Hence, for each class $k \in \{1, \ldots, K\}$, it learns a parameter vector $\boldsymbol{w}_k$ of a linear discriminant function given by

$$s(\boldsymbol{x}; \boldsymbol{w}_k) = \langle \boldsymbol{w}_k, \boldsymbol{\Phi}(\boldsymbol{x}) \rangle. \tag{1-4}$$

Up to this point, this approach is very similar to the one-vs-all approach cited earlier. However, it differs on the training strategy and also on the prediction function that is given by

$$F(\boldsymbol{x}; \boldsymbol{w}) = \arg\max_{y \in \mathcal{Y}} s(\boldsymbol{x}; \boldsymbol{w}_y), \tag{1-5}$$

where $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K)$ is the concatenation of the parameter vectors for all classes and $\mathcal{Y} = \{1, \ldots, K\}$ is the set of classes. As one can notice, the K discriminant functions are *jointly* employed to perform a prediction.

There are different training algorithms that jointly estimate the discriminant parameters for this multiclass model. Weston and Watkins (1998)

propose an SVM-based formulation by designing joint constraints for each training example, and Crammer and Singer (2001) propose an alternative formulation in order to derive an efficient algorithm for the resulting learning problem. Later, Crammer and Singer (2003) introduce a family of algorithms, called ultraconservatives, for training this kind of joint multiclass model with proven error bounds. In this same work, Crammer and Singer further propose the *margin infused relaxed algorithm* (MIRA), a ultraconservative algorithm that incorporates a generalized notion of margin for multiclass problems. The *multiclass perceptron* is a generalization of the binary perceptron and is a member of the ultraconservative family. We present the pseudo-code of this algorithm in Figure 1.3. As its binary counterpart, the multiclass

$\boldsymbol{w} \leftarrow \mathbf{0}$
while no convergence
  for each $(\boldsymbol{x}, y) \in \mathcal{D}$
    $\hat{y} \leftarrow \arg\max_{y' \in \mathcal{Y}} \langle \boldsymbol{w}_{y'}, \boldsymbol{\Phi}(\boldsymbol{x}) \rangle$
    $\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y + \boldsymbol{\Phi}(\boldsymbol{x})$
    $\boldsymbol{w}_{\hat{y}} \leftarrow \boldsymbol{w}_{\hat{y}} - \boldsymbol{\Phi}(\boldsymbol{x})$
return $\boldsymbol{w}$

Figure 1.3: Multiclass perceptron algorithm.

perceptron is an online algorithm that, on each iteration, picks a training example $(\boldsymbol{x}, y)$ and performs two main steps. First, a prediction $\hat{y}$ is obtained by applying the multiclass prediction function (1-5) using the current joint model $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K)$. Then, if the predicted class $\hat{y}$ is not the correct one y, the joint model $\boldsymbol{w}$ is updated as follows. The correct class model $\boldsymbol{w}_y$ is incremented by the input feature vector $\boldsymbol{\Phi}(\boldsymbol{x})$ and the predicted class model $\boldsymbol{w}_{\hat{y}}$ is decremented by this vector. For all other classes, their models are not updated. This update rule benefits the correct class in detriment of the predicted one in the next predictions regarding the current input features. If the predicted class is correct, the correct and predicted updates cancel each other and actually no update is performed.

Collins (2002b) further generalizes the multiclass perceptron for sequence labeling problems, like POS tagging. Following Weston and Watkins (1998) and Collins (2002b), Altun et al. (2003) propose an SVM-based formulation for sequence labeling and Joachims (2003) develops a related approach for sequence alignment. McDonald et al. (2005) further extend these methods and propose MSTParser, a system that uses a generalization of MIRA to train a dependency parser. Its training algorithm learns a linear discriminant function over head-modifier edges. Given an input sentence $\boldsymbol{x}$ and an edge (i, j) that

corresponds to a candidate dependency between the words x_i and x_j in $\boldsymbol{x}$, they define the following discriminant function

$$s(\boldsymbol{x}, i, j; \boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle, \tag{1-6}$$

where $\boldsymbol{\Phi}(\boldsymbol{x}, i, j)$ is a feature vector that describes the given dependency edge. The prediction function is then given by

$$F(\boldsymbol{x}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}} s(\boldsymbol{x}, i, j; \boldsymbol{w}), \tag{1-7}$$

where $\mathcal{Y}(\boldsymbol{x})$ is the set of all rooted trees over the words in $\boldsymbol{x}$, i.e., all possible dependency trees for the input sentence. This optimization problem is thus to find a rooted tree such that the sum of its edges scores is maximum. This problem corresponds to the maximum branching problem that is efficiently solved by Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967).

Fernandes and Milidiú (2012) use a generalization of the multiclass perceptron to estimate a parameter vector that minimizes the empirical risk of this DP model. In Figure 1.4, we present the pseudo-code for this algorithm. Again, it is an online algorithm that, at each iteration, performs

$\boldsymbol{w} \leftarrow \mathbf{0}$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{(i,j) \in \boldsymbol{y}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j) - \sum_{(i,j) \in \hat{\boldsymbol{y}}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j)$
return $\boldsymbol{w}$

Figure 1.4: Generalized perceptron algorithm for dependency parsing.

two main actions regarding a training instance $(\boldsymbol{x}, \boldsymbol{y})$: prediction and model update. The predicted tree $\hat{\boldsymbol{y}}$ is obtained by solving (1-7), which is done by Chu-Liu-Edmonds algorithm. The update rule is similar to the one used in the multiclass perceptron. Features present in the edges within $\boldsymbol{y}$ have their weights incremented, and weights for features within $\hat{\boldsymbol{y}}$ are decremented. Edges that are present in both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are canceled, thus their weights are not updated. Edges not present in either structures are also ignored.

Eventually, it has been realized that these task-specific methods are just instances of a general *structure learning framework*. In this framework, for a given input $\boldsymbol{x}$, the prediction problem is formulated as the following $\boldsymbol{w}$-parameterized optimization problem

$$F(\boldsymbol{x}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}), \tag{1-8}$$

where $\mathcal{Y}(\boldsymbol{x})$ is the set of possible predictions for the given input $\boldsymbol{x}$; and $s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$ is a $\boldsymbol{w}$-parameterized function that jointly scores an input-output pair $(\boldsymbol{x}, \boldsymbol{y})$. Intuitively, the scoring function measures how well the output structure fits the input. And, it is given by the following linear discriminant function

$$s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle, \tag{1-9}$$

where $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$ is an arbitrary joint feature vector representation of the input-output pair. Thus, for a given input, the prediction problem is to find the output with the highest score; where an output score is given by a discriminant function that is linear on some joint feature representation.

The structure learning framework *directly* solves structured problems in a general and principled way. The *structure perceptron* (SPerc) is a general training algorithm that follows the ERM principle to estimate the required parameter vector $\boldsymbol{w}$. This algorithm is presented in Figure 10.4. It is easy

$\boldsymbol{w} \leftarrow \mathbf{0}$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}') \rangle$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{\Phi}(\boldsymbol{x}, \hat{\boldsymbol{y}})$
return $\boldsymbol{w}$

Figure 1.5: Structure perceptron algorithm.

to show that the algorithms from Figures 1.2, 1.3 and 1.4 are instances of the SPerc algorithm. Moreover, an extension of Novikoff's theorem (Novikoff, 1962) proves that this algorithm converges to a zero-risk solution, if one exists.

Collins (2002b) proposes the structure perceptron algorithm with an averaging strategy. This is a known strategy used even with the binary perceptron and turns the algorithm more robust. In Figure 1.6, we present the pseudo-code of the averaged SPerc. It is very similar to the algorithm

$\boldsymbol{w}^0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \langle \boldsymbol{w}^t, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}') \rangle$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{\Phi}(\boldsymbol{x}, \hat{\boldsymbol{y}})$
    $t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}^k$

Figure 1.6: Averaged structure perceptron algorithm.

presented in Figure 10.4. Given that the algorithm executes for T iterations, the

averaged SPerc builds a sequence of models $\boldsymbol{w}^0, \ldots, \boldsymbol{w}^T$. Instead of returning the last model $\boldsymbol{w}^T$, like the ordinary SPerc, it returns the average among all built models, that is, $\boldsymbol{w} = \frac{1}{T}\sum_{k=1}^{T} \boldsymbol{w}^k$. Each update in the SPerc algorithm has potentially a big impact on the model parameters. Thus, the averaged algorithm is more robust to noisy examples, and usually performs significantly better than the non-averaged version.

The power of this framework relies mainly on the freedom to design the feature representation and the prediction problem. Specific feature representations allow the design of adherent models for complex SL problems. The decomposition of the joint feature vector along the output structures gives rise to meaningful prediction problems that frequently are reduced to well studied optimization problems. For instance, Collins (2002b) and Altun et al. (2003) model sequence labeling problems through the SL framework, and the resulting prediction problems are solved by the well known Viterbi algorithm; Joachims (2003) develops an approach to learn sequence alignment score functions that are then plugged in the Smith-Waterman alignment algorithm (Smith and Waterman, 1981); and Altun et al. (2007) create a constituent parser whose prediction is performed by the CKY parsing algorithm. This natural fitting between prediction problems and task-meaningful algorithms is no coincidence. It is a consequence of the fact that the SL framework is general yet very flexible.

1.3 Nonlinearity

Linear models are pervasive in ML, mainly because there are efficient training algorithms with proven error bounds to estimate such models. On the other hand, many SL problems are highly non-linear on the available input features. Therefore, when training a linear model within the SL framework, it is necessary to use some feature generation method in order to provide the required nonlinear feature combinations.

Feature generation is frequently solved by a domain expert that generates complex and discriminative feature templates by conjoining input features. Manual template generation is a limited and expensive way to obtain feature templates and is recognized as a modeling bottleneck. Another popular alternative is to employ a kernel function, if the learning algorithm allows it. Besides the fact that kernelized training algorithms are computationally expensive, it is difficult to control the generalization performance of the learned models.

1.4 Entropy-Guided Structure Learning

In this work, we use *Entropy-guided Feature Generation* (EFG), an automatic method to generate feature templates. EFG is based on the conditional entropy of local prediction variables given basic features. It receives a training dataset with basic features and produces a set of feature templates by conjoining features that together are highly discriminative. EFG is based on the same strategy of Entropy-guided Transformation Learning (ETL) (Milidiú et al., 2008; dos Santos and Milidiú, 2009a), which generalizes Transformation-Based Learning (Brill, 1995) by automatically generating rule templates.

We introduce EFG in Fernandes and Milidiú (2012), where we apply it to Portuguese dependency parsing and show that it obtains higher performance than the best available manual templates, when the same basic features are given to both systems. In the Conference on Computational Natural Language Learning (CoNLL) 2012 Shared Task (Pradhan et al., 2012), we propose another EFG-based system that achieves the very first place in this competition (Fernandes et al., 2012b). Here, we experimentally compare EFG to manual template generation and kernel methods on three tasks. In Table 1.1, we summarize these results. Observe that EFG outperforms both alternative

	Alternative System		**EFG**	
Task	**Method**	$\mathbf{F_1}$	$\mathbf{F_1}$	**Error Reduction**
Portuguese DP	Manual Templates	90.06	90.28	2.2%
English Chunking	Kernel	93.48	94.12	9.8%
Portuguese Chunking	Kernel	86.67	87.72	7.9%

Table 1.1: Comparison of EFG with other feature generation methods.

methods on the evaluated datasets. Additionally, it is much cheaper than manual templates and computationally faster than kernel methods.

EFG is easily integrated into the general structure learning framework. We extend this framework by including EFG as a preprocessing step. We denote this extension *Entropy-guided Structure Learning* (ESL) framework. ESL is not restricted to natural language processing tasks. In Figure 1.7, we present the pseudo-code of the extended framework. The training dataset $\mathcal{D}$ is given with the available basic features. Before running the learning algorithm, we apply EFG to generate non-linear features that compose the feature vectors $\mathbf{\Phi}(\mathcal{D}) = \{\mathbf{\Phi}(\boldsymbol{x}, \boldsymbol{y})\}_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}}$ given as input for the learning algorithm.

We evaluate the entropy-guided structure learning framework on nine datasets involving five NLP tasks and four languages. In Table 1.2, we depict the performances achieved by ESL systems compared with the best known

$\boldsymbol{\Phi}(\mathcal{D}) \leftarrow \texttt{EFG}(\mathcal{D})$
$\boldsymbol{w}^0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \langle \boldsymbol{w}^t, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}') \rangle$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{\Phi}(\boldsymbol{x}, \hat{\boldsymbol{y}})$
    $t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}^k$

Figure 1.7: Entropy-Guided Structure Learning framework.

results for each task. ESL reduces the smallest known error for some tasks and

Task	Language	State of the Art	ESL	
			Accuracy	Error Reduction
POS Tagging	Portuguese	96.94	**97.12**	5.9%
POS Tagging	English	**96.83**	96.72	−3.5%
Text Chunking	Portuguese	87.46	**87.72**	2.1%
Text Chunking	English	**94.21**	94.12	−1.6%
Dependency Parsing	Portuguese	**93.03**	92.66	−5.3%
Quotation Extraction	Portuguese	71.26	**76.80**	19.3%
Coreference Resolution	Arabic	53.55	**54.22**	1.4%
Coreference Resolution	Chinese	62.24	**62.87**	1.7%
Coreference Resolution	English	61.31	**63.37**	5.3%
Coreference Resolution	Multilingual	58.25	**60.15**	4.5%

Table 1.2: Comparison of ESL with state-of-the-art systems.

achieves state-of-the-art comparable results for others. These are remarkable results, specially considering that ESL is relatively simple to be applied on such complex tasks. Moreover, many of the systems that outperform ESL include additional information that could be also included in our systems.

1.5 Contributions

This work has seven main contributions:

1. the ESL framework;
2. a comparison of EFG with manual templates and polynomial kernels;
3. nine ESL based systems for fundamental NLP tasks;
4. state-of-the-art systems for three Portuguese tasks;
5. a novel SL modeling of coreference resolution based on latent trees;

6. ESL-based systems that achieve the best performance on the renowned CoNLL-2012 Shared Task on multilingual coreference resolution;

7. state-of-the-art systems for coreference resolution on Arabic, Chinese and English.

Partial results related to this work have been previously presented in eight important scientific events (Fernandes et al., 2009a,b; dos Santos et al., 2010; Fernandes et al., 2010a,b; Fernandes and Brefeld, 2011; Fernandes and Milidiú, 2012; Fernandes et al., 2012b) and one journal paper (Fernandes et al., 2010c).

Our key contribution is the entropy-guided structure learning framework that extends the general linear discriminative SL framework. ESL employs the entropy-guided feature generation method to solve the problem of non-linear feature generation, a known modeling bottleneck. This framework provides a general machine learning approach that can be applied to supervised structure learning problems.

Our second main contribution is to compare the EFG method with two alternative methods for non-linear feature generation, namely manual templates and polynomial kernel methods. We demonstrate that EFG is superior to both alternatives, since it achieves higher performance. Moreover, it is cheaper than manual templates, faster than kernel methods and avoids the overfitting issue shown by the latter. EFG outperforms the best available manual template set for dependency parsing on the Portuguese dataset provided in the CoNLL-2006 Shared Task. We also compare EFG with polynomial kernel methods for Portuguese and English text chunking. EFG outperforms both methods.

Our third main contribution is the ESL framework instantiation on five NLP tasks and the assessment of the resulting systems by comparing their performances with the best known results on nine datasets involving four languages. These five tasks involve four different structured outputs, namely: sequence, segmentation, rooted tree and clustering. The developed ESL systems present state-of-the-art comparable performances on all evaluated datasets.

Our fourth main contribution is to provide three ESL-based systems that outperform the best previous systems on three Portuguese tasks. On Mac-Morpho, a POS tagging dataset, our system reduces the previous smallest error by 5.9%. On Bosque, a text chunking dataset, the proposed ESL system reduces the previous smallest error by 2.1%. For quotation extraction, our

system reduces the previous smallest error on the GloboQuotes dataset by 19.3%.

Our fifth main contribution is a novel coreference resolution modeling which employs a latent structure in order to control the complexity of the prediction problem. Coreference resolution is a clustering problem and most objective functions for such problems lead to NP-hard optimization problems. In order to design a prediction problem that can be efficiently solved, we represent a coreference cluster as a directed tree that is intuitively adherent to the coreference task.

By employing this latent modeling along with the ESL framework, we achieve the very first place on the renowned CoNLL 2012 Shared Task. The competing systems are ranked by the mean score over three languages: Arabic, Chinese and English. Our ESL systems present an error that is 1.1% smaller than the runner-up competitor on this multilingual task. By further improving our Chinese system, we achieve a 4.5% error reduction over the runner-up system. This relevant achievement constitutes our sixth main contribution.

Our coreference resolution systems achieve the best known performance on the three languages considered in the CoNLL-2012 Shared Task. Our systems reduce the smaller known error by 1.4%, 1.7%, and 5.3%, respectively, on Arabic, Chinese and English. These state-of-the-art systems for multilingual unrestricted coreference resolution comprise our seventh main contribution.

1.6 Dissertation Organization

We use the dependency parsing task as an illustrative application of the main concepts underlying our work. In Chapter 2, we detail the application of the structure learning framework for this task. In this chapter, we also introduce two important extensions to the basic SL framework, namely large margin training and latent structures. In Chapter 3, we describe the proposed entropy-guided feature generation method for structure learning problems. Again, we use DP to illustrate EFG application. We also present in this chapter a comparison of the EFG method to manual templates and kernel methods under the same experimental conditions. In Chapter 4, we detail ESL, the extension of the structure learning framework by incorporating EFG. We also show that the DP system presented in the previous chapters is an instance of this general framework. One key component of ESL is the prediction problem. We discuss some important aspects related to prediction problems in Chapter 5. In this chapter, we also present important examples of prediction problems. Our dependency parsing system is summarized in

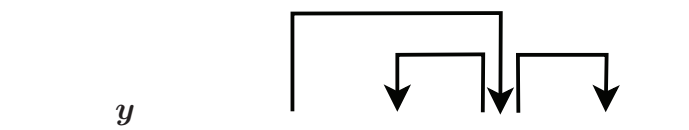
Chapter 6. We apply our framework to two part-of-speech tagging datasets. In Chapter 7, we detail these two applications and their experimental results. In Chapter 8, we introduce the application of ESL to two text chunking datasets and the corresponding experimental results. In Chapter 9, we present an ESL-based system to quotation extraction and report on some experiments with a Portuguese dataset. As mentioned before, we achieved the first place in the CoNLL-2012 Shared Task, which was dedicated to multilingual coreference resolution. In Chapter 10, we describe our ESL modeling for this task and the achieved results. This system further extends the ESL framework by introducing latent structures, which are also described. Finally, in Chapter 11, we present our concluding remarks.

2
Structure Learning Framework

In this chapter, we describe the structure learning framework and some known extensions by detailing its application to the dependency parsing task. In Section 2.1, we formalize this task. Then, in Section 2.2, we present the *large margin* structure perceptron, an important extension to the SPerc algorithm. A powerful extension to the SL framework allows the introduction of *latent structures* that are optionally used as auxiliary information to the target task. We present the latent structure perceptron in Section 2.3. Finally, in Section 2.4, we depict empirical results of the structure perceptron on a Portuguese DP dataset, comparing its results with other state-of-the-art systems.

2.1
Dependency Parsing

Dependency parsing is to identify the structure underlying a sentence that describes the syntactic dependencies among its words. This structure is called *dependency tree* and is a rooted tree whose nodes are the words and punctuation marks in the sentence, i.e., the sentence tokens. Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_N)$ be a sentence, where x_i is the i-th token and x_0 is an artificial token which is always the root of the dependency tree. For a given input sentence $\boldsymbol{x}$, the prediction output space $\mathcal{Y}(\boldsymbol{x})$ is the set of all rooted trees whose nodes are the tokens in $\boldsymbol{x}$ and the root node is x_0. For any dependency tree $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$, we say that $(i, j) \in \boldsymbol{y}$ whenever token x_j modifies token x_i in the tree $\boldsymbol{y}$. The token x_i is called *head token* and the token x_j is called *modifier token*. Figure 2.1 shows the sentence John saw Mary, its dependency tree $\boldsymbol{y} = \{(0, 2), (2, 1), (2, 3)\}$, and the corresponding head tokens. The head



i	0	1	2	3
$\boldsymbol{x}$	*root*	John	saw	Mary
Head $(y(i))$	–	2	0	2

Figure 2.1: Dependency tree represented as arcs ($\boldsymbol{y}$) and as head vector.

vector is an alternative representation for the dependency tree, since each modifier token has exactly one head given by an incoming edge; except the artificial root token which has no head. We denote by $y(i)$ the head token of the modifier token x_i within the tree $\boldsymbol{y}$, that is $(y(i), i) \in \boldsymbol{y}$, for $i = 1, \ldots, N$.

MSTParser (McDonald et al., 2005, 2006; McDonald and Pereira, 2006) is a SL system that employs an online algorithm to train a dependency parser. MSTParser's prediction problem is formulated as a linear discriminative problem. MSTParser learns a $\boldsymbol{w}$-parameterized edge scoring function $s(\boldsymbol{x}, i, j; \boldsymbol{w})$ for every candidate edge (i, j) over $\boldsymbol{x}$, such that the prediction problem is to find the maximum-scoring rooted tree, that is

$$F(\boldsymbol{x}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}} s(\boldsymbol{x}, i, j; \boldsymbol{w}). \tag{2-1}$$

This is equivalent to the well studied maximum branching problem that can be efficiently solved by Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). There is also an improved version of this algorithm by Tarjan (1977).

Obviously, the edge scoring function is key for MSTParser and has to generalize over any possible input sentence. For a sentence $\boldsymbol{x}$ and a candidate edge (i, j), MSTParser determines $s(\boldsymbol{x}, i, j; \boldsymbol{w})$ by means of M real-valued features denoted by $\phi_m(\boldsymbol{x}, i, j)$, for $m = 1, \ldots, M$. These features describe the dependency between the head token x_i and the modifier token x_j in a meaningful and general way. MSTParser uses several *binary* features like, for instance, the i-th word, the j-th word, the part-of-speech of the i-th word, the distance from x_i to x_j, the relative order between x_i and x_j, among others. Then, the edge scoring function is defined as an weighted sum of the edge features

$$s(\boldsymbol{x}, i, j; \boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle,$$

where $\boldsymbol{w} = (w_1, \ldots, w_M)$ comprises the model parameters, one for each feature; and $\boldsymbol{\Phi}(\boldsymbol{x}, i, j) = (\phi_1(\boldsymbol{x}, i, j), \ldots, \phi_M(\boldsymbol{x}, i, j))$ is the feature vector that describes the dependency edge (i, j). In that way, the learning problem is to determine a parameter vector $\boldsymbol{w}$ such that the predictor $F(\boldsymbol{x}; \boldsymbol{w})$ has small empirical risk on the training data and, at the same time, performs well on unseen data.

MSTParser's training algorithm is an extension of the margin infused relaxed algorithm, an online algorithm for multiclass problems. In this work, we use the averaged structure perceptron algorithm (Collins, 2002b) with a large margin extension (Fernandes and Brefeld, 2011; McAllester et al., 2011; Tsochantaridis et al., 2005) as the training algorithm for all tasks. We use SPerc even for DP, despite the fact that we use MSTParser's modeling for this

task.

2.2 Large Margin Training

The structure perceptron algorithm finds a classifier with no concern about its margin. However, it is well known that large margin classifiers provide better generalization performance on unseen data. We use a large-margin generalization of the structure perceptron that is based on the margin rescaling technique for SVM^{struct} (Altun et al., 2003; Tsochantaridis et al., 2005). During *training*, for an example $(\boldsymbol{x}, \boldsymbol{y})$, instead of the ordinary prediction problem (2-1), we use the following *loss-augmented* prediction function

$$F_\ell(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \left[\left(\sum_{(i,j) \in \boldsymbol{y}'} s(\boldsymbol{x}, i, j; \boldsymbol{w}) \right) + C \cdot \ell(\boldsymbol{y}, \boldsymbol{y}') \right], \qquad (2\text{-}2)$$

where $\ell(\boldsymbol{y}, \boldsymbol{y}') \geq 0$ is an appropriate loss function that measures the difference between a candidate tree $\boldsymbol{y}'$ and the correct one $\boldsymbol{y}$; and C is a constant to balance the weight between the loss function (margin) and the learned edge weights. We use a loss function that just counts how many incorrect edges are in the predicted tree $\boldsymbol{y}'$, which is given by

$$\ell(\boldsymbol{y}, \boldsymbol{y}') = \sum_{(i,j) \in \boldsymbol{y}'} \mathbf{1}[(i,j) \notin \boldsymbol{y}]. \qquad (2\text{-}3)$$

This loss function can be decomposed along the tree edges and we can thus rewrite the loss-augmented prediction function as

$$F_\ell(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} \left(s(\boldsymbol{x}, i, j; \boldsymbol{w}) + C \cdot \mathbf{1}[(i,j) \notin \boldsymbol{y}] \right). \qquad (2\text{-}4)$$

This characteristic is desirable because we can define a loss-augmented *edge scoring* function as

$$s_\ell(\boldsymbol{x}, \boldsymbol{y}, i, j; \boldsymbol{w}) = s(\boldsymbol{x}, i, j; \boldsymbol{w}) + C \cdot \mathbf{1}[(i,j) \notin \boldsymbol{y}],$$

and then we have that

$$F_\ell(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} s_\ell(\boldsymbol{x}, \boldsymbol{y}, i, j; \boldsymbol{w}),$$

which is a maximum branching problem just as the original prediction problem, but with modified edge weights. In that way, we can still use Chu-Liu-Edmonds algorithm in the large margin structure perceptron. We present the pseudo-code of the large-margin structure perceptron algorithm

in Figure 2.2. The unique modification to the SPerc algorithm is in the

$\boldsymbol{w}^0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \sum_{(i,j) \in \boldsymbol{y}'} \left(\langle \boldsymbol{w}^t, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle + C \cdot \mathbf{1}[(i,j) \notin \boldsymbol{y}] \right)$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \sum_{(i,j) \in \boldsymbol{y}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j) - \sum_{(i,j) \in \hat{\boldsymbol{y}}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j)$
    $t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}^k$

Figure 2.2: Large margin structure perceptron algorithm for dependency parsing.

computation of edge scores, where we add a constant C to the score of every incorrect edge. The constant C is a meta-parameter of this algorithm that allows us to balance the relative importance of the two components in the objective function. This parameter can be calibrated by means of cross-validation or a development set.

By using the loss-augmented prediction problem during training, an example $(\boldsymbol{x}, \boldsymbol{y})$ implies a model update whenever the current model does not respect the following margin constraint

$$s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) - s(\boldsymbol{x}, \boldsymbol{y}'; \boldsymbol{w}) \geq C \cdot \ell(\boldsymbol{y}, \boldsymbol{y}'), \ \forall \boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x}),$$

where $s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \sum_{(i,j) \in \boldsymbol{y}} s(\boldsymbol{x}, i, j; \boldsymbol{w})$ is the score of a whole tree. If a model respects this prediction margin, then the current predictor $F(\boldsymbol{x}; \boldsymbol{w})$ separates the correct output $\boldsymbol{y}$ from every alternative $\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})$ by a margin as large as $C \cdot \ell(\boldsymbol{y}, \boldsymbol{y}')$. In that way, the training algorithm incorporates *some* information about the structured empirical risk $\mathcal{R}_\ell(\mathcal{D}, \boldsymbol{w})$ of the current model, defined as

$$\mathcal{R}_\ell(\mathcal{D}, \boldsymbol{w}) = \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}} \ell(\boldsymbol{y}, F(\boldsymbol{x}; \boldsymbol{w})).$$

2.3 Latent Structure Training

For some structure learning problems, to directly predict an output $\boldsymbol{y}$ from the input $\boldsymbol{x}$ is a very hard problem, considering any meaningful feature representation. In some cases, if an appropriate auxiliary structure $\boldsymbol{h}$ is given for the input $\boldsymbol{x}$, the prediction problem becomes much easier. In coreference resolution, for instance, the input comprises a document containing a set of mentions to real world entities, like companies, places or people. The prediction

problem is then to cluster mentions that correspond to the same entity. Most clustering metrics lead to NP-hard optimization problems. Thus, we introduce coreference trees to represent a cluster of mentions. Giving such tree for an input document turns prediction into a polynomial problem.

Usually, the auxiliary structures are not explicitly given in the training data. Thus, we assume that these structures are *latent* and make use of the latent structure perceptron (Fernandes and Brefeld, 2011; Yu and Joachims, 2009). The original prediction problem is then split into two sub-problems: the *latent* prediction problem $F_h(\boldsymbol{x}; \boldsymbol{w}_h)$ and the *target* prediction problem $F_y(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{w}_y)$, where $\boldsymbol{w}_h$ is the latent model and $\boldsymbol{w}_y$ is the target model. In that way, the final prediction is performed by combining these two predictors

$$F(\boldsymbol{x}; \boldsymbol{w}_h, \boldsymbol{w}_y) = F_y(\boldsymbol{x}, F_h(\boldsymbol{x}; \boldsymbol{w}_h); \boldsymbol{w}_y).$$

Both the latent and the target predictors have the same form. Not surprisingly they are based on linear discriminative functions. The latent prediction function is given by

$$F_h(\boldsymbol{x}; \boldsymbol{w}_h) = \arg \max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x})} \langle \boldsymbol{w}_h, \boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h}) \rangle,$$

where $\mathcal{H}(\boldsymbol{x})$ is the set of feasible latent structures for the given input $\boldsymbol{x}$ and $\boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h})$ is an arbitrary joint feature vector representation of the input and the latent structure. The target prediction function is then defined as

$$F_y(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{w}_y) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}, \boldsymbol{h})} \langle \boldsymbol{w}_y, \boldsymbol{\Phi}_y(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y}) \rangle,$$

where $\mathcal{Y}(\boldsymbol{x}, \boldsymbol{h})$ is the set of feasible output structures for the input $\boldsymbol{x}$ *and* the latent structure $\boldsymbol{h}$; and, $\boldsymbol{\Phi}_y(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y})$ is an arbitrary joint feature vector representation of the input, the latent structure and the output structure.

In Figure 2.3, we depict the latent structure perceptron algorithm. The latent structure perceptron is very similar to the original SPerc, which, for each training instance, performs two major steps: (i) a prediction for the given input using the current model; and (ii) a model update based on the difference between the predicted and the ground truth outputs. The latent SPerc performs an additional step to predict the latent ground truth $\tilde{\boldsymbol{h}}$ using a specialization of the latent predictor.

Golden latent structures are usually not available in the training data. However, during training, for a given input $\boldsymbol{x}$, we have the golden output structure $\boldsymbol{y}$. Thus, we predict the *constrained latent structure* $\tilde{\boldsymbol{h}}$ for the training instance $(\boldsymbol{x}, \boldsymbol{y})$ using a specialization of the latent predictor – the *constrained*

$\boldsymbol{w}_0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\tilde{\boldsymbol{h}} \leftarrow \arg\max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x},\boldsymbol{y})} \langle \boldsymbol{w}_h^t, \boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h}) \rangle$
    $\hat{\boldsymbol{h}} \leftarrow \arg\max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x})} \langle \boldsymbol{w}_h^t, \boldsymbol{\Phi}_h(\boldsymbol{x}, \boldsymbol{h}) \rangle + \ell(\tilde{\boldsymbol{h}}, \boldsymbol{h})$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x},\hat{\boldsymbol{h}})} \langle \boldsymbol{w}_y^t, \boldsymbol{\Phi}_y(\boldsymbol{x}, \hat{\boldsymbol{h}}, \boldsymbol{y}') \rangle + \ell(\boldsymbol{y}, \boldsymbol{y}')$
    $\boldsymbol{w}_h^{t+1} \leftarrow \boldsymbol{w}_h^t + \boldsymbol{\Phi}_h(\boldsymbol{x}, \tilde{\boldsymbol{h}}) - \boldsymbol{\Phi}_h(\boldsymbol{x}, \hat{\boldsymbol{h}})$
    $\boldsymbol{w}_y^{t+1} \leftarrow \boldsymbol{w}_y^t + \boldsymbol{\Phi}_y(\boldsymbol{x}, \tilde{\boldsymbol{h}}, \boldsymbol{y}) - \boldsymbol{\Phi}_y(\boldsymbol{x}, \hat{\boldsymbol{h}}, \hat{\boldsymbol{y}})$
    $t \leftarrow t + 1$
$\boldsymbol{w}_h \leftarrow \frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}_h^k$
$\boldsymbol{w}_y \leftarrow \frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}_y^k$
return $(\boldsymbol{w}_h, \boldsymbol{w}_y)$

Figure 2.3: Latent structure perceptron algorithm.

latent predictor – that makes use of $\boldsymbol{y}$. The constrained predictor finds the maximum scoring latent structure among all structures that follow the correct output $\boldsymbol{y}$. That is, the constrained set $\mathcal{H}(\boldsymbol{x}, \boldsymbol{y}) \subset \mathcal{H}(\boldsymbol{x})$ does not include latent structures that lead to incorrect output structures $\boldsymbol{y}' \neq \boldsymbol{y}$. The constrained latent structure $\tilde{\boldsymbol{h}}$ is then used as the ground truth for the current iteration. Therefore, the model update is determined by the difference between the constrained structure latent and the document tree predicted by the ordinary predictor.

The latent structure perceptron algorithm learns to predict latent structures that help to solve the target task. This algorithm is an instance of the Concave-Convex Procedure and converges to a local optimum (Yuille and Rangarajan, 2003; Yu and Joachims, 2009).

2.4 Empirical Results

Recently, dependency parsing has attracted much attention, and fast progress has been made on pushing the performance of dependency parsers. In 2005, McDonald et al. (2005) proposes the MSTParser system that achieves state-of-the-art performance on different datasets. In the next year, the CoNLL-2006 Shared Task (Buchholz and Marsi, 2006) is devoted to multilingual dependency parsing, and McDonald et al. (2006) achieves the best performances by applying an extension of MSTParser that uses *second-order features*. MSTParser's original features are based on individual edges. The second-order features depend on two edges, which link a head token to two *sibling* modifiers. Since this model considers more complex dependencies

in the output structure, the corresponding prediction problem is also more complex. In fact, the prediction problem in this case is NP-Hard (McDonald and Pereira, 2006). McDonald and Pereira (2006) propose an approximation algorithm to this problem and show that the second-order model outperform the first-order one, even using approximated prediction. In Table 2.1, we show the performances of these two systems on the Portuguese CoNLL-2006 dataset. The performances are reported using the *unlabeled attachment score* (UAS).

Reference	**Learning Algorithm**	**Basic Features**	**UAS**
Koo et al. (2010)	MIRA	3rd order	93.03
		2nd order	92.57
McDonald et al. (2005, 2006); McDonald and Pereira (2006)	MIRA	2nd order	91.36
		1st order	90.68
This work	SPerc	1st order	90.06

Table 2.1: Performances of dependency parsers using manual templates on the Portuguese CoNLL-2006 dataset. These systems use different learning algorithms and also different basic features.

UAS is the percentage of tokens that are attached to the correct head or, equivalently, the percentage of correct edges in the predicted tree.

Nowadays, as far as we know, the best performing system on the Portuguese CoNLL-2006 dataset is the dual decomposition system proposed by Koo et al. (2010). In Table 2.1, we present the performances of this system. This system introduces a new algorithm to perform approximated prediction with second- and third-order features. However, the second-order features used in this system are slightly different from the ones used in MSTParser, as we can see from the achieved results. The third-order features include grandparent dependencies, in addition to the sibling dependencies given by second-order features. All these models are trained with MIRA and complex features are generated with the manual templates proposed by McDonald and Pereira (2006).

We also present in Table 2.1 the performance achieved by a first-order model trained by the SPerc algorithm with the same templates used in MSTParser. We notice that this system is outperformed by MSTParser, even when MSTParser is using first-order features. This difference is probably due to the training algorithm and also to some differences in the feature templates that are not completely described in the corresponding references.

3
Entropy-Guided Feature Generation

A task dataset includes features that usually are either (i) naturally included in the very phenomenon of interest, like words in NLP tasks; (ii) simply derived from other basic features, like word capitalization patterns; or (iii) automatically generated by external systems, like part-of-speech tags. We denote this dataset-provided information as *basic features*. At the same time, most structure learning algorithms are based on linear models, since such algorithms have strong theoretical guarantees regarding their prediction performance and, moreover, are computationally efficient. However, linear models on basic features alone do not capture all the relevant relationships among these variables.

For instance, dependency parsing is highly non-linear on the basic features. One important basic feature for DP is the POS of words. By observing the example in Figure 1.1, one can notice that both nouns system and result are modifiers of the verb achieves. This is a very strong pattern in DP and, thus, the binary feature `modifier is a noun AND head is a verb` is very important. This feature is a conjunction of two basic features, namely the modifier POS tag and the head POS tag. On the other hand, taking each of these basic features independently is not informative, since most POS tags can be head or modifier of many dependencies, depending on the context. For instance, the nouns system and results, that modify the main verb, are also heads of the pronoun Our and the adjective best, respectively.

Conjoining basic features to derive new features is a common way to introduce nonlinear contextual patterns into linear models. Frequently, a domain expert manually generates feature templates by conjoining the given basic features in order to capture discriminative contextual patterns. MSTParser, for instance, uses 21 feature templates that were manually created from basic features. These templates include from one to six features, indicating that the trained model is highly non-linear on the basic input features.

In this chapter, we describe the proposed entropy-guided feature generation method for structure learning. EFG automatically derives a set of basic feature conjunctions, which we denote *feature templates*. These templates

are later used to generate the *derived features*, which comprise the joint feature vectors $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$ used in the structured modeling described earlier. EFG conjoins basic features that are useful to predict *local* variables.

3.1 Basic Dataset

As seen in the previous chapters, in dependency parsing, features are functions of dependency tree edges. Given an edge $e = (i, j)$ linking token x_i to token x_j, let us examine only its *categorical* basic features. Assume there are K basic features given by the vector $\boldsymbol{\Psi}(e) = (\psi_1(e), \ldots, \psi_K(e))$. For $k = 1, \ldots, K$, we have that $\psi_k(e) \in X_k$, where X_k is the finite set of possible values for the basic feature ψ_k. Additionally, we associate each edge $e = (i, j)$ with a binary label $y(e)$, such that $y(e) = 1$ if x_i is the head of x_j and $y(e) = 0$ otherwise.

Using all edges in the training set $\mathcal{D}$, we obtain the *basic dataset* $D = \{(\boldsymbol{\Psi}(e), y(e))\}$ comprising the basic feature vectors of edges along with their binary labels. In Table 3.1, we depict an example of such a dataset for the sentence in Figure 2.1. This example includes the following basic features: head-word is the word of the head token x_i; mod-word is the word of the modifier token x_j; head-pos is the POS tag of x_i; mod-pos is the POS tag of x_j; dist is the distance between x_i and x_j in tokens; and side is the side of x_j in relation to x_i.

e		$\boldsymbol{\Psi}(e)$						$y(e)$
i	j	**head-word**	**mod-word**	**head-pos**	**mod-pos**	**dist**	**side**	
0	1	*root*	John	*root*	noun	1	right	0
0	2	*root*	saw	*root*	verb	2	right	1
0	3	*root*	Mary	*root*	noun	3	right	0
1	2	John	saw	noun	verb	1	right	0
1	3	John	Mary	noun	noun	2	right	0
2	1	saw	John	verb	noun	1	left	1
2	3	saw	Mary	verb	noun	1	right	1
3	2	Mary	saw	noun	verb	1	left	0

Table 3.1: Basic dataset for the sentence in Figure 2.1.

The entropy guided feature generation method automatically generates feature templates for a structure learning problem by conjoining basic features that are highly discriminative together. EFG is based on the conditional entropy of the local decision variables $y(e)$ given the basic features $\boldsymbol{\Psi}(e)$.

3.2 Conditional Entropy and Information Gain

Entropy is a measure of the uncertainty in a random variable outcome. Given a binary random variable y and the probability $Pr[y = 1] = p$, the

entropy of y is given by

$$H(y) = -p \log_2 p - (1-p) \log_2 (1-p),$$

where $0 \log_2(0)$ is defined to be equal to 0. In Figure 3.1, we plot the entropy $H(y)$ versus p. One can see that the maximum entropy is achieved when $p = 0.5$, that is, when the uncertainty on the outcome of y is maximum.

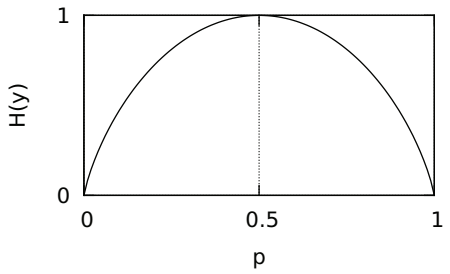


Figure 3.1: Entropy $H(y)$ of a random binary variable y versus $Pr[y = 1]$ that is denoted by p.

Given the basic dataset D, one can easily estimate p as the fraction of edges in which $y(e)$ is equal to 1. Then, the *empirical entropy* of y on D is denoted by $H_D(y)$ and can be calculated using the estimated probability of $y = 1$ on the given dataset. In the following, we simply use the term entropy to denote empirical entropy.

Considering the basic dataset D, we define the *conditional entropy* of y given the basic feature ψ_k as

$$H_D(y|\psi_k) = \sum_{\sigma \in X_k} \frac{|D_{k,\sigma}|}{|D|} \cdot H_{D_{k,\sigma}}(y),$$

where $D_{k,\sigma}$ is the subset of edges in D whose feature ψ_k is equal to σ, that is, $D_{k,\sigma} = \{(\mathbf{\Psi}(e), y(e)) \in D \mid \psi_k(e) = \sigma\}$. The conditional entropy $H_D(y|\psi_k)$ is the entropy of y on D when an additional information (feature ψ_k) is given. From Gibb's inequality, it follows that $H_D(y|\psi_k) \leq H_D(y)$, that is, the knowledge of any additional information can only reduce uncertainty.

Additionally, the *information gain* (IG) of ψ_k on D is given by

$$IG_D(\psi_k) = H_D(y) - H_D(y|\psi_k),$$

which corresponds to the reduction on the entropy of y if feature ψ_k is known. Hence, IG helps to select high discriminative features with respect to a target variable. It is straightforward to generalize information gain to

a subset of features. Thus, we have a valuable metric to measure nonlinear feature conjunctions and select the most informative ones.

Unfortunatelly, to analyse all possible feature conjunctions is practically infeasible and, moreover, to find the best conjunctions is an NP-complete problem (Hyafil and Rivest, 1976). On the other hand, decision tree (DT) learning provides a simple yet effective algorithm that generates different subsets of informative features, greedily guided by some informativeness metric. The most popular DT algorithms (Quinlan, 1992; Su and Zhang, 2006) use information gain as that metric. Therefore, we use decision tree induction to generate feature combinations that are highly discriminative together.

3.3 Decision Tree Learning

Decision tree learning is one of the most widely used machine learning algorithms. It performs a partitioning of the training set using principles of information theory. The learning algorithm executes a general to specific search of a feature space. The most informative feature is added to a tree structure at each step of the search. Information gain, which is based on the data entropy, is normally used as the informativeness measure. The objective is to construct a tree, using a minimal set of features, that efficiently partitions the training set into classes given by the prediction variable values. Usually, after the tree is grown, a pruning step is carried out in order to avoid overfitting. In Figure 3.2, we present a decision tree learned from a basic dataset. Each *internal* node in the DT corresponds to a feature, each *leaf* node has a label value (0 or 1, in the binary case), and each edge is labeled with a value of the source node feature.

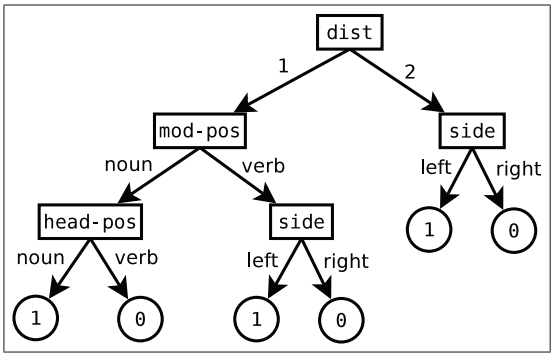


Figure 3.2: A decision tree.

The most popular decision tree learning algorithms (Quinlan, 1992; Su and Zhang, 2006) use information gain to select the most informative feature. Hence, they provide a quick way to obtain entropy guided feature selection.

We propose a new automatic feature generation method for structure learning algorithms. The key idea is to use decision tree induction to conjoin the basic features. One of the most used algorithms for DT induction is C4.5 (Quinlan, 1992). We use Quinlan's C4.5 system to obtain the required entropy guided selected features.

3.4 Feature Templates

The first step of the proposed method is to train a decision tree on the basic dataset. For dependency parsing, the decision variable indicates whether an edge links a token to its corresponding head token. We use the edges of all training examples, that is, for each training sentence, we generate an example for each candidate edge. Thus, the learned DT predicts whether an edge corresponds to a correct dependency or not.

Our method uses a very simple decomposition scheme to extract feature templates. This decomposition is based on a depth-first traversal of the learned DT and is recursively defined as follows. For each *internal* node that is visited, a new template is created by conjoining the node feature with its parent template. Since we aim to generate feature *templates* – conjunctions of basic features not including feature values – we disconsider the feature values and the decision variable values in the DT. Thus, we do not make use of edge labels nor leaf nodes. Figure 3.3 illustrates our method. The tree in the left side of this figure is the skeleton obtained from the decision tree in Figure 3.2 by discarding the aforementioned pieces of information. Then, it remains a tree whose nodes are basic features with high discriminative power. The generated templates are listed in the right side of the figure. In other words, we create a template with the features in each path from the root node to every other internal node in the given decision tree. Additionally, we eliminate template duplicates.

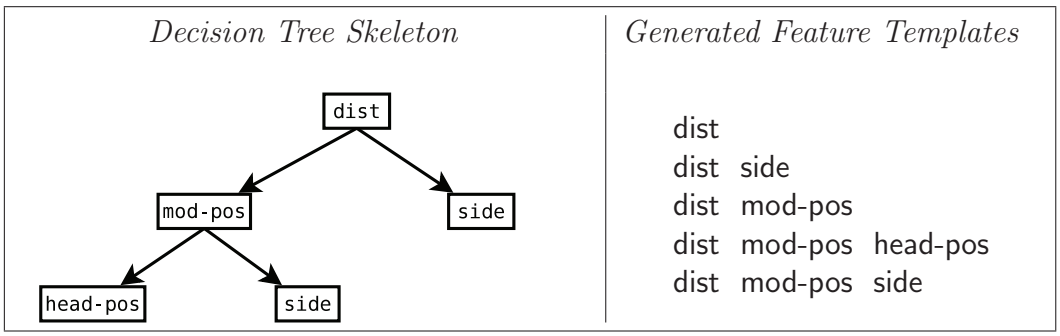


Figure 3.3: Feature template induction from a decision tree.

Since the DT learning algorithm greedily chooses the feature with the

highest information gain at each step, our method generates feature templates with high discriminative power based on entropy. This method is able to provide a very large number of templates. Hence, to limit the maximum template length, we use C4.5 pruned trees and additionally limit the maximum template length when traversing the DT. This parameter is clearly task dependent and must be calibrated by cross-validation or by means of a development set.

3.5 Generated Features

Finally, we employ the generated templates to induce all binary contextual features that occur in the structured dataset $\mathcal{D}$. For each template, we generate several binary features, each one corresponding to the assignment of valid values to the template features. These derived features comprise the structured model feature vectors $\mathbf{\Phi}(\boldsymbol{x}, i, j) = (\phi_1(\boldsymbol{x}, i, j), \ldots, \phi_M(\boldsymbol{x}, i, j))$. For instance, one of the derived features for the dist mod-pos side template in Figure 3.3 is given by

$$\phi_m(\boldsymbol{x}, i, j) = \begin{cases} 1 & \text{if dist=2 and mod-pos=noun and side=left,} \\ 0 & \text{otherwise.} \end{cases}$$

Observe that this feature captures a context that is not used by the DT in Figure 3.2. Indeed, we drop the DT feature values when generating the templates and then instantiate these templates based on every context that occurs in the dataset.

3.6 Empirical Results

In this section, we compare the performances of systems based on the proposed EFG method to a system based on the best available set of manual templates for dependency parsing. For this task, our systems do not make use of second- or third-order features. Thus, we use only first-order features to perform this comparison. In Table 3.2, we show the performance of an SPerc system using the best available manual templates along with the performances of two systems based on EFG. First, let us focus on a comparison of EFG to manual templates under same conditions, i.e., same basic features, same learning algorithm, and same datasets. We use the SPerc learning algorithm and the first-order features provided in the templates from McDonald et al. (2006). The first row of Table 3.2 shows the performance when using the

Basic Features	**Feature Generation**	**UAS**
1st order	Manual	90.06
1st order	EFG	90.28
1st+ck+clause	EFG	92.66

Table 3.2: Performances of EFG and manual templates on the Portuguese CoNLL-2006 dependency parsing dataset.

manual templates; and, the second row presents the performance when using EFG to automatically generate non-linear features. EFG outperforms manual templates by 0.22 on UAS. This is not a very big improvement, but shows that EFG is able to automatically generate complex feature templates that are competitive, and even better, than state-of-the-art templates that require substantial human effort.

Fernandes et al. (2010b) present text chunking and clause identification for the Bosque corpus, which comprises the Portuguese CoNLL-2006 dataset. We provide these two basic pieces of information as basic features to the EFG method, as well as the basic features used earlier, and train an SPerc model using the provided templates. The performance obtained by this system is shown in the last row of Table 3.2. We can see that we achieve an impressive improvement around 2.4% on UAS. Moreover, this improvement is achieved by simply including two basic features, without any human effort, as would be required if one used manual templates.

4
Entropy-Guided Structure Learning Framework

We show in this chapter that the proposed entropy-guided feature generation method is naturally integrated to the general structure learning framework, thus extending it. We again make use of dependency parsing as an illustrative application to show that the system presented in the previous chapters is an instantiation of the extended framework.

Structure learning is to learn a function that maps an input $\boldsymbol{x}$ to the correct output structure $\boldsymbol{y}$. The output is an arbitrary structure, i.e., it comprises many variables with complex interdependencies. In the SL framework, the prediction problem is recast as an optimization problem of the form

$$F(\boldsymbol{x}; \boldsymbol{w}) = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}), \tag{4-1}$$

where $\boldsymbol{w} = (w_1, \ldots, w_M)$ is the parameter vector of some learned model, $\mathcal{Y}(\boldsymbol{x})$ is the set of feasible predictions for the given input, and $s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w})$ is a $\boldsymbol{w}$-parameterized scoring function that measures how well an output $\boldsymbol{y}$ fits the input $\boldsymbol{x}$. The prediction output space is arbitrary, but the scoring function must have the following strict form

$$s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle, \tag{4-2}$$

where $\langle \cdot, \cdot \rangle$ is the scalar product operator and $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = (\phi_1(\boldsymbol{x}, \boldsymbol{y}), \ldots, \phi_M(\boldsymbol{x}, \boldsymbol{y}))$ is some joint feature vector representation of the input-output pair. That is, the scoring function is linear on the feature representation.

4.1
Feature Factorization

Each value $\phi_m(\boldsymbol{x}, \boldsymbol{y})$ in the feature vector is called the *global* feature m, which is the value of a specific feature m on the whole structure. For dependency parsing, we define the global feature m as $\phi_m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} \phi_m(\boldsymbol{x}, i, j)$, which is the sum of the *local* feature values over all *edges* in the tree $\boldsymbol{y}$. In that way, we can rewrite the dependency tree scoring function

in the general form of (4-2)

$$\begin{aligned} s(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) &= \sum_{(i,j)\in\boldsymbol{y}} \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j)\rangle \\ &= \sum_{(i,j)\in\boldsymbol{y}} \sum_{m=1}^{M} w_m \cdot \phi_m(\boldsymbol{x}, i, j) \\ &= \sum_{m=1}^{M} w_m \sum_{(i,j)\in\boldsymbol{y}} \phi_m(\boldsymbol{x}, i, j) \\ &= \sum_{m=1}^{M} w_m \cdot \phi_m(\boldsymbol{x}, \boldsymbol{y}) \\ &= \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})\rangle. \end{aligned}$$

Thus, global DP features are factored along the edges of the dependency tree. Consequently, the score of a dependency tree in the prediction function is also factored along its edges. Feature factorization is a key point in SL modeling and must give rise to efficient prediction algorithms. For DP, for instance, we end up with a maximum branching problem that is efficiently solved by the Chu-Liu-Edmonds algorithm.

4.2 Entropy-Guided Feature Generation

The derived feature vector $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$ is automatically generated by means of the proposed entropy-guided feature generation method. EFG induces feature templates by conjoining the available basic features and then instantiates these templates to generate the derived feature vectors. Basic features are factored in the same way as derived features. This factorization determines the prediction scoring function and, consequently, directly affects the prediction problem, which is the core of the SL framework. Therefore, we use the same factorization to derive the EFG basic dataset. For each factor in a structured example, we generate an example in the basic dataset, which comprises a vector of basic features and a decision variable. These variables correspond to local decision variables in the prediction problem. Hence, EFG conjoins basic features that help to discriminante the prediction problem variables.

For instance, dependency parsing features are factored along candidate edges. Thus, for each edge in a training sentence, we generate an example in the basic dataset. And, a binary decision variable is associated with each candidate edge and determines whether an edge is present in the corresponding decision

tree. Hence, EFG generates feature templates that are highly discriminative with respect to dependency edge prediction, which corresponds to the local decision in dependency parsing.

Each generated template is used to instantiate a feature in $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$. Then, this derived feature vector is used to train the structured model. Therefore, the structured model is linear on the derived feature representation, which corresponds to a non-linear combination of the basic features. EFG is completely aligned to the SL framework and it is naturally integrated in it.

4.3 Training Algorithm

There are some training algorithms that learn the parameter vector $\boldsymbol{w}$ from a given training dataset $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{y})\}$ of correct input-output pairs. For instance, Collins (2002b) proposed the structured perceptron algorithm, a generalization of the well known binary perceptron algorithm for sequence labeling problems. The structured perceptron can be easily applied to any structured problem. Collins (2002b) also proved that the structured perceptron converges to a zero-error solution, if one exists. Crammer and Singer (2003) proposed the margin infused relaxed algorithm (MIRA), an online algorithm to train structured models for multiclass problems. MIRA can also be extended for virtually any structured problem and, for instance, is used in MSTParser. Crammer and Singer (2003) also proved some mistake bounds for an algorithm class called *ultraconservative*, which includes MIRA and structure perceptrons. SVM^{struct} Tsochantaridis et al. (2004) formulates the structure learning problem through a regularized max-margin framework, inspired on the binary support vector machine formulation. They also proposed a cutting plane method to efficiently solve this problem. However, this method still requires more computational power and memory than online algorithms like structure perceptron and MIRA. Additionally, the online algorithms are much simpler to implement than SVM^{struct}.

In this work, we use the structure perceptron algorithm (Collins, 2002b). Given a training sample $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{y})\}$ of correct input-output pairs, the algorithm generates a sequence of models until convergence. At each iteration, a training instance is drawn from $\mathcal{D}$ and two major steps are performed: prediction using the current model and model update based on the difference between the correct and the predicted outputs. We use the large-margin structure perceptron Fernandes and Brefeld (2011).

During training, instead of the ordinary prediction problem in (4-1), we

use the following loss-augmented version

$$F_\ell(\boldsymbol{x};\boldsymbol{w}) = \arg\max_{\boldsymbol{y}'\in\mathcal{Y}(\boldsymbol{x})} \left[\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y}')\rangle + C\cdot\ell(\boldsymbol{y},\boldsymbol{y}')\right],$$

where $\ell(\cdot,\cdot) \geq 0$ is an appropriate loss function that measures the difference between a candidate output and the correct one. For dependency trees, as presented in Section 2.2, we use a loss function that just counts the number of incorrect edges in the predicted tree. This loss function factorizes along the dependency tree edges, just like the global features. Thus, the nature of the underlying optimization problem is not modified when using the loss-augmented prediction. This is a desirable, yet *not necessary*, characteristic of loss functions in the SL framework. The model update usually is also factored along the output structure and efficient algorithms can be used. For dependency trees, for instance, this update can be performed in linear time, since a tree has no more than N edges (for a sentence with N tokens plus the artificial token).

We further extend the SL framework by introducing the EFG method into it. EFG is naturally integrated in the SL framework as a preprocessing step. In Figure 4.1, we present the pseudo-code of the entropy-guided large-margin structure perceptron. EFG is used to generate the derived feature vectors

$\boldsymbol{\Phi}(\mathcal{D}) \leftarrow \texttt{EFG}(\mathcal{D})$
$\boldsymbol{w}^0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}$
    $\hat{\boldsymbol{y}} \leftarrow \arg\max_{\boldsymbol{y}'\in\mathcal{Y}(\boldsymbol{x})} \left[\langle \boldsymbol{w}^t, \boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y}')\rangle + C\cdot\ell(\boldsymbol{y},\boldsymbol{y}')\right]$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y}) - \boldsymbol{\Phi}(\boldsymbol{x},\hat{\boldsymbol{y}})$
    $t \leftarrow t+1$
return $\frac{1}{t}\sum_{k=1}^{t} \boldsymbol{w}^k$

Figure 4.1: ESL training algorithm – the entropy-guided large-margin structure perceptron.

$\boldsymbol{\Phi}(\mathcal{D}) = \{\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y})\}_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}}$. The derived features are then used to train the structured model. Note that, when a correct prediction is made, that is $\hat{\boldsymbol{y}} = \boldsymbol{y}$, the model does not change, that is $\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k$. When the prediction is wrong, the update rule favors the correct output $\boldsymbol{y}$ over the predicted one $\hat{\boldsymbol{y}}$. Regarding binary features, for instance, the update rule increases the weights of features that are present in $\boldsymbol{y}$ but missing in $\hat{\boldsymbol{y}}$ and decreases the weights of features that are present in $\hat{\boldsymbol{y}}$ but not in $\boldsymbol{y}$. The weights of features that are present in both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are not changed.

A simple extension of Novikoff's theorem (Novikoff, 1962) shows that the structure perceptron is guaranteed to converge to a zero loss solution, if one exists, in a finite number of steps (Altun et al., 2003; Collins, 2002a). The convergence theorem for SPerc is stated in Theorem 1. Crammer and Singer (2003) further prove some mistake bounds for the structure perceptron algorithm.

Theorem 1 (Structure Perceptron Convergence) *For any training dataset $\mathcal{D}$ that is separable by margin δ, the structure perceptron algorithm makes no more than $\frac{R^2}{\delta^2}$ prediction errors, where $R = \max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D};\boldsymbol{y}'\in\mathcal{Y}(\boldsymbol{x})} ||\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y}) - \boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y}')||$ is the radius of a hypersphere that, centered at $\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y})$, encloses the joint feature vectors for all alternative outputs $\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})$, for all training examples $(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}$.*

4.4 Kernelization

We argue that EFG has two main advantages over kernel functions. EFG training algorithm is much faster than kernelized algorithms, and EFG makes generalization performance control easier than with kernel methods. In this section, we present the kernelized structure perceptron in order to better understand the differences between this method and EFG.

Analogously to the binary perceptron algorithm, its structure generalization can be easily kernelized. Given the sequence $(\boldsymbol{x}^1,\boldsymbol{y}^1,\hat{\boldsymbol{y}}^1),\ldots,(\boldsymbol{x}^T,\boldsymbol{y}^T,\hat{\boldsymbol{y}}^T)$ of inputs, correct outputs and predicted outputs considered by the training algorithm up to iteration T, the parameter vector at this point can be defined as

$$\boldsymbol{w}^T = \sum_{t=1}^{T-1} \left[\boldsymbol{\Phi}(\boldsymbol{x}^t,\boldsymbol{y}^t) - \boldsymbol{\Phi}(\boldsymbol{x}^t,\hat{\boldsymbol{y}}^t)\right].$$

The algorithm can keep track of how many times each alternative output $\hat{\boldsymbol{y}}$ has been predicted instead of the correct output $\boldsymbol{y}$ for each example pair $(\boldsymbol{x},\boldsymbol{y})$ by means of counters $\alpha_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{y}}}$. Thus, the parameter vector can be rewritten as

$$\boldsymbol{w} = \sum_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{y}}} \alpha_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{y}}} \cdot \left[\boldsymbol{\Phi}(\boldsymbol{x},\boldsymbol{y}) - \boldsymbol{\Phi}(\boldsymbol{x},\hat{\boldsymbol{y}})\right], \tag{4-3}$$

which is called the *dual model representation*. The output space $\mathcal{Y}(\boldsymbol{x})$ of most SL problems is exponential on the input size or even infinity. Thus, the dual model representation may comprise an intractable number of parameters. However, these parameters are initially *zero* for all $\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{y}}$ and only need to be instantiated once the respective triple is actually seen.

Using (4-3), the objective function of the prediction problem can also be rewritten as

$$\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}', \boldsymbol{y}') \rangle = \sum_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{y}}} \alpha_{\boldsymbol{x},\boldsymbol{y},\hat{\boldsymbol{y}}} \cdot \left[\langle \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{\Phi}(\boldsymbol{x}', \boldsymbol{y}') \rangle - \langle \boldsymbol{\Phi}(\boldsymbol{x}, \hat{\boldsymbol{y}}), \boldsymbol{\Phi}(\boldsymbol{x}', \boldsymbol{y}') \rangle \right],$$

which depends only on inner products of feature vectors of the form $\langle \boldsymbol{\Phi}^a, \boldsymbol{\Phi}^b \rangle$, where $\boldsymbol{\Phi}^a$ and $\boldsymbol{\Phi}^b$ are shortcuts to, respectively, $\boldsymbol{\Phi}(\boldsymbol{x}^a, \boldsymbol{y}^a)$ and $\boldsymbol{\Phi}(\boldsymbol{x}^b, \boldsymbol{y}^b)$ for any two input-output pairs $(\boldsymbol{x}^a, \boldsymbol{y}^a)$ and $(\boldsymbol{x}^b, \boldsymbol{y}^b)$. Following the *kernel trick* (Vapnik, 1998), the inner products of feature vectors can then be replaced by an appropriate kernel function

$$K(\boldsymbol{\Phi}^a, \boldsymbol{\Phi}^b) = \langle \boldsymbol{\Psi}(\boldsymbol{\Phi}^a), \boldsymbol{\Psi}(\boldsymbol{\Phi}^b) \rangle,$$

where $\boldsymbol{\Psi}(\cdot)$ expands elements from the original feature vector space $\boldsymbol{\Phi}(\cdot, \cdot)$ to a much higher dimensional space. The kernel trick relies on the kernel function $K(\cdot, \cdot)$ to efficiently compute inner products in the high dimensional space of $\boldsymbol{\Psi}$ without explicitly expanding the original feature space.

The most successful kernel function family for NLP problems is the *polynomial kernel*. Considering binary features, a polynomial kernel of degree d conjoins all possible combinations with up to d original features. The polynomial kernel of degree d can be efficiently computed by

$$K_d(\boldsymbol{\Phi}^a, \boldsymbol{\Phi}^b) = \left(\langle \boldsymbol{\Phi}^a, \boldsymbol{\Phi}^b \rangle + 1 \right)^d,$$

which involves only an inner product in the original feature space, a sum of a constant, and an exponentiation. For instance, if $d = 2$ and the original space has exactly 3 *binary* features, then the *explicit* polynomial kernel expansion of $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = (\phi_1, \phi_2, \phi_3)$ corresponds to $\boldsymbol{\Psi}(\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})) = (1,\ \phi_1,\ \phi_2,\ \phi_3,\ \phi_1\phi_2,\ \phi_1\phi_3,\ \phi_2\phi_3)$, if we omit redundant permutations.

The polynomial kernel of degree d is equivalent to generating all possible templates with length up to d. The problem with these kernels is that the only way to control which combinations are used is through the parameter d. For some SL task, for instance, $d = 2$ can be not enough to capture all relevant contextual patterns, but $d = 3$ can bring so many patterns that it is harmful to the generalization performance. This is a known issue with kernel methods and is related to overfitting. Another issue with kernel functions is training time. Performing predictions with the dual model is much slower than with the primal, because the former is represented by a list of dual variables that usually keeps growing during training. Since the prediction problem is constantly solved during training, the training algorithm becomes very slow.

Just like features, kernel functions can also be decomposed along the output structures. Thus, the dual model representation can be even more sparse by using α counters for each factor that appears in $(\boldsymbol{x}, \boldsymbol{y})$ but not in $(\boldsymbol{x}, \hat{\boldsymbol{y}})$, and vice versa. For DP, for instance, we can store a counter for each possible edge within a training sentence.

4.5 Empirical results

We compare ESL to polynomial kernels on two text chunking tasks. We use ESL to train a text chunking system on the Portuguese dataset provided by Fernandes et al. (2010b). We also train a kernelized SPerc system on the same data using a second-degree polynomial kernel. Previous work (Kudo and Matsumoto, 2001; Wu et al., 2006) report that this is the optimum degree for text chunking. Again, in this experiment, we use the same basic features, training algorithm, and datasets for both systems. In the first row of Table 4.1, we report the performances of these two systems. ESL outperforms the kernel

Task	**Kernel Method**		**ESL**	
	Learning	**F_1**	**F_1**	**Error Reduction**
Portuguese Chunking	SPerc	86.67	87.72	7.9%
English Chunking	SVM	93.48	94.12	9.8%

Table 4.1: Comparison of ESL to second-degree polynomial kernel.

method, reducing its error by 7.9%. That is an impressive achievement, since polynomial kernels present state-of-the-art results on many tasks, and training time for kernel methods is more than one order of magnitude longer than for ESL.

We also compare ESL to a kernel system on the CoNLL-2000 text chunking dataset. The second row of Table 4.1 presents the performances of ESL and a second-degree polynomial kernel system. The kernel method result is reported by Kudo and Matsumoto (2001). They use an SVM algorithm to train their system. However, they employ a training strategy that considers sequential interdependencies among output variables and also use Viterbi decoding during test. ESL outperforms this system, reducing its error by 9.8%.

5
Prediction Problems

In this chapter, we discuss prediction problems, a component that plays a central role in the SL framework. The prediction problem for an input $\boldsymbol{x}$ in the SL formulation has the form

$$\arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle.$$

This is a very general form for an optimization problem. In fact, the only requirement is that the objective function be linear on some feature representation. On the other hand, the joint feature representation has no explicit restriction and the output space is also arbitrary. Thus, ESL is just a framework for learning parameters for general linear predictors. And, there are several learning algorithms for this framework with strong guarantees regarding both prediction performance and learning time. In Table 5.1, we present a list of SL problems along with the corresponding output structures and prediction problems.

Task	Output Structure	Prediction Problem
Dependency parsing	Rooted tree	Maximum branching
Part-of-speech tagging	Sequence	Longest path on DAG
Text chunking	Sequence	Longest path on DAG
Quotation extraction	Segmentation	Weighted interval scheduling
Coreference resolution	Clustering	*Latent* maximum branching

Table 5.1: List of tasks and the corresponding output structures and prediction problems.

The output space $\mathcal{Y}(\boldsymbol{x})$ is represented by task-specific hard constraints that are embedded in the prediction algorithm. Hence, $\mathcal{Y}(\boldsymbol{x})$ can comprise any constraint that is efficiently handled by the optimization algorithm. For most SL problems, these constraints are difficult to be learned from data; or, at least, it is unnecessary to do so, since they are never violated in any example. For instance, in dependency parsing, the learning algorithm expends no effort on estimating parameters to avoid cycles in the output structure. The prediction algorithm never extracts cycles because it is constrained to extract only trees. This is a very flexible way to represent the valid prediction outputs.

Additionally, it allows the modeler to make use of countless theoretical and practical results from combinatorial optimization.

The second arbitrary component in the prediction problem is the joint feature vector representation $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = (\phi_1(\boldsymbol{x}, \boldsymbol{y}), \ldots, \phi_M(\boldsymbol{x}, \boldsymbol{y}))$. Each feature function $\phi_m(\boldsymbol{x}, \boldsymbol{y})$ is called global feature because it gives a value regarding the whole output structure $\boldsymbol{y}$. However, global features are usually factored along the output structure and, consequently, the scoring function $\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle$ is factored in the same way. Otherwise, the prediction algorithm would need to enumerate all possible outputs to determine the best scoring one. In multiclass classification, for instance, there is no feature factorization and the prediction algorithm just enumerates all classes, computes their scores and picks the highest scoring one. That is feasible when the number of classes is limited, which is the case for multiclass classification. The feature factorization defines the dependencies among the output variables. For dependency parsing, features are factored on dependency edges (i, j) and a tree score is given just by independently summing the scores of its edges. Thus, the prediction problem is equivalent to the maximum branching problem.

In the next sections, we briefly describe some important structures along with the proposed feature factorizations and the resulting prediction algorithms. These aspects are discussed later in more details.

5.1 Rooted Tree

Dependency parsing consists in predicting a rooted tree underlying a given sentence. The nodes of this tree are fixed: the sentence tokens. Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_N)$ be an input sequence, where x_t is the t-th token and x_0 is a special node that is always the root of the tree. The prediction output space $\mathcal{Y}(\boldsymbol{x})$ is the set of all rooted trees whose nodes are tokens in $\boldsymbol{x}$ and the root node is x_0.

In this work, we use the feature factorization proposed by McDonald et al. (2005), which defines features over independent edges. In this case, the prediction problem is efficiently solved by the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). However, more complex models are possible. McDonald and Pereira (2006) propose features that depend on more than two tokens. More specifically, they use the so called second-order features that depend on two dependency edges (i, j) and (i, k). Koo et al. (2010) further extends this model by introducing third-order features that also depend on two edges, but include dependencies on grandparent tokens. They show significant improvements on parsing performance by including

high-order features. However, the complexity of the prediction algorithm also grows. In fact, the prediction problem becomes NP-hard when these features are considered. Thus, they solve the problem by approximation algorithms.

5.2 Sequence Labeling

Sequence labeling Dietterich (2002) is to find a sequence of labels $\boldsymbol{y} = (y_1, \ldots, y_N)$, where $y_t \in S$, for a given input sequence of tokens $\boldsymbol{x} = (x_1, \ldots, x_N)$. That is, each token x_t is annotated with a label $y_t \in S$, where S is a fixed set of labels. The prediction output space for an input $\boldsymbol{x}$ is simply the set of all possible label sequences with length N, that is $\mathcal{Y}(\boldsymbol{x}) = S^N$.

Collins (2002b) proposes a feature factorization for sequence labeling problems that relies on a Markovian property. The best scoring label for a specific token x_t depends only on the label itself y_t and the previous token label y_{t-1}. In this factorization, there are two types of features: $\boldsymbol{\Phi}^{\text{obs}}(\boldsymbol{x}, y_t)$ are observation features that depend only on the input $\boldsymbol{x}$ and individual token labels; and $\boldsymbol{\Phi}^{\text{trans}}(y_{t-1}, y_t)$ are transition features that depend on two consecutive labels. The resulting prediction problem is reduced to the longest path problem on a directed acyclic graph, which can be efficiently solved by a dynamic programming algorithm.

5.3 Sequence Segmentation

Given a document, quotation extraction is to identify quotes and, additionally, to associate each quote to its author. For a training example $(\boldsymbol{x}, \boldsymbol{y})$, the input $\boldsymbol{x}$ is composed by a set of K candidate authors $\boldsymbol{a} = \{a_1, \ldots, a_K\}$ and a set of N candidate quotes $\boldsymbol{q} = \{q_1, \ldots, q_N\}$, where each quote corresponds to a segment of the input document. The set of candidate quotes can overlap each other, but the correct quotes do not. The output space is thus all subsets of non-overlapping candidate quotes such that each selected quote is associated to exactly one author.

Fernandes (2012) proposes a structure learning modeling for quotation extraction in which features depend on the association of a quote to an author. In that way, the prediction problem is to find non-overlapping segments associated to authors whose weights are maximum. This problem is equivalent to the weighted interval scheduling for which there is an efficient dynamic programming algorithm.

5.4 Clustering

Coreference resolution Pradhan et al. (2011) consists in identifying mentions to real-world entities in a document and clustering mentions that refer to the same entity. This task is usually split into two subtasks: mention detection and mention clustering. Mention detection is easily performed by recovering all noun phrases in the document. The most interesting task is mention clustering. The prediction output space for this task comprises all possible clustering of the given mentions. The number of clusters is unknown.

Usually, coreference systems use features that depend on pairs of mentions. We follow this idea, but we introduce a novel modeling for coreference resolution. We assume that an entity cluster is represented by a rooted tree, denoted *coreference tree*. A directed edge (m_i, m_j) from mention m_i to mention m_j in this tree indicates that m_j is a reference to the more general mention m_i. In that way, we model the prediction problem as a maximum branching problem on the graph whose nodes are the given set of mentions.

6
Dependency Parsing

Dependency parsing is to identify a rooted tree underlying a sentence. The nodes in this tree are the sentence tokens. The dependency tree represents the syntactic dependencies among the sentence tokens.

6.1
Task Formalization

Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_N)$ be a sentence, where x_i is the i-th token and x_0 is an artificial token which is always the root of the dependency tree. For a given input sentence $\boldsymbol{x}$, the prediction output space $\mathcal{Y}(\boldsymbol{x})$ is the set of all rooted trees whose nodes are the tokens in $\boldsymbol{x}$ and the root node is x_0. For any dependency tree $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$, we say that $(i, j) \in \boldsymbol{y}$ whenever token x_j *modifies* the *head* token x_i in the tree $\boldsymbol{y}$.

6.2
Feature Factorization

We follow McDonald et al. (2005, 2006); McDonald and Pereira (2006) and factorize the joint feature vector $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$ along the edges of the dependency tree $\boldsymbol{y}$. In that way, an edge (i, j) connecting x_i to x_j is represented by a vector $\boldsymbol{\Phi}(\boldsymbol{x}, i, j) = (\phi_1(\boldsymbol{x}, i, j), \ldots, \phi_M(\boldsymbol{x}, i, j))$ of M binary features. These features describe the dependency between the head token x_i and the modifier token x_j. Then, the *global* feature vector is

$$\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} \boldsymbol{\Phi}(\boldsymbol{x}, i, j),$$

which gives the frequency distribution of the local features in the tree $\boldsymbol{y}$.

6.3
Prediction Problem

The prediction problem for this DP modeling is reduced to the maximum branching problem, which can be efficiently solved by Chu-Liu-Edmonds algorithm. In the following, we just summarize this result.

The objective function of the prediction problem is

$$s(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle.$$

Using the aforementioned factorization, it is easy to see that

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle,$$

which is just the sum of the edge weights, that is,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} s(i, j),$$

where $s(i, j) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, i, j) \rangle$ gives the weight of edge (i, j). The objective function of the prediction problem is equivalent to the tree weight given by this function. Thus, we can generate a maximum branching instance using $s(i, j)$ as edge weight function. In that way, to solve this instance is equivalent to solve the DP prediction problem in the ESL framework.

We use a loss function that just counts how many predicted edges are not correct, that is $\ell(\boldsymbol{y}, \boldsymbol{y}') = \sum_{(i,j) \in \boldsymbol{y}'} \mathbf{1}[(i, j) \notin \boldsymbol{y}]$.

6.4 Basic Features

We use the same basic features proposed by McDonald et al. (2006). For a given edge (i, j), we have the following feature list:

- *Side* – Whether x_j is on the left or on the right side of x_i in the input sentence.
- *Distance* – How many tokens there are between x_i and x_j.
- *Word* – Surface representation of both x_i and x_j.
- *POS* – Part-of-speech tag of x_i, x_{i-1}, x_{i+1}, x_j, x_{j-1} and x_{j+1}.
- *POS Between* – Part-of-speech tag of all tokens between x_i and x_j.
- *Feats* – Syntactic and morphological features that are included in the CoNLL-2006 dataset. We include these features for x_i, x_j and all tokens that occur between x_i and x_j.

Fernandes et al. (2010b) present text chunking and clause identification for the Bosque corpus, which comprises the Portuguese CoNLL-2006 dataset. We perform additional experiments using basic features based on this information. These features are the following:

– *Chunk Tag* – The chunk tag, in IOB2 style, for x_i and x_j.

– *Start Clause* – Indicates whether a token starts a clause.

– *End Clause* – Indicates whether a token ends a clause.

6.5 Empirical Results

The CoNLL-2006 (Buchholz and Marsi, 2006) provided a dependency parsing dataset that is derived from Bosque (Freitas et al., 2008), a Portuguese corpus comprising European and Brazilian news articles. In Table 6.1, we provide basic statistics about this dataset.

	Sentences	**Tokens**
Train	8,546	207,000
Test	241	5,838

Table 6.1: Bosque dependency parsing dataset statistics.

In this section, we compare the performances of systems based on the proposed ESL framework with state-of-the-art systems. Our systems use only first-order features, while the best performing systems for this task use second- and third-order features. In Table 6.2, we show the performances of several systems along with two systems based on ESL. The two first rows in the

System	**Learning Algorithm**	**Basic Features**	**Feature Generation**	**UAS**
Dual Decomposition	MIRA	3rd order	Manual	93.03
		2nd order		92.57
MSTParser	MIRA	2nd order	Manual	91.36
		1nd order		90.68
SPerc	SPerc	1st order	Manual	90.06
ESL	SPerc	1st order	EFG	90.28
		1st+ck+clause	EFG	92.66

Table 6.2: Performances of ESL and state-of-the-art systems on the Portuguese CoNLL-2006 dependency parsing dataset.

table present the system results by Koo et al. (2010). This system uses an algorithm based on dual decomposition (DD) to approximately solve the NP-hard optimization problem when second- and third-order features are considered. The DD algorithm provides a certificate of optimality for 99.65% of the test examples. The third and fourth rows in the results table show the results of MSTParser with first- and second-order features (McDonald et al., 2005, 2006; McDonald and Pereira, 2006). As we showed before, our ESL system outperforms an SPerc with the first-order templates from McDonald et al. (2006). When we provide text chunking and clause features (Fernandes

et al., 2010b) to our ESL system, we achieve a performance comparable with systems based on second- and third-order features. Moreover, this improvement is achieved by simply including two basic features, without any human effort, as would be required if one used manual templates.

We use 10% of the training data as validation set in order to pick the ESL meta-parameters. The loss weight C is set to 300 and the number of epochs is 10. We generate templates containing from 2 to 4 basic features.

7
Part-of-Speech Tagging

Part-of-speech tagging is to categorize words according to its part of speech in a given sentence. In Figure 7.1, we present the sentence Flies like flowers with the corresponding part of speech of each word. The task is to give

Word	Flies	like	flowers
POS	noun	verb	noun

Figure 7.1: Part-of-speech tagging example.

each word a tag according to its part of speech. The set of tags, or simply tagset, is fixed within a particular POS tagging task. However, different applications or datasets provide different tagsets, mainly varying POS granularity. For instance, some POS tagsets include only one broad category for verbs, while others include categories like main verb, auxiliary verb, among others. The main difficult of this task is ambiguity, since one word can have different POS tags depending on the context. For instance, the words Flies and flowers can act as verbs in other contexts; and the word like can act as adverb, noun, conjunction, among several other parts of speech. POS provides basic morphological and syntactic information to more complex NLP tasks. It can be even directly used to solve simple information extraction tasks.

We use a general sequence labeling modeling to approach POS tagging. In Section 7.1, we formalize this general task. In order to apply ESL framework to this problem, we still need to define the factorization of the global feature vector $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$ along the output sequence $\boldsymbol{y}$, and the resulting prediction problem. We describe these two aspects in Section 7.2 and Section 7.3, respectively. In Section 7.5, we present empirical results of two ESL applications for POS tagging.

7.1
Task Formalization

The general task of sequence labeling is to find a mapping from an input token sequence $\boldsymbol{x} = (x_1, \ldots, x_N)$ to a label sequence $\boldsymbol{y} = (y_1, \ldots, y_N)$, where $y_t \in S$. That is, each token in $\boldsymbol{x}$ is tagged with a label, or tag, from a given

set S. The output space $\mathcal{Y}(\boldsymbol{x})$ for an input sequence $\boldsymbol{x}$ is the set of all possible sequences of N labels, i.e., $\mathcal{Y}(\boldsymbol{x}) = S^N$. Part-of-speech tagging is an instance of sequence labeling in which S is the given POS tagset.

7.2 Feature Factorization

We use the decomposition scheme from Collins (2002b). Each input token x_t is represented by a vector $\boldsymbol{\Phi}^{\text{surf}}(x_t) = (\phi_1^{\text{surf}}(x_t), \ldots, \phi_M^{\text{surf}}(x_t))$ of M binary features that we call *surface features*. For instance, some surface features that are *present* – have value 1 – in the second token of the example in Figure 7.1 are: `the current word is` like, `the previous word is` Flies, and `the previous word is capitalized`. The number of such features in a dataset with hundreds of thousands of tokens is huge, but just a dozen are active on each token. For a given example $(\boldsymbol{x}, \boldsymbol{y})$, the surface features depend only on the input $\boldsymbol{x}$, which is fixed within the prediction problem. To compose $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$, surface features are combined with the output labels in $\boldsymbol{y}$. Additionally, transition features within $\boldsymbol{y}$ are used to cope with label interdependencies.

Each surface feature $m \in \{1, \ldots, M\}$ is combined with every possible label $s \in S$ to generate the *observation* feature $\phi_{m,s}^{\text{obs}}(x_t, y_t) = \phi_m^{\text{surf}}(x_t) \cdot \mathbf{1}[y_t = s]$, for a given token x_t and its corresponding label y_t. This observation feature indicates whether both the surface feature m is present in token x_t and the token label y_t is equal to s. Then, we can define the observation feature *vector* for a token-label pair (x_t, y_t) as

$$\boldsymbol{\Phi}^{\text{obs}}(x_t, y_t) = \left(\phi_{m,s}^{\text{obs}}(x_t, y_t)\right)_{m \in \{1,\ldots,M\}; s \in S}.$$

Furthermore, we combine these local vectors into the *global* observation feature vector

$$\boldsymbol{\Phi}^{\text{obs}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{N} \boldsymbol{\Phi}^{\text{obs}}(x_t, y_t),$$

which is the frequency distribution of the observation features in $(\boldsymbol{x}, \boldsymbol{y})$.

For each possible pair of labels $s, r \in S$, the *transition* feature $\phi_{s,r}^{\text{trans}}(y_{t-1}, y_t) = \mathbf{1}[y_{t-1} = s] \cdot \mathbf{1}[y_t = r]$ indicates whether two consecutive labels y_{t-1} and y_t are equal to s and r, respectively. The transition feature vector is then defined as

$$\boldsymbol{\Phi}^{\text{trans}}(y_{t-1}, y_t) = \left(\phi_{s,r}^{\text{trans}}(y_{t-1}, y_t)\right)_{s,r \in S},$$

which is a unit vector whose non-zero position is the one corresponding to

$s = y_{t-1}$ and $r = y_t$. The global transition feature vector is given by

$$\boldsymbol{\Phi}^{\text{trans}}(\boldsymbol{y}) = \sum_{t=2}^{T} \boldsymbol{\Phi}^{\text{trans}}(y_{t-1}, y_t),$$

which is the frequency distribution of the transitions in the output sequence $\boldsymbol{y}$. Finally, the global feature vector is simply the concatenation of the global observation feature vector and the global transition feature vector, that is

$$\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{\Phi}^{\text{obs}}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{\Phi}^{\text{trans}}(\boldsymbol{x}, \boldsymbol{y})\right).$$

7.3 Prediction Problem

The used feature factorization relies on a Markovian property. Thus, in the prediction problem, the best scoring label for a specific token x_t depends only on its label y_t and the previous token label y_{t-1}. In that way, the prediction problem can be reduced to a longest path problem on an weighted directed acyclic graph (DAG), which can be efficiently solved by dynamic programming. In Figure 7.2, we present an example of such DAG for a sentence with three tokens and a tagset with 2 labels (a and b). This graph comprises one layer for

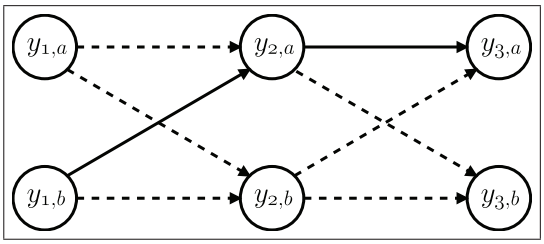


Figure 7.2: Illustrative directed acyclic graph for a sentence $\boldsymbol{x} = (x_1, x_2, x_3)$ and a tagset $S = \{a, b\}$. The continuous path $(y_{1,b}, y_{2,a}, y_{3,a})$ corresponds to the labeling $\boldsymbol{y} = (b, a, a)$.

each token x_t in the input sentence. At every layer t, there is a node labeled $y_{t,s}$ for each label $s \in S$. The node $y_{t,s}$ represents that the t-th token is tagged as s, that is $y_t = s$. For each pair of labels $(s, r) \in S \times S$ and each consecutive layers $t-1$ and t, there is one directed edge $(y_{t-1,s}, y_{t,r})$ in the graph. The edge $(y_{t-1,s}, y_{t,r})$ represents a transition from $y_{t-1} = s$ to $y_t = r$ and its weight is given by

$$s(s, r, x_t) = \langle \boldsymbol{w}^{\text{trans}}, \boldsymbol{\Phi}^{\text{trans}}(s, r) \rangle + \langle \boldsymbol{w}^{\text{obs}}, \boldsymbol{\Phi}^{\text{obs}}(x_t, r) \rangle,$$

where $\boldsymbol{w}^{\text{trans}}$ is the model parameter vector corresponding to transition features and $\boldsymbol{w}^{\text{obs}}$ comprises the parameters for observation features.

Each path from layer $t = 1$ to layer $t = N$ corresponds to a possible output $\boldsymbol{y} = (y_1, \ldots, y_N)$ whose accumulated weight in the graph is

$$s(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{w}^{\text{obs}}, \boldsymbol{\Phi}^{\text{obs}}(x_1, y_1) \rangle + \sum_{t=2}^{N} s(y_{t-1}, y_t, x_t). \tag{7-1}$$

By expanding the edge weight function in the formula above, we have that

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{N} \langle \boldsymbol{w}^{\text{obs}}, \boldsymbol{\Phi}^{\text{obs}}(x_t, y_t) \rangle + \sum_{t=2}^{N} \langle \boldsymbol{w}^{\text{trans}}, \boldsymbol{\Phi}^{\text{trans}}(y_{t-1}, y_t) \rangle.$$

And, by using the feature factorization described earlier, we can derive that

$$\begin{aligned} s(\boldsymbol{x}, \boldsymbol{y}) &= \langle \boldsymbol{w}^{\text{obs}}, \boldsymbol{\Phi}^{\text{obs}}(\boldsymbol{x}, \boldsymbol{y}) \rangle + \langle \boldsymbol{w}^{\text{trans}}, \boldsymbol{\Phi}^{\text{trans}}(\boldsymbol{x}, \boldsymbol{y}) \rangle \\ &= \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle, \end{aligned}$$

where $\boldsymbol{w} = (\boldsymbol{w}^{\text{obs}}, \boldsymbol{w}^{\text{trans}})$ is the complete model, that is the concatenation of the observation and transition parameters. Thus, to find the longest path in the aforementioned DAG is equivalent to solve the ESL prediction problem for the presented sequence labeling modeling.

For sequence labeling, we use the loss function $\ell(\boldsymbol{y}, \boldsymbol{y}') = \sum_{t=1}^{T} \mathbf{1}[y_t \neq y'_t]$ that counts the number of mislabeled tokens.

7.4 Basic Features

Our basic features for POS tagging are obtained from dos Santos and Milidiú (2009a). We use the following features:

- *Word:* The surface form of a token;
- *Prefix/Suffix:* Word prefixes and suffixes up to 5-character long;
- *Known Word Prefix:* Adding (or subtracting) 5-character prefix (or suffix) results in a known word, where known words are the ones that occur in the training dataset;
- *Known Word Bigram:* Occurrence of the word before (or after) a specific word in a given long list of word bigrams. For instance, for the English language, if the word appears after `to`, then it is likely to be a verb in the infinitive form;
- *Word Window:* Words of the previous two tokens and the next two tokens.

7.5 Empirical Results

We evaluate our system performances on two POS datasets: Mac-Morpho (Aluísio et al., 2003), a Portuguese language corpus; and Brown (Francis and Kučera, 1982), an English language corpus. In Table 7.1, we present some statistics of these datasets. Both datasets are split into training and test

Dataset	Language	Tagset size	Training Tokens	Test Tokens
Mac-Morpho	Portuguese	22	1,007,671	213,794
Brown	English	182	950,975	210,217

Table 7.1: Basic statistics of the part-of-speech tagging datasets.

partitions. Brown and Mac-Morpho have relatively the same size, but Brown includes a much larger tagset. Performance on POS tagging is reported on simple token accuracy, that is the percentage of correctly tagged tokens among all tokens.

7.5.1 Mac-Morpho Dataset

The best performing system on the Mac-Morpho dataset is the ETL Committee (dos Santos and Milidiú, 2009a), an ensemble composed by 100 ETL models. We compare our system to this ensemble system and also to the best single model ETL-based system. In Table 7.2, we depict the performance of these systems. We notice that ESL reduces the accuracy error by 5.9% when

System	Accuracy
ETL single model	96.75
ETL Committee	96.94
ESL	**97.12**

Table 7.2: Performances on the Mac-Morpho dataset.

compared to ETL Committee and by 11.4% when compared to the single model ETL system. These are substantial improvements on this dataset, since there is not much room for improvements.

Regarding ESL training meta-parameters, we use the following setting. The number of epochs is 50 and the loss weight parameter C is set to 50. One epoch corresponds to one complete pass over all examples in the training set. The minimum and the maximum feature template length is set to 2.

7.5.2 Brown Dataset

The best performing system on the Brown dataset is also ETL Committee. Thus, in Table 7.3, we again present the performances of the best single model ETL system, ETL Committee, and ESL. We notice that

System	Accuracy
ETL single model	96.69
ETL Committee	**96.83**
ESL	96.72

Table 7.3: Performances on the Brown dataset.

ESL outperforms the single model ETL system, but does not outperform ETL Committee. ETL Committee error is 3.4% smaller than ESL error. Nevertheless, ESL is still competitive with state-of-the-art systems. Moreover, we can also train ensembles of ESL models and probably have some gain in performance.

We use the following values for ESL meta-parameters. The number of epochs is 50 and the loss weight parameter C is set to 100. One epoch corresponds to one complete pass over all examples in the training set. The minimum and the maximum feature template length is set to 2.

8
Text Chunking

Text chunking is another basic NLP task and consists in identifying segments, or *chunks*, of words that are syntactically related in a given sentence. Additionally, each chunk needs to be classified among some classes of interest, which gives the chunk type. In Figure 8.1, we present a sentence along with the corresponding chunking output. In this example, there are three chunk

Sentence	He	reckons	the deficit	will narrow	to	1.8 billion
Chunk	⌊ ⌋	⌊ ⌋	⌊ ⌋	⌊ ⌋	⌊ ⌋	⌊ ⌋
Type	nominal	verbal	nominal	verbal	prep.	nominal
IOB2 Tag	B-NP	B-VP	B-NP I-NP	B-VP I-VP	B-PP	B-NP I-NP

Figure 8.1: Text chunking example.

types: nominal (NP), verbal (VP), and prepositional (PP). Many text chunking systems cast this task as a sequence labeling problem by employing some appropriate tagging style. For instance, the row labeled IOB2 Tag in the figure indicates the token tags that encode the given chunks according to the IOB2 tagging style. This tagging style makes use of three prefix tags: B, for beginning, indicates the first token of a chunk; I, for inside, indicates any other token in a chunk; and O, for outside, indicates tokens that are not part of a chunk (most punctuation marks, for instance). Additionally, the B and I tags are combined with the chunk type.

8.1
Task Formalization

We employ the IOB2 tagging style to cast text chunking as a sequence labeling problem. In that way, we use same modeling presented in the previous chapter, in which the input $\boldsymbol{x} = (x_1, \ldots, x_N)$ is a sequence of tokens and the output $\boldsymbol{y} = (y_1, \ldots, y_N)$ comprises a tag sequence, where $y_i \in S$ is the tag given for token x_i, and S is the IOB2 tagset.

8.2 Feature Factorization

Since we model text chunking as a sequence labeling problem, we use the observation and transition features and decompose them just as for the POS task, that is,

$$\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{\Phi}^{\text{obs}}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{\Phi}^{\text{trans}}(\boldsymbol{x}, \boldsymbol{y})\right).$$

8.3 Prediction Problem

The prediction problem is exactly the same as presented in Section 7.3, that is, a longest path problem on an weighted directed acyclic graph. We also use the same loss function $\ell(\boldsymbol{y}, \boldsymbol{y}') = \sum_{t=1}^{T} \mathbf{1}[y_t \neq y'_t]$, which counts the number of mislabeled tokens.

8.4 Basic Features

Our chunking datasets include the following basic features:

- *Word* – the word of each token;
- *POS tag* – the part-of-speech tag;
- *Basic form* – which type of word among the following types: (i) number; (ii) alphabetical in lower case; (iii) alphabetical in upper case; (iv) capitalized alphabetical; or (v) something else.

The basic features of each token also include the values of the aforementioned features for the three previous tokens and the three next tokens.

8.5 Empirical Results

We evaluate ESL on two text chunking datasets: the Bosque dataset (Fernandes et al., 2010b), a Portuguese language corpus; and the CoNLL-2000 dataset (Sang and Buchholz, 2000), an English language corpus. In Table 8.1, we present the number of chunks, sentences and tokens in each dataset. Both datasets have relatively the same size and are split into training and

Dataset	Language	Training			Test		
		Cks	Sents	Tkns	Cks	Sents	Tkns
Bosque	Portuguese	116,233	9,368	226,758	18,908	1,405	35,256
CoNLL-2000	English	106,978	8,936	211,727	23,852	2,012	47,377

Table 8.1: Basic statistics of the text chunking datasets.

test partitions. Performance for text chunking is reported on precision, recall and F-score. Precision is the percentage of recovered chunks that are correct and recall is the percentage of correct chunks that are recovered. A chunk is considered correct when both its span and its type are correct. The F-score is the harmonic mean of precision and recall. That is, $F = 2 \cdot P \cdot R/(P + R)$, where P is the precision ratio and R is the recall ratio.

8.5.1 Bosque Dataset

Fernandes et al. (2010b) propose a heuristic to extract text chunks from the Bosque treebank (Freitas et al., 2008). The Bosque corpus includes news articles comprising Brazilian and European Portuguese. Fernandes et al. also propose an ETL-based system for text chunking and evaluate it on the Bosque dataset. We report in Table 8.2 the performance of this system along with the performance of our ESL-based system. ESL reduces the error of the ETL

System	P	R	*F*
ETL	**89.61**	85.41	87.46
ESL	88.06	**87.39**	**87.72**

Table 8.2: Performances on the Bosque dataset.

system by 2.1%, regarding F-score. On the other hand, ETL achieves a higher precision than ESL.

We use the following values for the ESL meta-parameters in order to train this system. The number of epochs is 20. The loss weight parameter C is set to 300. And, we generate feature templates containing from 2 to 4 features.

8.5.2 CoNLL-2000 Dataset

System	P	R	*F*
Wu et al. (2006)	**94.16**	**94.26**	**94.21**
Kudo and Matsumoto (2001)	93.47	93.49	93.48
ESL	94.05	94.18	94.12

Table 8.3: Performances on the CoNLL-2000 dataset.

The CoNLL-2000 Shared Task (Sang and Buchholz, 2000) provides an English text chunking dataset that includes some sections from the WSJ in the Penn Treebank. The best performing system on this dataset is presented by Wu et al. (2006). This system introduces a masking strategy to approach hard examples that involve words not seen in the training data. In Table 8.3, we present the performances of our ESL system, Wu et al.'s masking system, and

the kernelized SVM presented in Kudo and Matsumoto (2001). The masking system outperforms ESL, achieving an error that is 1.5% smaller than ESL's. Nevertheless, ESL is still competitive to state-of-the-art systems.

By using the standard validation set, we select the following values for the ESL meta-parameters. The number of epochs is 50, the loss weight parameter C is set to 300, and all feature templates contain 2 features.

9
Quotation Extraction

Quotation extraction is to identify quotes and their authors in a given document. A quote is a segment of the input document and quotes cannot overlap each other. We consider that an input for this task, in addition to the document itself, also includes two sets: the candidate authors and the candidate quotes. Candidate quotes in the input can overlap. In that way, a feasible output for this task is a subset of non-overlapping quotes and their associated authors. In Figure 9.1, we exemplify this task. In this figure, authors are highlighted in bold type and quotes in italic. The subscripted numbers indicate the association between quotes and their respective authors. For instance, the author for the quote 'estranha' is Nélio Machado.

> **Nélio Machado**$_1$, que defende **Daniel Dantas**$_2$, considerou '*estranha*'$_1$ a acusação de que **Dantas**$_2$ teria cogitado subornar **o juiz**$_3$. '*Isso é o fim da picada. Completamente sem fundamento e bem no dia em que o* **Daniel**$_2$ *vai prestar depoimento. Estou inclinado a pedir suspeição* **dele**$_3$ *[***Fausto de Sanctis**$_3$*]. Acho muito estranho, tem conteúdo de mais armação do que qualquer outra coisa'*$_1$ disse **ele**$_1$.

Figure 9.1: Quotation extraction example.

9.1
Task Formalization

An input-output pair $(\boldsymbol{x}, \boldsymbol{y})$ for quotation extraction is represented as follows. The input $\boldsymbol{x} = (\boldsymbol{a}, \boldsymbol{q})$ comprises two sets: the candidate authors $\boldsymbol{a} = \{a_1, \ldots, a_K\}$ and the candidate quotes $\boldsymbol{q} = \{q_1, \ldots, q_N\}$. Each quote $q_i = (s_i, e_i)$, for $i \in \{1, \ldots, N\}$, is a segment in the document and is represented by its starting token s_i and its end token e_i, where $s_i \leq e_i$. The output $\boldsymbol{y} = (y_1, \ldots, y_N)$ is a vector of author indexes, where $y_i \in \{1, \ldots, K\} \cup \{0\}$ indicates the author associated to the quote q_i; and $y_i = 0$ means that q_i is not included in the output $\boldsymbol{y}$.

9.2
Feature Factorization

Fernandes (2012) proposes a structure learning modeling for quotation extraction that is based on an input feature vector $\boldsymbol{\Phi}(i,j) = (\phi_1(i,j), \ldots, \phi_M(i,j))$ that describes the candidate quote-author association (q_i, a_j). Then, for a given output $\boldsymbol{y}$, the global feature *vector* is defined as

$$\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1,\ldots,N; y_i \neq 0} \boldsymbol{\Phi}(i, y_i),$$

which is the histogram of the local features for all associations selected in $\boldsymbol{y}$.

9.3
Prediction Problem

Here, the prediction problem is to find non-overlapping quotes associated to authors whose association weights are maximum. This problem can be reduced to the weighted interval scheduling (WIS) problem for which there is a known efficient dynamic programming algorithm. In order generate a WIS instance from a quotation extraction input $\boldsymbol{x}$, we create an weighted interval for each association (q_i, a_j). The segment, or span, for this interval is given by the quote segment (s_i, e_i); and, given the current model $\boldsymbol{w}$, the interval weight is

$$s(i,j) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(i,j) \rangle.$$

Since the associations $(q_i, a_1), \ldots, (q_i, a_K)$ have the same span in the WIS instance – which is (s_i, e_i) – the WIS algorithm never selects more than one author for q_i. Additionally, if $s(i,j) < 0$ for all $j \in \{1, \ldots, K\}$, then q_i is not selected. And, clearly, overlapping quotes are never selected together. The weight of a complete solution $\boldsymbol{y}$ is then given by

$$\begin{aligned} s(\boldsymbol{y}) &= \sum_{i=1,\ldots,N; y_i \neq 0} s(i, y_i) \\ &= \sum_{i=1,\ldots,N; y_i \neq 0} \langle \boldsymbol{w}, \boldsymbol{\Phi}(i, y_i) \rangle \\ &= \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle. \end{aligned}$$

Thus, this WIS problem is equivalent to the prediction problem in the ESL framework.

We use a loss function that counts how many quotes have been associated to incorrect authors, that is $\ell(\boldsymbol{y}, \boldsymbol{y}') = \sum_{i=1,\ldots,N} \mathbf{1}[y_i \neq y'_i]$.

9.4 Basic Features

We use the same basic features from Fernandes (2012). The following basic features are used for each quote-author association (q_i, a_j):

- *Distance* – Number of tokens between q_i and a_j.
- *Direction* – Indicates whether a_j is on the left or on the right of q_i.
- *Say-verb between* – Indicates whether there is a *say-verb* between q_i and a_j. Fernandes proposes a list of say-verbs that are frequently used to indicate quotes, like say, speak and comment.
- *Number of say-verbs* – Number of say-verbs between q_i and a_j.
- *Author between* – Indicates whether there is an author between q_i and a_j.
- *Quote between* – Indicates whether there is a quote between q_i and a_j.
- *BLS* – Indicates whether Fernandes's baseline system selects (q_i, a_j) association.
- *Say-verb around* – Indicates whether a *say-verb* occurs at most two tokens away from q_i.
- *First letter uppercased* – Indicates whether the first letter in q_i is uppercased.

9.5 Empirical Results

We evaluate ESL on the GloboNotes dataset (Fernandes, 2012) that includes news articles from the Globo.com portal. These articles comprise ten different news genres, namely Sports, General, Celebrities, Arts, Economy, Education, Politics, Science, Technology, and World. This dataset is split into training and test subsets. In Table 9.1, we present basic statistic of these datasets.

	Docs	**Sentences**	**Tokens**	**Quotations**
Train	552	7,963	174,415	802
Test	133	1,834	41,613	205

Table 9.1: GloboNotes dataset statistics.

We compare ESL performance to ETL and a baseline system, both proposed by Fernandes (2012). We present these performances in Table 9.2. Performances are reported in precision, recall and F-score. We notice that ESL outperforms both ETL and the baseline system.

Model	Precision	Recall	*F*
ESL	**83.24**	71.49	**76.80**
ETL	69.44	**73.17**	71.26
Baseline	64.35	67.80	66.03

Table 9.2: Performances on the GloboQuotes dataset.

The results aforementioned are obtained by using the set of candidate authors that are manually annotated in the GloboNotes dataset. We follow the setting used in Fernandes (2012) to allow a fair comparison. On the other hand, the set of candidate quotes given to ESL are generated by simple rules that are part of Fernandes's baseline system. These rules select some segments from the document by applying a sequence of regular expressions. The extracted segments correspond to more than 90% of the quotes in the dataset. However, this heuristic still greatly reduces the number of segments given as input to the ESL system when compared to all possible segments.

Since GloboQuotes has no standard validation set, we use 5-fold cross-validation on the training set to tune ESL meta-parameters. The loss weight C is set to 10 and the number of epochs is 65. We generate templates containing from 2 to 4 basic features and, additionally, include templates by removing the feature at the root node of the decision tree.

10
Coreference Resolution

The CoNLL-2012 Shared Task (Pradhan et al., 2012) is dedicated to the modeling of coreference resolution for multiple languages. The participants are provided with datasets for three languages: Arabic, Chinese and English. These datasets are provided by the OntoNotes project and, besides accurate coreference information, contain various annotation layers such as part-of-speech (POS) tagging, syntax parsing, named entities (NE) and semantic role labeling (SRL). The shared task consists in the automatic identification of coreferring mentions of entities and events, given predicted information on other OntoNotes layers. The official ranking for this task is given by the mean score on the three languages. We take part in the CoNLL-2012 Shared Task *closed track*, in which training data is restricted to the information provided by the shared task organizers. We propose a language-independent approach based on ESL and submit its results to the shared task (Fernandes et al., 2012b). The developed systems obtain the very best performance among all participants. In this chapter, we describe this ESL application.

Coreference resolution consists in identifying mention clusters in a document. Mentions are textual references to real world entities, like people, companies or places. In Figure 10.1, we present an illustrative document with nine highlighted mentions. In a given document, mentions that refer to the

North Korea$_{a_1}$ opened **its**$_{a_2}$ doors to the **U.S.**$_{b_1}$ today, welcoming **Secretary of State Madeleine Albright**$_{c_1}$. **She**$_{c_2}$ says **her**$_{c_3}$ visit is a good start. The **U.S.**$_{b_2}$ remains concerned about **North Korea's**$_{a_3}$ missile development program and **its**$_{a_4}$ exports of missiles to Iran.

Figure 10.1: Document with nine highlighted mentions that refer to three different entities: North Korea is referenced by mentions $\{a_1, a_2, a_3, a_4\}$; the U.S. is referenced by $\{b_1, b_2\}$; and Madeleine Albright by $\{c_1, c_2, c_3\}$. The letter in the mention subscript identifies its entity cluster and the number uniquely identifies the mention within its cluster.

same entity are called *coreferring mentions* and form an *entity cluster*. In the example, the letter in a mention subscript indicates its entity cluster and the number uniquely identifies the mention within its cluster. There are three entity clusters in the example that are related to the following real

world entities: North Korea, which is identified by the letter a; the United States, which is identified by b; and Madeleine Albright, which is identified by c. The coreference resolution task is to identify entity mentions in a given document and to cluster the coreferring mentions. Clusters that comprise only one mention are ignored. For instance, in the example, the mention Iran is ignored.

The remainder of this chapter is organized as follows. In Section 10.1, we formalize the coreference resolution task. In this chapter, we propose a novel structure learning modeling for this task. In Section 10.2, we present the feature factorization used in this modeling. The resulting prediction problem is equivalent to the maximum branching problem, just as for dependency parsing. However, we use a slightly different loss function. These aspects are discussed in Section 10.3. We describe the basic features provided to ESL in Section 10.4. We apply our ESL-based coreference modeling to three very different languages, since our modeling is highly language independent. Nevertheless, some datasets lack basic features and we need to adapt some parts of the systems. In Section 10.5, we describe these adaptations and some additional preprocessing procedures. Finally, in Section 10.6, we present our empirical results.

10.1 Task Formalization

Regarding our ESL modeling, the input for the coreference resolution task is a set of mentions $\boldsymbol{x} = \{x_1, \ldots, x_N\}$ within a document. The task is to cluster the coreferring mentions together, that is, mentions that are references to the same entity are in the same cluster. A feasible output is then a set of non-overlapping clusters $\boldsymbol{y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\}$, where $\boldsymbol{y}_i \subset \boldsymbol{x}$; $\boldsymbol{y}_i \cap \boldsymbol{y}_j = \emptyset$, for $i, j \in \{1, \ldots, K\}$ and $i \neq j$; and K is unknown. Additionally, *singleton* mentions are ignored. A singleton is a unique mention to its entity in the input document.

10.2 Feature Factorization

Usually, coreference systems use features that depend on pairs of mentions (x_i, x_j). We follow this idea, but we introduce a novel modeling for coreference resolution. Most clustering metrics lead to NP-hard optimization problems. Hence, we assume that an entity cluster is represented by a rooted tree. A directed edge (i, j) in this tree indicates that x_j is a reference to the more general mention x_i.

10.2.1 Coreference Trees

We introduce *coreference trees* to represent clusters of coreferring mentions. A coreference tree is a rooted tree whose nodes are the coreferring mentions and arcs represent *some* coreference relation between mentions. In Figure 10.1, we present a document with nine highlighted mentions comprising three clusters. One plausible coreference tree for the cluster $\{a_1,a_2,a_3,a_4\}$ is presented in Figure 10.2. We are not really concerned about the semantics

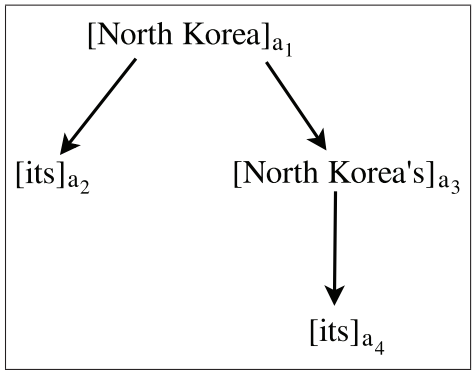


Figure 10.2: Coreference tree for the cluster a in Figure 10.1.

underlying coreference trees, since they are just auxiliary structures for the clustering task. However, we argue that this concept is linguistically plausible, since usually there is indeed a specific-to-general relation between two coreferring mentions. Observing the aforementioned example, one may agree that mention a_3 (North Korea's) is more likely to be associated with mention a_1 (North Korea) than with mention a_2 (its), even considering that a_2 and a_3 are closer to each other than a_1 and a_3, in the document text.

For a given document, we have a forest of coreference trees, one tree for each entity cluster. However, for the sake of simplicity, we link the root node of every coreference tree to an *artificial* root node, obtaining the *document tree*. In Figure 10.3, we depict a document tree for the text in Figure 10.1.

10.2.2 Latent Structure Learning

Coreference trees are not given in the training data. Thus, we assume that these structures are *latent* and make use of the latent structure perceptron (Fernandes and Brefeld, 2011; Yu and Joachims, 2009) to train our models. We introduced this algorithm earlier in Figure 2.3. Here, we describe its application to coreference resolution by using coreference trees. We decompose the original predictor into two predictors, namely the *latent predictor* $F_h(\boldsymbol{x};\boldsymbol{w})$ and the

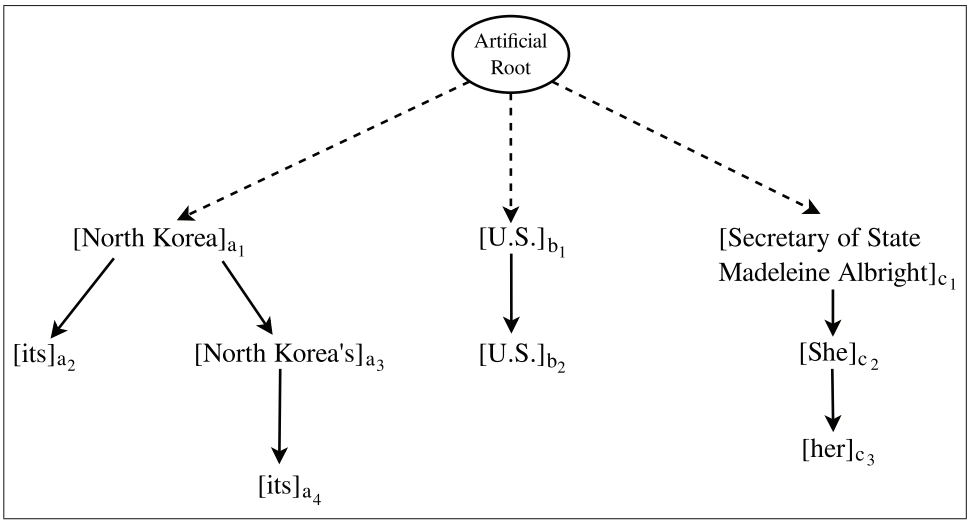


Figure 10.3: Document tree with three coreference trees that corresponds to the text in Figure 10.1. Dashed lines indicate artificial arcs.

target predictor $F_y(\boldsymbol{x}, \boldsymbol{h})$. The latent predictor is defined as

$$F_h(\boldsymbol{x}; \boldsymbol{w}) = \arg \max_{\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x})} \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{h}) \rangle,$$

where $\mathcal{H}(\boldsymbol{x})$ is the set of feasible document trees for $\boldsymbol{x}$ and $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{h})$ is the joint feature vector representation for the mentions $\boldsymbol{x}$ and the document tree $\boldsymbol{h}$. Hence, the latent predictor finds a maximum scoring rooted tree over the given mentions $\boldsymbol{x}$, where a tree score is given by a linear function over its features. $F_y(\boldsymbol{x}, \boldsymbol{h})$ is a straightforward procedure that creates a cluster for each subtree connected to the artificial root node in the document tree $\boldsymbol{h}$. Then, for a given input $\boldsymbol{x}$, a complete prediction is given by $F_y(\boldsymbol{x}, F_h(\boldsymbol{x}; \boldsymbol{w}))$.

As one can observe, in this application of the latent SPerc, we do not use the target model $\boldsymbol{w}_y$ introduced in Section 2.3, since the target predictor $F_y(\boldsymbol{x}, \boldsymbol{h})$ predicts an output based exclusively on the latent structure $\boldsymbol{h}$. Thus, in this chapter, the model $\boldsymbol{w}$ corresponds to the latent model $\boldsymbol{w}_h$ presented in Chapter 2.

In Figure 10.4, we depict the latent structure perceptron algorithm for the mention clustering task. Likewise its binary counterpart (Rosenblatt, 1957), the structure perceptron is an online algorithm that iterates through the training set. For each training instance, it performs two major steps: (i) a prediction for the given input using the current model; and (ii) a model update based on the difference between the predicted and the ground truth outputs. The latent SPerc performs an additional step to predict the latent ground truth $\tilde{\boldsymbol{h}}$ by using a specialization of the latent predictor.

Golden coreference trees are not available, however, during training, for

$\boldsymbol{w}_0 \leftarrow \mathbf{0}$
$t \leftarrow 0$
while no convergence
  for each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
    $\tilde{\boldsymbol{h}} \leftarrow \arg\max_{h \in \mathcal{H}(\boldsymbol{x},\boldsymbol{y})} \langle \boldsymbol{w}_t, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{h}) \rangle$
    $\hat{\boldsymbol{h}} \leftarrow \arg\max_{h \in \mathcal{H}(\boldsymbol{x})} \langle \boldsymbol{w}_t, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{h}) \rangle + \ell(\boldsymbol{h}, \tilde{\boldsymbol{h}})$
    $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{\Phi}(\boldsymbol{x}, \tilde{\boldsymbol{h}}) - \boldsymbol{\Phi}(\boldsymbol{x}, \hat{\boldsymbol{h}})$
    $t \leftarrow t + 1$
$\boldsymbol{w} \leftarrow \frac{1}{t} \sum_{k=1}^{t} \boldsymbol{w}_k$

Figure 10.4: Latent structure perceptron algorithm.

an input $\boldsymbol{x}$, we have the golden clustering $\boldsymbol{y}$. Thus, we predict the *constrained document tree* $\tilde{\boldsymbol{h}}$ for the training instance $(\boldsymbol{x}, \boldsymbol{y})$ using a specialization of the latent predictor – the *constrained latent predictor* – that makes use of $\boldsymbol{y}$. The constrained predictor finds the maximum scoring document tree among the *constrained document tree set* $\mathcal{H}(\boldsymbol{x}, \boldsymbol{y}) \subset \mathcal{H}(\boldsymbol{x})$, which includes all rooted trees of $\boldsymbol{x}$ that follow the correct clustering $\boldsymbol{y}$. That is, a constrained tree $\boldsymbol{h} \in \mathcal{H}(\boldsymbol{x}, \boldsymbol{y})$ comprises only arcs between coreferring mentions – according to $\boldsymbol{y}$ – plus one arc from the artificial node to each cluster. In that way, the constrained predictor guarantees that $F_y(\tilde{\boldsymbol{h}}) = \boldsymbol{y}$, for any $\boldsymbol{w}$. The constrained tree is then used as the ground truth on each iteration. Therefore, the model update is determined by the difference between the constrained document tree and the document tree predicted by the ordinary predictor.

The latent structure perceptron algorithm learns to predict document trees that help to solve the clustering task. Thereafter, for an unseen document $\boldsymbol{x}$, the latent predictor $F_h(\boldsymbol{x}; \boldsymbol{w})$ is employed to produce a predicted document tree $\boldsymbol{h}$ which, in turn, is fed to $F_y(\boldsymbol{x}, \boldsymbol{h})$ to give the predicted clusters.

10.3 Prediction Problem

We decompose the joint feature vector $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{h})$ along coreference tree edges, that is, mention pairs. Thus, the prediction problem is reduced to the maximum branching problem, just as for dependency parsing, and it can be efficiently solved by the Chu-Liu-Edmonds algorithm.

We use a loss function that is similar to the one used for dependency parsing

$$\ell(\boldsymbol{h}, \hat{\boldsymbol{h}}) = \sum_{(i,j) \in \hat{\boldsymbol{h}}; i > 0} \mathbf{1}[(i,j) \notin \boldsymbol{h}] + \sum_{(i,j) \in \hat{\boldsymbol{h}}; i = 0} r \cdot \mathbf{1}[(i,j) \notin \boldsymbol{h}]$$

where $(0, j)$ is an artificial edge and r is a meta-parameter denoted *root loss value*. This loss function just counts how many predicted edges are not present in the constrained document tree. Additionally, for arcs from the artificial root node, we use a different loss value r.

10.4 Basic Features

We use 70 basic features to describe a candidate edge. All of them give hints on the coreference strength of individual edges. These features provide lexical, syntactic, semantic, and positional information. They have been adapted from previously proposed features dos Santos and Carvalho (2011); Sapena et al. (2010); Ng and Cardie (2002). All features have been transformed into categorical, even the integer ones.

In Table 10.1, we briefly describe the set of basic features used in our system. The *Id* column identifies each feature. The *Type* column indicates the value type of each feature, such as boolean (*yes*, *no*) or ternary (*yes*, *no*, *not applicable*). The # column indicates how many basic features correspond to each description.

10.5 Data Preparation

In this section, we present some specific procedures that are performed before the application of the ESL system to coreference resolution.

10.5.1 Mention Detection

The CoNLL-2012 shared task datasets do not explicitly provide entity mentions; the system needs to detect them. For each document, we generate a list of candidate mentions using the strategy of dos Santos and Carvalho (2011). The basic idea is to use all noun phrases and, additionally, pronouns and named entities, even if they are inside larger noun phrases. We do not include verbs as mentions.

10.5.2 Coreference Arcs Generation

The input for the prediction problem is a graph whose nodes are the mentions in a document. Ideally, we could consider the complete graph for each document, thus every mention pair would be an option for building the document tree. However, since the total number of mentions is huge and a big

Id	Description	Type	#
	Lexical Features		*25*
L1	Head word of x_i (x_j)	word	2
L2	String matching of x_i and x_j	boolean	1
L3	String matching of the head words of x_i and x_j	boolean	1
L4	Both x_i and x_j are pronouns and their strings match	ternary	1
L5	Both x_i and x_j are *not* pronouns and their string match	ternary	1
L6	Previous and next two words of x_i (x_j)	word	8
L7	Length of x_i (x_j)	integer	2
L8	Edit distance of head words x_i and x_j	integer	1
L9	Edit distance of x_i and x_j after removing determiners	integer	1
L10	x_i (x_j) is a definitive noun phrase	boolean	2
L11	x_i (x_j) is a demonstrative noun phrase	boolean	2
L12	The head word of both x_i and x_j are proper nouns	boolean	1
L13	Both x_i and x_j are proper names and their strings match	ternary	1
L14	Both x_i and x_j are proper names and their head word strings match	ternary	1
	Syntactic Features		*28*
Sy1	POS tag of the head word of x_i (x_j)	POS tag	2
Sy2	Previous and next two POS tags of x_i (x_j)	POS tag	8
Sy3	x_i (x_j) is a pronoun	boolean	2
Sy4	Gender of x_i (x_j), if pronoun	f, m, n/a	2
Sy5	x_i and x_j are both pronouns and agree in gender	ternary	1
Sy6	x_i and x_j are both pronouns and agree in number	ternary	1
Sy7	x_i (x_j) is a proper name	boolean	2
Sy8	x_i and x_j are both proper names	boolean	1
Sy9	Previous and next predicate of x_i (x_j)	verb	4
Sy10	x_i and x_j are pronouns and agree in number, gender and person	ternary	1
Sy11	Noun phrase embedding level of x_i (x_j) in the syntactic parse	integer	2
Sy12	Number of embedded noun phrases in x_i (x_j)	integer	2
	Semantic Features		*13*
Se1	The prediction of the baseline system proposed in dos Santos and Carvalho (2011)	binary	1
Se2	Sense of the head word of x_i (x_j)	sense	2
Se3	Named entity type of x_i (x_j)	NE tag	2
Se4	x_i and x_j have the same named entity	ternary	1
Se5	Semantic role for the previous and next words of x_i (x_j)	SRL tag	4
Se6	Concatenation of semantic roles of x_i and x_j for the same predicate, if they are in the same sentence	(SRL tag)^2	1
Se7	x_i and x_j have the same speaker	ternary	1
Se8	x_j is an alias of x_i	boolean	1
	Positional Features		*4*
P1	Distance between x_i and x_j in number of sentences	integer	1
P2	Distance between x_i and x_j in number of mentions	integer	1
P3	Distance between x_i and x_j in number of person names (applies only for the cases where x_i and x_j are both pronouns or one of them is a person name)	integer	1
P4	One mention is in apposition to the other	boolean	1

Table 10.1: Description of all 70 basic features.

portion of arcs can be easily identified as incorrect, we filter the arcs and, thus, include only candidate mention pairs that are more likely to be coreferent.

We filter arcs by simply adapting the sieves method proposed by Lee et al. (2011) to English coreference resolution. Lee et al. propose a list of handcrafted

rules that are sequentially applied to mention pairs in order to iteratively merge mentions into entity clusters. These rules are denoted *sieves*, since they filter the correct mention pairs. In Lee et al.'s system, sieves are applied from higher to lower precision. However, in our filtering strategy, precision is not a concern and the application order is not important. The objective here is to build a small set of candidate arcs that shows good recall. Additionally, we do not have interest on sieves that are strongly language dependent, since our target is multilingual coreference resolution. We thus select the most general sieves, which can be easily applied to the Arabic and Chinese datasets provided in the CoNLL-2012 shared task.

Given a mention pair (x_i, x_j), where x_i appears before x_j in the text, we create a directed arc (i, j) if at least one of the following conditions holds:

1. *Distance* – The number of mentions between x_i and x_j is not greater than a given parameter.

2. *Alias* – If both mentions are people, check if the head word of one mention is part of the other mention, like Dilma and Dilma Rousseff. If both mentions are organizations, check if the head word of one mention is contained in the other, or if one is the acronym of the other.

3. *Relaxed String Match* – There is a match of both mentions up to their head words.

4. *Head Word Match* – The head word of x_i matches the head word of x_j.

5. *Shallow Discourse* – Test if shallow discourse attributes match for both mentions. For instance, two first person pronouns assigned to the same speaker are considered coreferent.

6. *Pronouns* – Check if x_j is a pronoun and x_i has the same gender, number, speaker and animacy of x_j. For this filter, we use number and gender data provided by Bergsma and Lin (2006).

7. *Pronouns/NE* – Check if x_j is a pronoun and x_i is a compatible pronoun or proper name (named entity).

Sieves 2 to 7 are adapted from Lee et al. (2011). Most of these sieves are relaxed versions of the ones proposed by Lee et al. (2011). Sieve 1 is introduced by us to lift recall, yet avoiding strongly language-dependent sieves.

10.5.3 Language Specifics

Our system can be easily adapted to different languages. In our experiments, only minor changes are needed to train and apply the system to three different languages. The adaptations are due to: (i) lack of input features for some languages; (i) different POS tagsets across datasets; and (iii) creation of static lists of language specific pronouns. The necessary adaptations are restricted to only two preprocessing steps: basic features and coreference arcs generation.

Some input features available in the English dataset are not available in the Arabic nor in the Chinese datasets. The Arabic dataset does not contain named entity, semantic role labeling and speaker features. Therefore, for Arabic, we do not derive the following basic features: Sy9, Se3, Se4, Se5, Se6, Se7, and P3. For Chinese, information related to named entity is not provided. Thus, we do not derive the following basic features: Se3, Se4, and P3. Additionally, the Chinese dataset uses a different POS tagset. Hence, some mappings are used during the basic feature derivation stage.

The lack of input features for Arabic and Chinese also impact the sieve based arc generation. For Chinese, we do not use sieve 6, and, for Arabic, we only use sieves 1, 3, 4 and 7. Sieve 7 is not used for the English dataset, since it is a specialization of sieve 6. The first sieve threshold is 4 for Arabic and Chinese, and 8 for English.

In the arc generation and basic feature derivation steps, our system makes use of static lists of language specific pronouns. In our experiments, we use the POS tagging information and the golden entity clusters to automatically extract these pronoun lists from training data.

Our system submitted to the CoNLL-2012 Shared Task ignores arcs linking *nested* mentions. While this kind of mentions are never coreferent in Arabic nor in English, the Chinese datasets include many nested coreferring mentions. Hence, in the latest version of our system, we include such arcs for the Chinese language.

10.5.4 EFG Setting

We experiment with different template sets for each language. The difference between these sets is the training data given as input to EFG. We obtain better results when merging different template sets. For the English language, it is better to use a set of 196 templates obtained by merging the output of two independent EFG executions. These two runs are fed with

training datasets comprising: (a) mention pairs produced by sieves 2 to 6; and (b) mention pairs produced by all sieves. For Chinese and Arabic, it is better to use template sets generated specifically for these languages and merge them with the template set (a), generated for the English language. The final set for Chinese comprises 197 templates, while the final set for Arabic comprises 223. All these templates conjoin from two to seven basic features.

10.5.5 Evaluation Metrics

Evaluating coreference systems is a hard task. The main issue is that coreference information is highly faceted and the value of each facet varies a lot from one application to another. Thus, when reporting and comparing coreference performances, it is really hard to define *one* metric that fits all purposes. Therefore, we follow the methodology proposed in the CoNLL-2012 Shared Task to assess our systems, since it combines three of the most popular metrics. The metrics used are the link based MUC metric (Vilain et al., 1995), the mention based B^3 metric (Bagga and Baldwin, 1998) and the entity based $CEAF_e$ metric (Luo, 2005). All these metrics are based on precision and recall measures, which are combined to give an F-score value. The mean F-score of these three metrics gives a unique score for each language. Additionally, when appropriate, the *official* CoNLL-2012 Shared Task score is reported, which is the average of the F-scores for all languages. We denote this metric as *CoNLL score*.

Another important aspect of coreference evaluation is mention matching. Some methodologies, like the ones used in MUC or ACE evaluations, consider approximate matching of mention spans. However, the CoNLL-2012 Shared Task evaluation considers only exact span matching. We use the latter in our performance measures. In fact, the experimental results reported in this work are generated by the official CoNLL-2012 Shared Task evaluation scripts.

10.6 Empirical Results

In this section, we present five sets of empirical findings on the CoNLL-2012 Shared Task datasets. Namely, (i) we show our system overall quality, that is, the best one for Arabic, Chinese and English; (ii) we assess the EFG impact, showing that it significantly improves the resulting system quality; (iii) we assess the root loss value impact, also showing that it significantly improves system quality; (iv) we show that by enhancing our Chinese modeling with nested mentions, we achieve state-of-the-art quality

for this language; and (v) we present the supplementary results provided by the shared task organizers. These empirical findings highlight the main contributions of this work regarding multilingual unrestricted coreference resolution on OntoNotes.

10.6.1 State-of-the-art Systems

In Table 10.2, we present per-language and CoNLL scores of the best performing systems on the CoNLL-2012 test sets. The first row of this table

Reference	AR	CH	EN	CoNLL Score
This work	**54.22**	**62.87**	**63.37**	**60.15**
Fernandes et al. (2012a)	**54.22**	58.49	**63.37**	58.69
Björkelund and Farkas (2012)	53.55	59.97	61.24	58.25
Chen and Ng (2012)	47.13	62.24	59.69	56.35

Table 10.2: State-of-the-art systems for multilingual unrestricted coreference resolution in OntoNotes. Performances on the CoNLL-2012 Shared Task test sets.

corresponds to the last version of our system and the second row corresponds to our official entry in the CoNLL-2012 Shared Task. The difference between these two versions is the inclusion of candidate arcs linking *nested* mentions for the Chinese language. By including such arcs, the score increases almost 4.5 points for that language.

The last two rows of Table 10.2 correspond to the competitors that are ranked second Björkelund and Farkas (2012) and third Chen and Ng (2012) in the shared task. Our system obtains a remarkable performance on the English language, outperforming the runner-up by more than two points. We also achieve the highest performance on Arabic and Chinese.

The detailed performance of our systems is presented in Table 10.3, where we report recall, precision and F-score for all metrics and languages considered in the CoNLL-2012 Shared Task. We can observe that the mean scores on Chinese and English are similar. On the other hand, the performance on the Arabic language is much lower. Given the smaller size of the Arabic training corpus, this variation is expected.

Lang	MUC			B^3			$CEAF_e$			Mean
	R	P	F	R	P	F	R	P	F	
Arabic	43.63	49.69	46.46	62.70	72.19	67.11	52.49	46.09	49.08	**54.22**
Chinese	59.20	71.52	64.78	67.17	80.55	73.25	57.46	45.20	50.59	**62.87**
English	65.83	75.91	70.51	65.79	77.69	71.24	55.00	43.17	48.37	**63.37**
									CoNLL Score	**60.15**

Table 10.3: Detailed performance of our system on the CoNLL-2012 Shared Task test sets.

10.6.2
Entropy Guided Feature Generation

In this work, we employ entropy guided feature generation to automatically generate non-linear features that conjoin the used 70 basic features. In this section, we compare our EFG-based system with a system trained with basic features alone. It is important to notice that among these 70 basic features there are several complex features. Some of these features are even conjunctions of other simpler basic features, and others provide complex task dependent information, like head words and agreement on number and gender, for instance. These 70 basic features were manually generated by domain experts and encode valuable coreference information.

In Table 10.4, we present the performances of four systems on the English development set. In the upper half of this table, we report the performance of our EFG-based system (first row) and the performance of a model trained with basic features alone (second row). We can notice that the EFG system outperforms the baseline by 7.31 points. Moreover, EFG consistently outperforms the baseline on all metrics.

Basic Feats.	**EFG**	**MUC**			**B^3**			**$CEAF_e$**			**Mean**
		R	P	F_1	R	P	F_1	R	P	F_1	
70	Yes	61.34	75.71	67.77	62.94	79.59	70.29	56.23	40.36	46.99	**61.68**
	No	51.32	73.28	60.37	54.85	78.06	64.43	50.87	30.71	38.30	**54.37**
54	Yes	60.86	74.82	67.12	62.50	78.83	69.72	54.53	39.46	45.79	**60.88**
	No	36.65	73.44	48.90	45.25	82.26	58.38	49.97	22.09	30.64	**45.97**

Table 10.4: EFG effect on system performance for the English development set.

We perform another experiment to assess EFG. We remove 16 basic features out of the 70 original ones and perform the same experiment as before. That is, we evaluate an EFG-based system trained with the remaining 54 basic features and compare it to another system trained with the same 54 basic features alone. Namely, we remove the following basic features: L2, L3, L4, L5, L8, L9, L12, L13, L14, Sy5, Sy8, Sy10, Se1, Se4, Se7, P4. In the lower half of Table 10.4, we present the performance of these two systems. We can see that, while the EFG-based system performance (third row) drops only 0.8 point when the 16 features are removed, the performance of the baseline system (fourth row) drops impressive 8.4 points. The difference between the two systems doubles in respect to the difference when using all 70 basic features. These findings highlight two important points. First, the removed features are very informative. Moreover, EFG is able to almost completely overcome the omission of these informative features by automatically generating conjunctions of the remaining 54 basic features.

10.6.3 Root Loss Value

Just as some coreference metrics can be more important than others for some applications, precision and recall have different values for applications. Specifically for the CoNLL score – which is based on the $F_{\beta=1}$ score – the balance between precision and recall is important. For this reason, we introduce one important parameter in our system: the *root loss value*. This parameter specifies a different loss function value for outgoing arcs in the artificial root node. Observe that, in a document tree, each arc from the root node corresponds to a cluster. The effect of a root loss value larger than one is to reduce the creation of new clusters, stimulating larger clusters. Therefore, one can use this parameter to adjust the balance between precision and recall.

In the upper half of Table 10.5, we present our system performances on the development sets when we set this parameter to one, which is equivalent to not use this parameter at all. We can notice that in this case recall and precision have very distinct values, lowering the F-score values. Using the

Root Loss	**Lang**	**MUC**			**B^3**			**$CEAF_e$**			**Mean**
		R	P	F	R	P	F	R	P	F	
Off	Arabic	34.18	58.85	43.25	50.61	82.13	62.63	57.37	33.75	42.49	**49.45**
	Chinese	49.17	76.03	59.72	58.16	86.33	69.50	57.56	34.38	43.05	**57.42**
	English	62.75	77.41	69.31	63.88	81.34	71.56	57.46	41.08	47.91	**62.92**
										CoNLL Score	**56.59**
On	Arabic	43.00	47.87	45.30	61.41	70.38	65.59	49.42	44.19	46.66	**52.52**
	Chinese	54.40	68.19	60.52	64.17	78.84	70.76	51.42	38.96	44.33	**58.54**
	English	64.88	74.74	69.46	66.53	78.28	71.93	54.93	43.68	48.66	**63.35**
										CoNLL Score	**58.14**

Table 10.5: Root loss value effect on development set performances.

development sets for tuning, we set the root loss value to 6, 2 and 1.5 for Arabic, Chinese and English, respectively. In the lower half of Table 10.5, we present the performances when we use these values for the root loss value parameter. We can observe that this parameter really causes a better balancing between precision and recall, consequently increasing the F-score values. Its effect is accentuated on Arabic and Chinese, since the unbalancing issue is worse on these languages. The increase in the CoNLL score is over 1.5 point.

10.6.4 Chinese Nested Mentions

Nested noun phrases are very common. For instance, the noun phrase the smart boy includes the nested noun phrase boy. Whether to consider these two noun phrases as coreferring mentions or only consider the longer noun phrase as a mention is an annotation design choice. OntoNotes mostly consider only the

longer noun phrase. However, in many Chinese documents, nested mentions are annotated as coreferring. Thus, in this work, we evaluate the effect of whether arcs linking nested mentions are considered or not. In Table 10.6, we present the detailed results when such arcs are ignored (first row) and when they are included (second row). To consider these arcs remarkably increases our system score by almost 4 points on the Chinese language.

Nested Mentions	**MUC**			**B^3**			**CEAF$_e$**			**Mean**
	R	P	F_1	R	P	F_1	R	P	F_1	
Yes	60.35	70.56	65.05	67.37	79.49	72.93	55.15	44.94	49.52	**62.50**
No	54.40	68.19	60.52	64.17	78.84	70.76	51.42	38.96	44.33	**58.54**

Table 10.6: Effect whether nested coreferring mentions are considered or not for the Chinese language.

10.6.5 Supplementary Results

We report in Table 10.7 the supplementary results provided by the CoNLL-2012 Shared Task organizers on the test sets. These additional

Lang	**Config**	**MUC**			**B^3**			**CEAF$_e$**			**Mean**
		R	P	F_1	R	P	F_1	R	P	F_1	
AR	A/A	43.63	49.69	46.46	62.70	72.19	67.11	52.49	46.09	49.08	**54.22**
	A/GB	45.18	47.39	46.26	64.56	69.44	66.91	49.73	47.39	48.53	**53.90**
	A/GM	57.25	76.48	65.48	60.27	79.81	68.68	72.61	46.00	56.32	**63.49**
	G/A	46.38	51.78	48.93	63.53	72.37	67.66	52.57	46.88	49.56	**55.38**
	G/GB	46.38	51.78	48.93	63.53	72.37	67.66	52.57	46.88	49.56	**55.38**
	G/GM	56.89	76.27	65.17	60.07	80.02	68.62	72.24	45.58	55.90	**63.23**
CH	A/A	52.69	70.58	60.34	62.99	80.57	70.70	53.75	37.88	44.44	**58.49**
	A/GB	58.76	71.46	64.49	66.62	79.88	72.65	54.09	42.02	47.29	**61.48**
	A/GM	61.64	90.81	73.43	63.55	89.43	74.30	72.78	39.68	51.36	**66.36**
	G/A	59.35	74.49	66.07	66.31	81.43	73.10	55.97	41.50	47.66	**62.28**
	G/GB	59.35	74.49	66.07	66.31	81.43	73.10	55.97	41.50	47.66	**62.28**
	G/GM	61.70	91.45	73.69	63.57	89.76	74.43	72.84	39.49	51.21	**66.44**
EN	A/A	65.83	75.91	70.51	65.79	77.69	71.24	55.00	43.17	48.37	**63.37**
	A/GB	64.92	77.53	70.67	64.25	78.95	70.85	56.48	41.69	47.97	**63.16**
	A/GM	70.69	91.21	79.65	65.46	85.61	74.19	74.71	42.55	54.22	**69.35**
	G/A	67.73	77.25	72.18	66.42	78.01	71.75	56.16	44.51	49.66	**64.53**
	G/GB	65.65	78.26	71.40	64.36	79.09	70.97	57.36	42.23	48.65	**63.67**
	G/GM	71.18	91.24	79.97	65.81	85.51	74.38	74.93	43.09	54.72	**69.69**

Table 10.7: Supplementary results on the test sets with different configurations (Config) for parse quality and mention candidates (parse/mentions). Parse quality can be automatic (A) or golden (G); and mention candidates can be automatically identified (A), golden mention boundaries (GB) or golden mentions (GM).

experiments investigate two key aspects of any coreference resolution system: the parse feature and the mention candidates that are given to the clustering procedure. In these results, we alternate the parse feature between the official

automatic parse (A in the results table) and the *golden* parse from OntoNotes (G). Regarding mention candidates, we use three different strategies: automatic mentions (A), golden mention boundaries (GB) and golden mentions (GM). Automatic mentions are the ones detected by our system. Golden mention boundaries comprise all noun phrases in the *golden* parse tree, even when the automatic parse is used as input feature. Golden mentions are all non-singleton mentions, i.e., all mentions that take part in some entity cluster. It is important to notice that golden mention information is much stronger than just golden boundaries.

By observing Table 10.7, it is clear that the most beneficial information is golden mentions (compare A/GM to A/A rows, for each language). The mean F-score over all languages when using golden mentions is almost 8 points higher than the official score. These results are not surprising since to identify non-singleton mentions accounts to a significant part of the final task. Golden mention boundaries (A/GB) increase the Chinese score by almost 3 points. Conversely, for the other two languages, the results are decreased when this information is given. This is probably due to parameter tuning, since any additional information potentially changes the learning problem and, nevertheless, we use exactly the same three models – one per language – to produce both the official and the supplementary results. One can observe, for instance, that the recall/precision balance greatly varies among the different configurations in these experiments. The golden parse feature (G/A) causes big improvements on all languages, specially Chinese.

11
Conclusions

We propose the entropy-guided structure learning framework that extends the general SL framework by integrating an automatic feature generation approach – the entropy-guided feature generation method that is based on the conditional entropy of local decision variables given input basic features.

We compare EFG with two important alternative feature generation methods, namely manual template generation and polynomial kernel functions. Our empirical results on dependency parsing show that EFG outperforms the best known template set on the Portuguese CoNLL-2006 dataset. We compare EFG with polynomial kernel methods on two text chunking datasets: the Portuguese Bosque dataset and the English CoNLL-2000 dataset. EFG outperforms both. Furthermore, our method presents additional advantages over these two alternative methods. It is faster than kernel methods and avoid the overfitting issue. Compared to manual feature templates, the fact that EFG bypasses domain experts is highly valuable.

We evaluate ESL on nine datasets involving five natural language processing tasks and four different languages. ESL presents state-of-the-art comparable performances on all evaluated datasets. Moreover, it outperforms the previous best performing systems on six datasets, namely the Mac-Morpho dataset for Portuguese part-of-speech tagging, the Bosque dataset for Portuguese text chunking, the GloboQuotes dataset for Portuguese quotation extraction, the CoNLL-2012 Shared Task datasets for Arabic, Chinese, and multilingual coreference resolution. Additionally, on the Portuguese dependency parsing task, we demonstrate the power of ESL by automatically including two basic features in our model, lifting the final performance by around 2.4 points.

We propose a novel modeling for coreference resolution based on latent structure learning. The ESL systems based on this modeling achieve the best results for Arabic and English coreference resolution. Moreover, the ESL coreference systems – for Arabic, Chinese and English – achieve the very first place on the renowned CoNLL-2012 Shared Task. The proposed modeling is

highly language-independent, allowing us to apply it on three very different languages with no more than minor adaptations, which are mainly necessary due to lack of features for some languages.

Koo et al. (2010) and McDonald and Pereira (2006) show significant improvements on the performance of DP systems by extending the first-order model used in this work to include second- and third-order features. As future work, we plan to apply these modelings to dependency parsing and coreference resolution.

For coreference resolution, some authors (Lee et al., 2011) report that features based on partial clusters bring substantial improvements on performance. We also plan to extend our latent modeling in order to include such type of features.

Text chunking and named entity recognition have been recurrently recast as sequence labeling problems. Nevertheless, these tasks require sentence segmentation and, additionally, segment classification. In this way, we can apply our modeling based on weighted interval scheduling to such tasks, just as we have done for quotation extraction. By using a sequence segmentation modeling, we are able to use more meaningful features for these tasks and, consequently, improve performance.

Bibliography

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden Markov support vector machines. In *Proceedings of the International Conference on Machine Learning*, 2003.

Yasemin Altun, Thomas Hofmann, and Ioannis Tsochantaridis. SVM learning for interdependent and structured output spaces. In *Machine Learning with Structured Outputs*, 2007.

Sandra Aluísio, Jorge Pelizzoni, Ana R. Marchi, Lucélia de Oliveira, Regiana Manenti, and Vanessa Marquiafável. An account of the challenge of tagging a reference corpus for brazilian portuguese. In *Proceedings of the 6th international conference on Computational processing of the Portuguese language*, PROPOR'03, pages 110–117, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40436-8. URL `http://dl.acm.org/citation.cfm?id=1758748.1758769`.

Amit Bagga and Breck Baldwin. Algorithms for scoring coreference chains. In *In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, pages 563–566, 1998.

Shane Bergsma and Dekang Lin. Bootstrapping path-based pronoun resolution. In *Proceedings of ACL2006*, ACL–44, pages 33–40, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

Anders Björkelund and Richárd Farkas. Data-driven multilingual coreference resolution using resolver stacking. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 49–55, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-4503`.

Eric Brill. Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Comput. Linguist.*, 21: 543–565, December 1995. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=218355.218367`.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Natural Language Learning*, pages 149–164, 2006.

Chen Chen and Vincent Ng. Combining the best of two worlds: A hybrid approach to multilingual coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 56–63, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-4504`.

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2002a.

Michael Collins. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, 2002b.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multi-class kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

Thomas G. Dietterich. Machine learning for sequential data: A review. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 2002.

Cícero N. dos Santos and Davi L. Carvalho. Rule and tree ensembles for unrestricted coreference resolution. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 51–55, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W11-1906`.

Cícero N. dos Santos and Ruy L. Milidiú. *Foundations of Computational Intelligence, Volume 1: Learning and Approximation*, volume 201 of *Studies*

in Computational Intelligence, chapter Entropy Guided Transformation Learning, pages 159–184. Springer, 2009a.

Cícero N. dos Santos and Ruy L. Milidiú. Entropy guided transformation learning. In *Foundations of Computational Intelligence (1)*, pages 159–184. Springer, 2009b.

Cícero N. dos Santos, Ruy L. Milidiú, Carlos E. M. Crestana, and Eraldo R. Fernandes. ETL ensembles for chunking, NER and SRL. In *11th International Conference on Computational Linguistics and Intelligent Text Processing, CICLing*, pages 100–112, 2010.

Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

Eraldo R. Fernandes and Ulf Brefeld. Learning from partially annotated sequences. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, Athens, Greece, 2011.

Eraldo R. Fernandes and Ruy L. Milidiú. Entropy-guided feature generation for structured learning of Portuguese dependency parsing. In *Proceedings of the Conference on Computational Processing of the Portuguese Language (PROPOR)*, volume 7243 of *Lecture Notes in Computer Science*, pages 146–156. Springer Berlin / Heidelberg, 2012.

Eraldo R. Fernandes, Ruy L. Milidiú, and Cícero N. dos Santos. Portuguese language processing service. In *18th International World Wide Web Conference*, April 2009a. URL `http://www2009.eprints.org/208/`.

Eraldo R. Fernandes, Bernardo A. Pires, Cícero N. dos Santos, and Ruy L. Milidiú. Clause identification using entropy guided transformation learning. In *Proceedings of the 2009 Seventh Brazilian Symposium in Information and Human Language Technology*, STIL '09, pages 117–124, Washington, DC, USA, 2009b. IEEE Computer Society. ISBN 978-0-7695-3945-4. doi: 10.1109/STIL.2009.10. URL `http://dx.doi.org/10.1109/STIL.2009.10`.

Eraldo R. Fernandes, Carlos E. M. Crestana, and Ruy L. Milidiú. Hedge detection using the relhunter approach. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning — Shared Task*, CoNLL '10: Shared Task, pages 64–69, Stroudsburg, PA, USA, 2010a. Association for Computational Linguistics. ISBN 978-1-932432-84-8. URL `http://dl.acm.org/citation.cfm?id=1870535.1870544`.

Eraldo R. Fernandes, Cícero N. dos Santos, and Ruy L. Milidiú. A machine learning approach to portuguese clause identification. In *Proceedings of the 9th international conference on Computational Processing of the Portuguese Language*, PROPOR'10, pages 55–64, Berlin, Heidelberg, 2010b. Springer-Verlag. ISBN 3-642-12319-8, 978-3-642-12319-1. doi: 10.1007/978-3-642-12320-7_8. URL `http://dx.doi.org/10.1007/978-3-642-12320-7_8`.

Eraldo R. Fernandes, Ruy L. Milidiú, and Raúl P. Rentería. RelHunter: a machine learning method for relation extraction from text. *Journal of the Brazilian Computer Society*, 16:191–199, 2010c. ISSN 0104-6500. URL `http://dx.doi.org/10.1007/s13173-010-0018-y`.

Eraldo R. Fernandes, Cícero dos Santos, and Ruy L. Milidiú. Latent structure perceptron with feature induction for unrestricted coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 41–48, Jeju Island, Korea, July 2012a. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-4502`.

Eraldo R. Fernandes, Cícero N. dos Santos, and Ruy L. Milidiú. Latent structure perceptron with feature induction for unrestricted coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 41–48, Jeju Island, Korea, July 2012b. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-4502`.

William P. D. Fernandes. Quotation extraction for portuguese. Master's thesis, PUC-Rio, Rio de Janeiro, 2012.

Winthrop N. Francis and Henry Kučera. *Frequency analysis of english usage: Lexicon and grammar*. Houghton Mifflin, Boston, 1982.

Cláudia Freitas, Paulo Rocha, and Eckhard Bick. Floresta Sintá(c)tica: Bigger, thicker and easier. In António Teixeira, Vera Lúcia Strube de Lima, Luís Caldas de Oliveira, and Paulo Quaresma, editors, *Computational Processing of the Portuguese Language*, volume 5190 of *Lecture Notes in Computer Science*, pages 216–219, 2008.

Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976. ISSN 0020-0190. doi: 10.1016/0020-0190(76)90095-8.

Thorsten Joachims. Learning to align sequences: A maximum-margin approach. Technical report, Cornell University, 2003.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1288–1298, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1870658.1870783`.

Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, NAACL'01, pages 1–8, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1073336.1073361. URL `http://dx.doi.org/10.3115/1073336.1073361`.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL Shared Task 2011, pages 28–34, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 9781937284084. URL `http://dl.acm.org/citation.cfm?id=2132936.2132938`.

Xiaoqiang Luo. On coreference resolution performance metrics. In *In Proc. of HLT/EMNLP*, pages 25–32. URL, 2005.

David McAllester, Tamir Hazan, and Joseph Keshet. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, 2011.

Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *In Proc. of EACL*, pages 81–88, 2006.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL'05, pages 91–98, 2005.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, pages 216–220, 2006.

Ruy L. Milidiú, Cícero N. dos Santos, and Julio C. Duarte. Phrase chunking using entropy guided transformation learning. In *Proceedings of ACL2008*, Columbus, Ohio, 2008.

Ruy L. Milidiú, Carlos E. M. Crestana, and Cícero N. dos Santos. A token classification approach to dependency parsing. *Information and Human Language Technology, Brazilian Symposium in*, 0:80–88, 2009. doi: http://doi.ieeecomputersociety.org/10.1109/STIL.2009.29.

Vincent Ng and Claire Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 104–111, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1073083.1073102. URL `http://dx.doi.org/10.3115/1073083.1073102`.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 221–225, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1596276.1596318`.

Albert B. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, 1962.

Sameer Pradhan, Lance Ramshaw, Mitchell Marcus, Martha Palmer, Ralph Weischedel, and Nianwen Xue. Conll-2011 shared task: Modeling unrestricted coreference in ontonotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning Shared Task*, pages 1–27, Portland, USA, 2011. ACL.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 1–40, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W12-4501`.

John R. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1 edition, 1992.

Frank Rosenblatt. The Perceptron – a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory, 1957. Report 85-460-1.

Erik F. T. K. Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: chunking. In *Proceedings of the 2nd workshop on Learning language in*

logic and the 4th conference on Computational natural language learning - Volume 7, ConLL '00, pages 127–132, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1117601.1117631. URL http://dx.doi.org/10.3115/1117601.1117631.

Emili Sapena, Lluís Padró, and Jordi Turmo. Relaxcor: A global relaxation labeling approach to coreference resolution. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 88–91, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. URL /brokenurl#http://publication.wilsonwong.me/load.php?id=233281657.

Jiang Su and Harry Zhang. A fast decision tree learning algorithm. In *Proceedings of the 21st National Conference on Artificial intelligence*, pages 500–505, 2006.

Robert E. Tarjan. Finding optimum branchings. *Networks*, 7:25–25, 1977.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning*, 2004.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.

Marc Vilain, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman. A model-theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message understanding*, MUC6 '95, pages 45–52, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics. ISBN 1-55860-402-2. doi: 10.3115/1072399.1072405. URL http://dx.doi.org/10.3115/1072399.1072405.

Jason Weston and Chris Watkins. Multi-class support vector machines. *Pattern Recognition*, (CSD-TR-98-04), 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9594.

Yu-Chieh Wu, Chia-Hui Chang, and Yue-Shi Lee. A general and multi-lingual phrase chunking model based on masking method. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 3878 of *Lecture Notes in Computer Science*, pages 144–155. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-32205-4.

Chun-Nam Yu and Thorsten Joachims. Learning structural SVMs with latent variables. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

Alan L. Yuille and Anand Rangarajan. The concave-convex procedure. *Neural Comput.*, 15(4):915–936, April 2003. ISSN 0899-7667. doi: 10.1162/08997660360581958. URL `http://dx.doi.org/10.1162/08997660360581958`.