Pontifícia Universidade Católica
do Rio de Janeiro

Yanely Milanés Barroso

Structured Learning with Incremental Feature Induction and Selection for Portuguese Dependency Parsing

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Ruy Luiz Milidiú

Rio de Janeiro
March 2016

Pontifícia Universidade Católica
do Rio de Janeiro

Yanely Milanés Barroso

Structured Learning with Incremental Feature Induction and Selection for Portuguese Dependency Parsing

Dissertation 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 Mestre.

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

Prof. Marco Antonio Casanova
Departamento de Informática — PUC-Rio

Prof. Maria Cláudia de Freitas
Departamento de Letras — PUC-Rio

Prof. Marcio da Silveira Carvalho
Coordinator of the Centro Técnico Científico — PUC–Rio

Rio de Janeiro, March 9th, 2016

Yanely Milanés Barroso

Graduated from University of Havana, Cuba in Computer Science. Her research is focused in Machine Learning and Natural Language Processing.

Bibliographic data

Milanés Barroso, Yanely

Structured Learning with Incremental Feature Induction and Selection for Portuguese Dependency Parsing / Yanely Milanés Barroso; advisor: Ruy Luiz Milidiú. — 2016.

92 f. : il. (color.); 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Departamento de Informática, 2016.

Inclui bibliografia.

1. Informática – Teses. 2. aprendizado de máquina supervisionado. 3. processamento de linguagem natural. 4. análise de dependência de português. 5. modelo linear esparso. 6. indução de atributos. I. Milidiú, Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgement

Thanks to my advisor Prof. Ruy Luiz Milidiú for his endless support and encouragement for this accomplishment.

Thanks to Prof. Maria Cláudia de Freitas for her help and the shared knowledge.

Thanks to PUC-Rio and CAPES, for the financial support.

Thanks to the friends of LEARN, for their support and friendship.

To all colleagues, faculty and staff of the Department of PUC-Rio, for the fellowship, encouragement and support.

To my parents, family and friends who supported me even with my absence in family life.

Abstract

Milanés Barroso, Yanely; Milidiú, Ruy Luiz (Advisor). **Structured Learning with Incremental Feature Induction and Selection for Portuguese Dependency Parsing**. Rio de Janeiro, 2016. 92p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Natural language processing requires solving several tasks of increasing complexity, which involve learning to associate structures like graphs and sequences to a given text. For instance, dependency parsing involves learning of a tree that describes the dependency-based syntactic structure of a given sentence. A widely used method to improve domain knowledge representation in this task is to consider combinations of features, called templates, which are used to encode useful information with nonlinear pattern. The total number of all possible feature combinations for a given template grows exponentialy in the number of features and can result in computational intractability. Also, from an statistical point of view, it can lead to overfitting. In this scenario, it is required a technique that avoids overfitting and that reduces the feature set. A very common approach to solve this task is based on scoring a parse tree, using a linear function of a defined set of features. It is well known that sparse linear models simultaneously address the feature selection problem and the estimation of a linear model, by combining a small subset of available features. In this case, sparseness helps control overfitting and performs the selection of the most informative features, which reduces the feature set. Due to its flexibility, robustness and simplicity, the perceptron algorithm is one of the most popular linear discriminant methods used to learn such complex representations. This algorithm can be modified to produce sparse models and to handle nonlinear features. We propose the incremental learning of the combination of a sparse linear model with an induction procedure of non-linear variables in a structured prediction scenario. The sparse linear model is obtained through a modifications of the perceptron algorithm. The induction method is the Entropy-Guided Feature Generation. The empirical evaluation is performed using the Portuguese Dependency Parsing data set from the CoNLL 2006 Shared Task. The resulting parser attains 92.98% of accuracy, which is a competitive performance when compared against the state-of-art systems. On its regularized version, it accomplishes an accuracy of 92.83%, shows a striking reduction of 96.17% in the number of binary features and reduces the learning time in almost 90%, when compared to its non regularized version.

Keywords

supervised machine learning; natural language processing; Portuguese dependency parsing; sparse linear model; feature induction.

Resumo

Milanés Barroso, Yanely; Milidiú, Ruy Luiz. **Aprendizado Estruturado com Indução e Seleção Incrementais de Atributos para Análise de Dependência em Português**. Rio de Janeiro, 2016. 92p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O processamento de linguagem natural busca resolver várias tarefas de complexidade crescente que envolvem o aprendizado de estruturas complexas, como grafos e seqüências, para um determinado texto. Por exemplo, a análise de dependência envolve o aprendizado de uma árvore que descreve a estrutura sintática de uma sentença dada. Um método amplamente utilizado para melhorar a representação do conhecimento de domínio em esta tarefa é considerar combinações de atributos usando conjunções lógicas que codificam informação útil com um padrão não-linear. O número total de todas as combinações possíveis para uma conjunção dada cresce exponencialmente no número de atributos e pode resultar em intratabilidade computacional. Também, pode levar a *overfitting*. Neste cenário, uma técnica para evitar o superajuste e reduzir o conjunto de atributos faz-se necessário. Uma abordagem comum para esta tarefa baseia-se em atribuir uma pontuação a uma árvore de dependência, usando uma função linear do conjunto de atributos. Sabe-se que os modelos lineares esparsos resolvem simultaneamente o problema de seleção de atributos e a estimativa de um modelo linear, através da combinação de um pequeno conjunto de atributos. Neste caso, promover a esparsidade ajuda no controle do superajuste e na compactação do conjunto de atributos. Devido a sua flexibilidade, robustez e simplicidade, o algoritmo de perceptron é um método linear discriminante amplamente usado que pode ser modificado para produzir modelos esparsos e para lidar com atributos não-lineares. Propomos a aprendizagem incremental da combinação de um modelo linear esparso com um procedimento de indução de variáveis não-lineares, num cénario de predição estruturada. O modelo linear esparso é obtido através de uma modificação do algoritmo perceptron. O método de indução é *Entropy-Guided Feature Generation.* A avaliação empírica é realizada usando o conjunto de dados para português da *CoNLL 2006 Shared Task*. O analisador resultante alcança 92,98% de precisão, que é um desempenho competitivo quando comparado com os sistemas de estado-da-arte. Em sua versão regularizada, o analizador alcança uma precisão de 92,83%, também mostra uma redução notável de 96,17% do número de atributos binários e, reduz o tempo de aprendizagem em quase 90%, quando comparado com a sua versão não regularizada.

Palavras–chave

aprendizado de máquina supervisionado; processamento de linguagem natural; análise de dependência de português; modelo linear esparso; indução de atributos.

Contents

1 Introduction **11**
1.1 Dependency Parsing 12
1.2 Structured Learning 13
1.3 Feature Generation 13
1.4 Feature Selection 14
1.5 Motivation and Goals 15
1.6 Contributions 16
1.7 Dissertation Organization 17

2 Background **18**
2.1 Dependency Grammars 18
2.2 Dependency Trees 19
2.3 Parsing with Dependency Trees 20
2.4 Chapter Conclusions 26

3 Dependency Parsing **27**
3.1 Auxiliary Predictors 27
3.2 Candidate Edge Filter 30
3.3 Basic Edge Features 33
3.4 Chapter Conclusions 37

4 Structure Perceptron **39**
4.1 Dependency Tree Learning 40
4.2 Structure Perceptron 41
4.3 Large Margin Classifiers 43
4.4 Feature Induction 45
4.5 Sparser Perceptron Models 48
4.6 Chapter Conclusions 51

5 S-IFIS Structured Learning with Incremental Feature Induction and Selection **52**
5.1 Attributes Representation 52
5.2 Learning Algorithm 53
5.3 Experiments Parametrization 56
5.4 Implementation Remarks 57
5.5 Chapter Conclusions 58

6 Empirical Evaluation **60**
6.1 Corpus 60
6.2 Evaluation Metrics 61
6.3 Portuguese Dependency Parsing 62
6.4 Results 64
6.5 Error Analysis 75
6.6 Discussion 76

6.7 Chapter Conclusions 78

7 Conclusions **79**

8 Bibliography **82**

A Portuguese Corpus Part-Of-Speech Tag Set **90**

B Chunk and Clause Tag Sets **91**
B.1 Clause Tags 91
B.2 Chunk Tags 92

1
Introduction

Nowadays, due to remarkable technology achievements, we observe an explosion of big data applications and services. Hence, the analysis of large collections of complex data is imperative. In order to pursue this goal, several *Machine Learning* techniques have been successfully applied to the analysis of large collection of texts, images and sensor measurements. These approaches have been successfully applied in several research fields, such as natural language processing (NLP), computer vision, and speech recognition.

NLP refers to the set of computational techniques that, combined with linguistic knowledge, allow computers to represent and use human knowledge expressed by human language sentences. Unlike programming languages, that are designed to be unambiguous, natural languages are not precise. Hence, NLP tasks are challenging because of this intrinsic ambiguity and due to the presence of linguistics structures that depend on many complex variables.

A distinctive characteristic of this field is that, for each task, it usually exists a competition that establishes a well-defined problem setting, standard corpora and evaluation metrics. The Conference on Natural Language Learning (CoNLL) Shared Tasks are examples of such competitions. Furthermore, they promote a great number of advances in this research area. Therefore, tasks on this field are suitable for empirical evaluation of novel machine learning algorithms.

NLP comprises several problems that involve the learning of complex structures like sequences or graphs. Here, we are interested in Dependency Parsing (DP), which is based on syntactic theories, like dependency grammars, that model dependency relations between the linguistic units of a sentence (68). This task involves the prediction of a tree representing the syntactic structure of a sentence. In this case, the output domain covers all possible dependency trees for a given sentence and, hence, can be modeled as a structured prediction problem. Moreover, following the line of research of the *Laboratorio de Engenharia de Algoritmos e Redes Neurais*[1] (LEARN) research group, we seek to integrate the machine learning approaches to DP previously used by (48), (17) and (13). Next, we define DP in the context of structured prediction and give details regarding the motivations, goals and contributions

[1] http://www.learn.inf.puc-rio.br

of this work.

In our problem statement and techniques review, we follow (48; 17; 13) making free use of their former descriptions. Our emphasis in this document is on reporting our new modeling strategies and the resulting new Portuguese Dependency Parsing system.

1.1 Dependency Parsing

Dependency parsing refers to the task of finding the syntactic structure underlying a sentence. The modern theoretical support for this type of parsing starts with the work of (68). The grammatical theories and formalisms comprised in this tradition share the same basic assumptions, that the syntactic structure of a sentence can be defined as *lexical elements* linked by binary assymetrical relations called *depedencies*.

In order to find the dependencies between the lexical elements of a given sentence, the dependence structure is modeled as a rooted graph, where the nodes represent the lexical items and the edges represent the dependency relations between such elements. Also, most dependency grammars consider additional constrains for this structure, which leads to consider this graph as a rooted tree. Here, we use the term token to refer the lexical elements of the sentence.

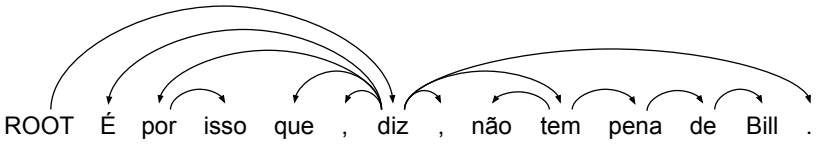

Figure 1.1: Dependency tree for a Portuguese sentence.

Let $x = (x_0 \dots x_n)$ be a sentence, where x_i represents the i-th token for $i \in [0, n]$. The dependency relation is directed by nature. Therefore, the dependency structure is often modeled as a directed acyclic graph. An example is shown in Figure 1.1 for the Portuguese sentence *É por isso que, diz, não tem pena de Bill.* The dependency relations are represented by arrows pointing from the head to the dependent.

A dependency tree y associated with the given sentence x is a directed rooted tree where there is a node i for each token x_i, an artificial node 0 that represents the tree root and several arcs (i, j) that represent dependency relations. When (i, j) is an arc in y, this means that there exists a dependency relation between x_i, the head token, and x_j, the dependent token.

1.2 Structured Learning

Structured Learning (SL) comprises several machine learning techniques that build maps between pairs of input and their corresponding correct output. In a general sense, such outputs can be complex structures like sequences or graphs. These approaches allow a more natural modeling for tasks that involve learning complex and interdependent structures. DP can be successfully modeled as a structured prediction problem because involves the prediction of a dependency tree given an input sentence. The output space is composed by all possible dependency trees.

In this case, the prediction problem $F(x)$ is to predict a dependency tree for an input sentence x, and coincides with finding the maximum-score tree among the valid rooted trees

$$F(x; w) = arg\, max_{y \in Y(x)} s(x, y; w),$$

where $s(x, y; w) = \sum_{(i,j) \in y} s(x, i, j; w)$ is the score of a candidate tree $y \in Y(x)$, i.e., the sum of its edges scores.

Most approaches recast parsing as a maximum branching problem. This prediction problem can be efficiently solved with Chu-Liu-Edmonds algorithm (15; 11). The scoring function is known as the edge-based model.

Linear discriminative models are one of the most popular techniques used in the NLP community to solve DP. The edge-based scoring function is represented with a linear discriminant function with the form of

$$s(x, i, j; w) = \langle w, \Phi(x, i, j) \rangle = \sum_{m}^{M} w_m \phi_m(x, i, j).$$

That is a linear combination of the vector $\Phi(x, i, j)$ of M-real valued feature functions $\phi_i(x, i, j)$ that represents the input x with a weight vector w. In this case, $\langle ., . \rangle$ refers to the scalar product operator. Due its flexibility, robustness and simplicity, the perceptron algorithm is one of the most popular linear discriminant methods used to learn such complex representations.

1.3 Feature Generation

A distinctive characteristic of NLP tasks is the presence of ambiguity. In the context of DP, it is possible to attribute more than one interpretation to a sentence. The problem of disambiguation of sentences in DP is hard because it is difficult for an analyzer choose the most adequate answer. In this case, to

tackle the disambiguation problem, it is necessary a large set of features that describe well the phenomenon of interest.

On the other hand, it is known that in supervised machine learning algorithms the quality of the chosen features describing the problem is critical to the learning process. Thus, the discovery of useful features has a major role.

Feature engineering is the process of using domain knowledge in the design of features. One drawback is that it is carried out manually. DP can be modeled as a structured problem and structured prediction is highly nonlinear on the set of inputs. In this case, are required domain experts, which is expensive in time and resources, to manually derive such patterns. As an alternative, there are several methods for automatic feature generation that can be employed such as nonlinear kernels, feature induction or deep learning approaches, among others.

Usually, these methods can lead to the learning of dense models in a feature space with large dimensions. As a consequence, it can appear the problem of overfitting the training data. Another potential issue to take into account is the introduction of a large number of features that are irrelevant or redundant. In this scenario, to maintain the benefits of feature generation, a mechanism to tackle overfitting and select the most informative features becomes a major task.

1.4 Feature Selection

In machine learning, the process of feature selection refers to the techniques used to select the set of the most relevant features for the learning process. These methods can be applied to increase training speed or to limit space of memory and storage. Also they are employed to obtain a better understanding of the data, or to increase predictive accuracy (25).

When considering linear discriminant models, feature selection can be attained by encouraging the weight vector w to be sparse (49). In this case, sparse linear models focus on the joint estimation of a linear model and the selection of the subset of the most informative features. Basically, they predict outputs by linearly combining a small subset of the features representing the data. Therefore, learning algorithms using this technique are suitable to promote a trade-off between model complexity and generalization power, while they prevent overfitting of the training data.

Due to the large dimensions of feature vectors used in DP, sparse models are an attractive solution, since they produce compact models. The model can be much smaller, since the approach encourages several feature weights to be

set to zero. Also, they are used to boost performance in very high-dimensional datasets and to help on the feature engineering process.

Feature selection and sparseness are currently very active topics in machine learning and statistics. One of the most popular techniques to obtain sparse solutions is the L1-regularization, which is well known to yield sparse solutions, by penalizing the weight vector using the L1-norm.

1.5 Motivation and Goals

Is well known that in supervised machine learning, when considering many input features, a potential problem is the overfitting of the training data. Hence, it is necessary the use of regularization techniques to mitigate this issue (50). This is also the ordinary scenario in the context of DP.

DP is modeled using a high dimensional feature space to describe the phenomenon. Usually, features are binary functions, each one indicating the presence of a certain condition. A distinctive characteristic here is that the instances are encoded with highly sparse vectors, since only a few features are active in each instance. Also, when using feature induction to encode nonlinear information, the dimension of the feature space is augmented dramatically. This leads to another critical issue, namely: the introduction of a large number of irrelevant features (48). When considering feature induction, we want to control a potential overfitting of the training data, produced by a large dimensional feature space. Therefore, we would like to select the set of features that should be used in the prediction model.

Main Goal

In order to mitigate the problems described above and following the line of research of the LEARN lab, we propose to integrate and enhance the approches reported by (48; 13; 17), as stated next.

> To build a graph-based dependency parser, with generalization power similar to the state-of-art analyzers, that uses a reduced amount of attributes. This parser shall incrementally learn a sparse linear model combined with a nonlinear features induction procedure.

Specific Goals

To accomplish the main goal proposed above, we state the following six specific subgoals:

1. to divide the dependency parsing task into less complex subtasks that predict the basic informations about the head of a given a dependent token: both its Part-Of-Speech tag and side (13);

2. to use the predicted information about the head of a given token, in order to identify its position on the sentence, in a structured context;

3. to adapt the incremental scheme proposed by (48) for the structured scenario of learning dependency trees, and use a structured perceptron as the main classifier;

4. to perform the induction procedure, using the Entropy-Guided Feature Generation algorithm (17; 19; 60; 61; 47);

5. to perform the selection of the most informative features, using the Structured Sparse Perceptron (23);

6. to empirically evaluate and compare the proposed solution with state-of-art systems, using the Portuguese data set and the evaluation metric provided by the CoNLL Shared Task 2006 (8; 1; 22).

1.6 Contributions

The main contributions of this dissertation are:

– the combination of multiclass auxiliary predictors as filters of non promising candidate arcs to improve the performance of the learning algorithm;
– an implementation based on the Incremental Feature Induction and Selection framework proposed by (48) for the scenario of structured prediction;
– the implementation of the Entropy-Guided Structured Learning Framework proposed by (17) with Structured Sparse Perceptron and dropout technique;
– a parser with competitive performance when compared against the state-of-art that attains 92.96% of accuracy when using the standard UAS metric.

1.7 Dissertation Organization

The remainder of this document is organized as follows. In chapter 2, we review the background and related work on the dependency parsing task. In chapter 3, we give the modeling details of our approach to parsing. In chapters 4 and 5, we describe our proposed learning system. Next, in chapter 6, we present the empirical evaluation of our approach. Finally, in chapter 7, we draw our conclusions and comment on interesting future work.

2
Background

Dependency parsing is a type of syntactic analysis for sentences in natural languages. This chapter is devoted to review the theoretical foundations that serve to define such analysis in terms of the dependencies relations between the words of an input sentence and to define the structure that serves for the representation of such relations. Also, we review the work that have been made to solve this task with main focus on data-driven approaches, particularly in the subcategory of graph-based parsers.

We based our background overview mainly on (24; 42; 45) for parsing systems overview and on (52) for theoretical foundations of dependency grammars and dependency parsing.

For a better comprehension, the remainder of this chapter is organized as follows. First we present the basic concepts of dependency parsing, which are dependency grammars and dependency structures. Finally we review the related work on parsing, making special emphasis on approaches tested on the Portuguese CoNLL 2006 data set.

2.1
Dependency Grammars

Dependency parsing is a type of syntactic analysis for sentences in natural language, for instance sentences in Portuguese. It is based on syntactic theories that model the dependency relations between the linguistic units of a sentence. Such syntactic theories refer to a specific way to describe the syntactic structure of a sentence.

The starting point of the modern theoretical support of DP is the work of (68). The grammatical theories and formalisms that are included in this tradition, share the same basic assumptions. They believe that the syntactic structure of a sentence, is defined as *lexical elements* linked by binary assymetrical relations called *depedencies*. In this case, a dependency relation holds between a syntactically subordinate linguistic unit, called the *dependent*, and another linguistic unit on which it depends, called the *head*. An example for the Portuguese sentence *É por isso que, diz, não tem pena de Bill.*, is given in Figure 2.1. The dependency relations are represented by arrows pointing from the head to the dependent. In Figure 2.1 is shown a dependency relation

between *tem* and *pena*. In this case, *pena* is the dependent linguistic unit and *temp* is its head.

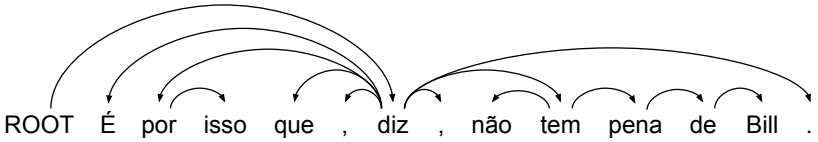

Figure 2.1: Dependency structure for a Portuguese sentence.

Most notable formalisms include Word Grammar (27), Meaning-T ext Theory (46), Functional Generative Description (62), and Constraint Dependency Grammar (37). Although they share basic assumptions, there exists important differences between these formalisms. One of those differences is regarding the analysis of certain syntactic constructions like *coordinations*. Also, they have different points of view regarding the formal properties of the syntactic representation. An extensive review on these theoretical subjects is given in (52). Next we describe the properties and constraints of the syntactic structure.

2.2 Dependency Trees

Usually, the dependency-based syntactic structure of a given sentence is modeled as a rooted graph. The nodes represent the lexical elements of the sentence and the edges represent the dependency relations between such elements. The dependency relations are directed by nature, from heads to dependents. Here we use the term token to refer the lexical elements of the sentence.

Let $x = (x_0, x_1, \ldots, x_n)$ be an input sentence, where x_i, $i \in [1, n]$, is the i-th token in x and x_0 is an artificial token that represents the root. The syntactic structure associated with x is a directed graph $G = (V, A)$ where there is a node i for each token x_i, then $V \subseteq \{0, 1, \ldots, n\}$; and several edges $(i, j) \in A \subseteq V \times V$ that represent the dependency relations. When $(i, j) \in A$ this means that there exists a dependency relation between the head token x_i and the dependent token x_j. The Figure 2.1 exemplifies this definition.

In particular, an edge (i, j) can be labeled with the syntactic dependency relation between the token x_i and x_j. When the structure has no concern with such labels is called *unlabeled dependency structure*, and in practice is commonly adopted by most parsing systems. Next we outline some constraints over this general structure.

Constraints over the dependency structure

In order to provide a complete syntactic analysis of a sentence, the graph must be *connected*. This constraint specifies that every node is related to at least an other node (46). Given this general characterization, we may then impose additional conditions on these graphs. Most versions of dependency grammars assume that each node has at most one head, this is called the *single-head constraint*. Another assumption, is that the graph must not contain cycles, this is the *acyclicity constraint*.

These two constraints, together with connectedness, imply that the graph should be a rooted tree, with a single root node that is not a dependent of any other node. Hence, this constrained structure is called *dependency tree*.

Another concern regarding the formal representation of dependencies is the relation between dependency structure and tokens order. The most well-known example is the constraint of *projectivity*. A dependency graph satisfies the constraint of projectivity with respect to a particular linear order of the nodes if, for every edge (h, d) and node w, w occurs between h and d in a linear order only if w is dominated by h (where dominates is the reflexive and transitive closure of the arc relation). The structure represented in Figure 2.1 represents a projective dependency tree. It is also often assumed that the constraint of projectivity is too rigid for the description of languages with free word order.

Having introduced the basic notions of dependency grammar, we can say that dependency parsing is a syntactic parsing of natural language based on syntactic dependency representations. Taking this into account, if we have a dependency parser, then we are capable of building a dependency tree given an input sentence (32). Next, we review the most paradigmatic approaches to dependency parsing.

2.3 Parsing with Dependency Trees

In computational linguistics, the use of dependency structures has become popular. That is because many dependency parsers combine a competitive parsing accuracy with highly efficient processing. This type of parsing has been successfully employed for many applications such as *information extraction* (14; 65; 9), *question answering* (63; 5) and *machine translation* (58; 71).

In this work we focus on a common parsing paradigm called *data-driven dependency parsing*. Unlike *grammar-based parsing* (52), data-driven approaches learn to produce dependency graphs for sentences from an anno-

tated corpus. The advantage of such models is that they are easily ported to any domain or language in which annotated resources exist.

In accordance with (42), the rise of data-driven methods in natural language processing coupled with the availability of dependency annotated corpora for multiple languages has led to an increased activity in this research area. Clear examples, are the competitions held by the CoNLL Shared Task on the years 2006, 2007, 2008 and 2009.

2.3.1 Conference on Natural Language Learning Shared Task

The Conference on Computational Natural Language Learning (CoNLL) features a shared task, in which the participant's learning systems are trained and tested on the same data sets. In 2006, 2007, 2008 and 2009, DP related tasks have been part of this competition.

In 2006 the shared task was multilingual dependency parsing, where participants had to train a single parser on data from thirteen different languages. This enabled a comparison not only of parsing and learning methods, but also of the performance that can be achieved for different languages (8). In 2007 (51), as in 2006, the shared task has been devoted to dependency parsing, that year they hosted a multilingual track and a domain adaptation track. For the other two years, a joint task on syntactic and semantic dependencies has been proposed. These competitions have created an standard for: task formalization, data sets extracted from available treebanks and evaluation metrics. In other words, they have served to set up a common ground input for the parsers and their further comparison.

There were 13 annotated *corpora* in the CoNLL 2006 Shared Task, one for each proposed language. However, only four of those corpora are still publicly available, namely, the Dutch, Danish, Portuguese and Swedish. Here we are interested in the *corpus* of Portuguese which is the Bosque part of the *Floresta Sintác(c)tica* (1). This corpora consists of European and Brazilian Portuguese language texts. Such information was automatically annotated by the parser Palavras (4) and manually corrected in a postprocessing step (22).

As the two parsers that provided the best results in these Shared Tasks were MSTParser (40) and MaltParser (55), these competitions point out that there are two dominant approaches approaches to data-driven dependency parsing: graph-based and transition-based techniques (45). Following, we review the main approaches in the mentioned paradigms of DP.

2.3.2 Transition-Based Approaches

The transition-based parsers parameterize the parsing problem by the structure of a transition system or abstract state machine. The goal is to map sentences to dependency trees and learn models for scoring individual transitions from one state to the other. In the words of (6), such parsers learn which actions to perform for building a dependency graph while scanning a sentence. The parser builds the dependency trees by going left-to-right (or right-to-left) through the words of a sentence. At each step, a classifier selects the appropriate parsing action for the current state based on a set of features.

MaltParser (55) is the classical structured learning system of the transition-based approaches. In accordance with (45), it uses a transition-based inference algorithm that greedily chooses the best parsing decision based on a trained classifier and current parser history. It also trains a model to make a single classification decision that is to choose the next transition. Regarding to feature representation, it can introduce a rich feature history based on previous parser decisions.

In general, transition-based parsers typically have a linear or quadratic complexity (54; 3). Traditionally, they have relied on local optimization and greedy deterministic parsing (72; 55; 3). The transition-based and non-projective parsing algorithm of (53) has quadratic complexity in the worst case and an expected linear parsing time. Also, globally trained models and non-greedy parsing methods, such as beam search, are increasingly used (29; 69; 75; 77). A combination of a transition-based parsing algorithm that uses a beam search with a latent variable machine learning technique is presented in (69). Next, we review another paradigmatic type of parser, the graph-based parsers.

2.3.3 Graph-Based Approaches

As DP involves the prediction of a dependency tree given a sentence, the graph-based parser defines its model over dependency trees. It learns models for scoring entire parse trees for a given sentence. In particular, it starts with a completely connected graph whose edges are weighted according to a statistical model. Then, it tries to find the spanning tree that covers all nodes in the graph and, at the same time, maximizes the sum of the weights of the edges belonging to such spanning tree.

MSTParser is the classical system of this type of parser. It recasts the problem of parsing as a structured linear prediction problem. When compared

with MaltParser, it uses near exhaustive search over a dense graphical representation of the sentence, with the goal of finding the dependency graph that maximizes the score. It also trains a model to maximize the global score of correct graphs. MSTParser is forced to restrict the score of features to a single or pair of nearby parsing decisions in order to make exhaustive inference tractable.

The original non-projective formulation by (40) had a complexity of $O(n^2)$ but it was not capable of taking into account the second-order features. In this case the choice for an edge is made depending on already chosen edges. Second-order MST parsing was shown to significantly improve results compared to first-order parsing (43; 10) but at the cost of a higher complexity (44).

Regarding the learning of labeled dependency structures, (10) integrated edge labels into the parsing procedure by adding an additional loop over the set of edge labels (L), thus, performance is improved. Also, the theoretical complexity is raised to $O(n^4L)$.

An efficient third-order dependency parsing algorithm is introduced by (31). The algorithm considers substructures containing three dependencies, and is called efficient because it has $O(n^4)$ complexity. The parsing algorithm can utilize features with sibling and grandchild information. This system holds the best result for the Portuguese corpus of the CoNLL 2006 Shared Task.

In the context of CoNLL 2006 competition, (8) observed that it is striking that recent work on data-driven dependency parsing has been dominated by *global, exhaustive, graph-based models*, on one hand, and *local, greedy, transition-based models*, on the other. Next we review the work made towards feature generation and selection in the context of graph-based parsers.

Feature Generation

The prediction of a dependency tree is highly nonlinear on the available input features. In this context, to provide the required nonlinear feature combinations, it is necessary to use some feature generation method.

For graph-based approaches is common the use of features with nonlinear pattern that were created manually by domain experts (40; 31; 10). Another common method is the use of kernel functions on the training algorithm (6; 67; 33). Also, an approach that embeds multiple kernel functions into training algorithms has been examined by (36).

A well explored feature generation method by the LEARN lab is the induction of features based on the entropy concept (17; 19; 60; 61; 47). *Entropy Guided Feature Generation* (EFG) method is a form of feature induction based

on the conditional entropy of basic attributes that are extracted from the paths of a learned decision tree, to build logical conjunctions of such basic attributes. Hence, it promotes the induction of new features with nonlinear pattern. As a result of the work of (17), EFG is successfully integrated with structured perceptron algorithm, leading to new structured framework called *Entropy Structured Learning* (ESL).

Feature Selection

Linear models are pervasive in dependency parsing approaches, mainly because they are efficient training algorithms with proven error bounds. Practical studies on dependency parsing did not pay much attention to its regularization. Efforts towards this direction have been made by (74). In particular, they study three simple but effective task-independent regularization methods. One is to average the weights of different trained models, with the main goal of reducing the bias caused by the specific order of the training examples. Also they consider a penalty term in the loss function to promote regularization. In the last method, they promote the random corruption of the data flow during training, which is called *dropout* in the neural network.

On the other hand, (35; 64) investigate regularizers that promote structured sparsity. The goal is to improve the models that include L_1-norm penalty term in the scoring function, which is known helps control weight complexity.

On the line of research of LEARN lab, the work of (48) uses a sparse linear model through an adaptation for the perceptron algorithm proposed by (23).

Is well known that sparse linear models have the ability of jointly perform the estimation of a linear model and the selection of the subset of the most informative features. In this case, feature selection is attained by encouraging several weights of the model to be zero. This regularization technique is applied by (48) in combination with feature induction, yielding a regularized and nonlinear model with high prediction power.

2.3.4 Portuguese Dependency Parsing

In the past decade, dependency parsing has attracted much attention. A fast progress has been made on pushing the performance of dependency parsers. We are interested in review the systems with best performance on the Portuguese CoNLL-2006 dataset, in order to establish a comparison with the proposed solution. Particularly, we pay attention to graph-based approaches.

In 2005, MSTParser (40) achieved state-of-the-art performance on This approach uses a feature decomposition and a scoring function defined over the edges of the candidate graph. It also recasts the problem of parsing as the problem of finding the maximum spanning arborescense.

In the CoNLL-2006 Shared Task (8), (41) achieved the best performance by applying an extension of MSTParser that uses *second-order features*. It uses a scoring function defined over a more complex decomposition of the candidate graph.

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. An approximation algorithm to this problem is proposed in (43). On the other hand, it also showed that the second-order model outperforms the first-order version, even when approximate prediction is used.

On the other hand, the cube-pruning technique can efficiently introduce higher order dependencies in graph-based parsing. Based on that, (39) seeks to extend (73) by forcing inference to maintain both label and structural ambiguity. In this case they handle higher order substructures to score candidates dependency trees.

As far as we know, the best performing system on the Portuguese CoNLL-2006 dataset is the dual decomposition system proposed by (31). This system introduces a new algorithm to perform approximate prediction with second- and third-order features. 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 the complex features are generated with the manual templates proposed by (43). More recently and also based on dual decomposition algorithm, (34) attained the second-best result for this task when using Portuguese data. Hence, dual decomposition has shown to leads to a certificate of optimality for the vast majority of the sentences.

Also, (76) gets results compared with the state-of-art by using a randomized greedy method of inference based on an approach that starts from a tree drawn from the uniform distribution and uses hill climbing for parameter updates. They test their system across several languages included Portuguese.

In the line of research of the LEARN lab, parsing has been an intensive line of work. Comprised in this research on DP, we found the token-based approaches of (48) and (13). Also (17) recasts DP as a structured linear prediction problem with first-order feature decomposition.

In particular, the token-based approach of (48) uses a supervised machine learning approach that incrementally inducts and selects feature conjunctions derived from base features. This approach integrates decision trees, support vector machines and feature selection using sparse perceptrons in a machine learning framework named IFIS – Incremental Feature Induction and Selection.

On the other hand the work of (48) uses EFG to model the feature generation component. EFG promotes the induction of new features with nonlinear pattern and it was successfully integrated with structured perceptron algorithm leading to a new structured framework called *Entropy-Guided Structure Learning ESL* (17). Also, the work of (48) is based in another token-based approach to dependency parsing named *Entropy Guided Transformation Learning* (ETL) (13).

2.4 Chapter Conclusions

Several techniques have been applied to solve the problem of syntactic analysis of an input sentence. Here we reviewed the main topics referring to dependency-based syntactic analysis, that is the theoretical foundations and the representation used to describe the structure underlying such analysis.

Also, we paid attention to the work on data-driven approaches to dependency parsing and the efforts made by the community of NLP on the CoNLL Shared Tasks to get an standard for evaluating and comparing parsers. Particularly, we focused on the work on graph-based parsers and it main characteristics when compared against their counterpart the transition-based parsers.

3
Dependency Parsing

Dependency parsing refers to the machine learning task of finding the dependency structure of a given sentence. The dependency structure is a directed rooted tree in which the nodes represent the lexical units of the sentence and the arcs represent the dependency relations between such lexical items.

This chapter is devoted to outline the modeling details of our approach to dependency parsing. Starting from the perspective of the canonical graph-based model, we describe a candidate edges filtering process which helps to build sparse candidate dependency graphs by eliminating unpromising arcs. The goal is to accelerate the adjustment of the weights of the linear model that we are proposing in this work. This process is based on auxiliary predictors described in the token classification approach of (48) and (13). We detail the feature engineering process. We divide the attributes by its context, in other words the attributes relative to a token and relative to a relation between tokens.

First, we describe the auxiliary predictors used as input to the edges filtering method. After, we outline the candidate edges filtering process and we make emphasis on two measures: density and recall of *standard golden edges*[1]. These are measures to describe the obtained structures. Finally, we depict the basic features and how we create derived features to improve the accuracy of the proposed learning method.

3.1
Auxiliary Predictors

In (48) and (13), dependency parsing is treated as a token classification machine learning task, where the goal is to predict the head of every token in the sentence using a set of classes resulting from a particular tagging style named relative tags.

They build their class set using three pieces of information about the head token of a given dependent. The first is where the head appears: if before or after of the dependent. The second is relative to the Part-Of-Speech (POS) tag or *postag* of the head. Finally, the distance counter refers to the number

[1] *Standard golden edge* refers to an edge that belongs to the structure that is taken as ground of truth.

of tokens between the head and the dependent that have its *postag* equal to the head's *postag*. Hence, a tag class is composed by the combination of the distance counter, the *postag* and the side of the head token. For example, the tag class "1_v_L" is understood as a head that corresponds with the first verb to the left of the dependent token. A common characteristic of these subtasks is that they consider an extra class name to denote the token that is the root of the dependency tree of a given sentence. So, they do not use the artificial node called ROOT of the canonical graph-based approach.

They use this tagging style to codify the column of the head token in the Portuguese data set of the CoNLL 2006 Shared Task (8; 1; 22). The Table 3.1 shows an example of the relative tag codification for the Portuguese sentence *"É por isso que, diz, não tem pena de Bill."*

Id	Word	***postag***	Head	Special Tagset
1	É	adv	9	2_v_R
2	por	prp	9	2_v_R
3	isso	pron	2	1_prp_L
4	que	adv	9	2_v_R
5	,	punc	6	1_v_R
6	diz	v	0	ROOT
7	,	punc	6	1_v_L
8	não	adv	9	1_v_R
9	tem	v	6	1_v_L
10	pena	n	9	1_v_L
11	de	prp	10	1_n_L
12	Bill	prop	11	1_prp_L
13	.	punc	6	2_v_L

Table 3.1: Relative tag codification for a Portuguese sentence in CoNLL format.

This particular set of classes allows them to decompose the main task of parsing into less complex multiclass subtasks that can be solved independently. The first subtask is to identify if the head appears before or after the dependent, relative to its position in the sentence. The second one is to identify the *postag* of the head token and the last one is to find the correct distance between the head and the dependent. In the particular case of (48), they use the head's *postag* prediction as well as the head's side prediction as input of the task of finding the right distance between the head and the dependent.

The idea here is to use the outputs of the auxiliary predictors to benefit the training of the perceptron algorithm. Based on the information of side and postag of the head given a dependent token, we propose as a preprocessing step, the removal of unpromising candidate edges. Following (48), we detail the input subtasks and report their respective accuracy.

3.1.1 Head Side Predictor

The task of identifying the side of the head involves the prediction of the side of the head relative to the position of the dependent in the sentence. Hence, the side can take two possible values: right (R) or left (L). We also add a special tag (ROOT) to denote the token that is the root of the dependency tree. This is a multiclass prediction problem. The Table 3.2 shows an example of the side tags codification for the Portuguese sentence *"É por isso que, diz, não tem pena de Bill."*

Id	Word	*postag*	Head	Special Tagset
1	É	adv	9	R
2	por	prp	9	R
3	isso	pron	2	L
4	que	adv	9	R
5	,	punc	6	R
6	diz	v	0	ROOT
7	,	punc	6	L
8	não	adv	9	R
9	tem	v	6	L
10	pena	n	9	L
11	de	prp	10	L
12	Bill	prop	11	L
13	.	punc	6	L

Table 3.2: Head side codification for a Portuguese sentence in CoNLL format.

The predictor trained by (48) used Incremental Feature Induction and Selection (IFIS) algorithm. Specifically, this predictor obtained 98.09% of accuracy on the Portuguese dataset of the CoNLL 2006 Shared Task.

3.1.2 Head Part-Of-Speech Predictor

The task of identifying the *postag* of the head involves the prediction of the *postag* of the head token of given dependent token. In this particular case, the postag takes values from the coarse-grained *postag* set of the Portuguese data set of the CoNLL-2006 Shared Task. Such tag set is detailed in Appendix A. We also add a special tag (ROOT) to denote the token that is the root of the dependency tree. This is a multiclass prediction problem. Revisiting our main example, the Table 3.3 shows an example of the *postags* codification for the Portuguese sentence *"É por isso que, diz, não tem pena de Bill."*

Id	Word	*postag*	Head	Special Tagset
1	É	adv	9	v
2	por	prp	9	v
3	isso	pron	2	prp
4	que	adv	9	v
5	,	punc	6	v
6	diz	v	0	ROOT
7	,	punc	6	v
8	não	adv	9	v
9	tem	v	6	v
10	pena	n	9	v
11	de	prp	10	n
12	Bill	prop	11	prp
13	.	punc	6	v

Table 3.3: Head POS codification for a Portuguese sentence in CoNLL format.

The Portuguese dataset of the CoNLL 2006 Shared Task and IFIS (48), were used in the training of this predictor. It attained 95.67% of accuracy on the test dataset.

Next, we describe the method for filtering candidates edges, that is the process of removing unpromising edges from the candidate dependency graph by making use of head's side and *postag* information.

3.2 Candidate Edge Filter

The canonical graph-based parser defines its model over dependency trees. It learns models to score entire parse trees for a given sentence. In particular, it starts with a completely connected graph whose edges are weighted according to a statistical model. Then, it tries to find the spanning tree that covers all nodes in the graph and, at the same time, maximizes the sum of the weights of the edges belonging to the spanning tree (40; 42). We seek to improve this representation by filtering a large amount of unpromising edges. Our goal is to accelerate the adjustment of the weights of our model.

Let $x = (x_0, x_1, \ldots, x_n)$ be an input sentence, where x_i is the i-th token in x for $i \in [1, n]$. In order to represent the syntactic structure associated with x, let's consider $G_x = (V_x, A_x)$ a completely connected graph, where there is a node i for each token x_i, then $V_x \subseteq \{0, 1, \ldots, n\}$ and several edges $(i, j) \in A_x \subseteq V_x \times V_x$ that represent the dependency relations. Also, let $D(G_x)$ be the set of subgraphs of G_x that are valid dependency trees for the sentence x. Because G_x contains all the possible arcs between tokens, the set $D(G_x)$ must necessary contain all dependency trees for x. An example graph G_x and the

corresponding dependency tree for the sentence in Portuguese *Um revivalismo refrescante*, is shown in Figure 3.1.

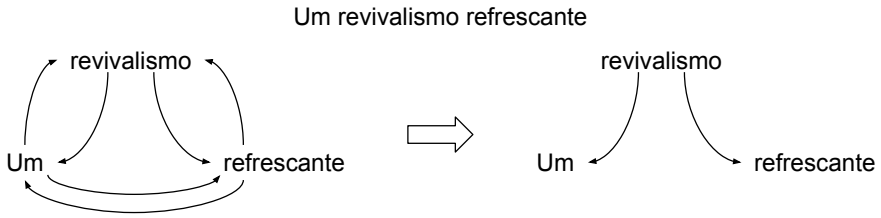

Figure 3.1: On the left is shown an example of the candidate graph G_x for the sentence *Um revivalismo refrescante*, to the right is shown the corresponding dependency tree.

To fulfill our goals, we seek to reduce as much as possible the number of irrelevant arcs, in other words, arcs that belong to A_x but does not belong to any one of the possible dependency trees for x. This reduction can be achieved by using a filter for the candidate arcs of G_x.

In order to reduce the density of the candidate graph G_x, we propose to remove a set of unpromising arcs from A_x based on information about the head's side and *postag*. Let's consider the set of unpromising arcs $\overline{A}_x$ as the set of arcs that does not belong to any subgraph in $D(G_x)$. Also, we consider $\overline{A}_d$ as the set of unpromising arcs for a given dependent token d, that is the set of unpromising arcs incoming the token d. Given a dependent token d, let *hpos* be the output of the auxiliary predictor for head's *postag* and *hside* the output of the auxiliary predictor for head's side. Making use of this information, we can say that the set of incoming arcs on d with *postag* different from *hpos* and with side different from *hside* are in $\overline{A}_d$, therefore can be considered as irrelevant for the learning task. For example, conjoining the information from head *postag* and side in Table 3.4 for the token *tem* we can eliminate all the incoming arcs on *tem* that are not verbs on the left. This means that the filter leaves out all edges entering *tem* except for (diz, tem).

A particular case occurs when the predictions of head's postag and side, for a given token d, coincide with the especial tag ROOT. In such case d is the root of the dependency tree, for that reason we can eliminate all incoming arcs on it. Returning to our example, we can eliminate all incoming arcs on *diz*.

Applying this process to all tokens on the sentence we obtained a candidate graph G'_x with $r \in V'_x$ denoted as root that have only a few arcs when compared against G_x. Particularly, Figure 3.2 shows the filtered candidate graph on the left and the corresponding dependency tree on the right for the sentence codified in Table 3.4. We strongly believe that the perceptron algorithm can be benefited if the learning process is carried out with these

Id	Word	*postag*	Head POS	Head Side
1	É	adv	v	R
2	por	prp	v	R
3	isso	pron	prp	L
4	que	adv	v	R
5	,	punc	v	R
6	diz	v	ROOT	ROOT
7	,	punc	v	L
8	não	adv	v	R
9	tem	v	v	L
10	pena	n	v	L
11	de	prp	n	L
12	Bill	prop	prp	L
13	.	punc	v	L

Table 3.4: Combining information of Head POS and Side for a Portuguese sentence in CoNLL format

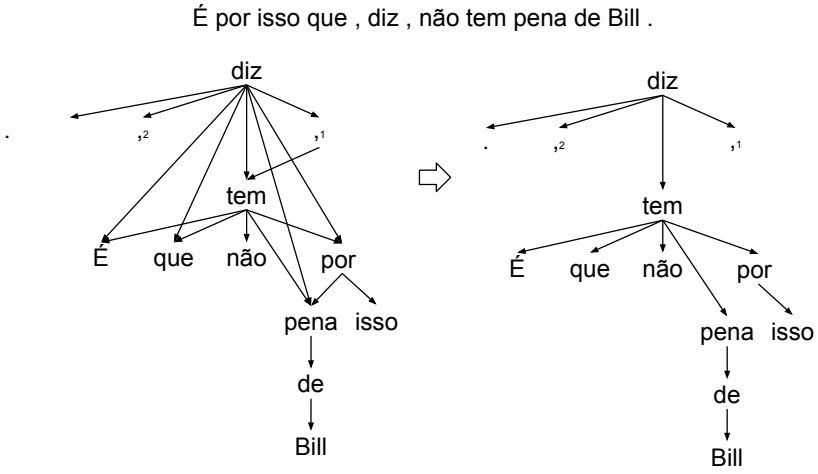

Figure 3.2: On the left is shown an example of the filtered candidate graph G'_x for the sentence *É por isso que, diz, não tem pena de Bill.*, to the right is shown the corresponding dependency tree.

filtered candidate graphs and we show evidences in the empirical evaluation of the proposed solution.

Following the idea of (48) and (13) of dividing parsing into less complex subtasks, we divide parsing into two subtasks. For a given dependent token, we used two auxiliary predictors that predict the information of head's postag and side. Such information is used as input for the parsing task. Our machine learning task is defined in terms of using the predicted information about of the head of a given dependent token, to identify its position on the sentence.

3.2.1
Candidate Graph Measures

The goal is to filter a large amount of irrelevant arcs to accelerate the adjustment of the weights of our learning algorithm. Sparseness and high percentage of *standard golden edges* are desirable characteristics in our candidate graphs. The last characteristic is important because we are using information from auxiliary predictors that is not 100% accurate.

In order to characterize the filtered candidate graph G'_x, we calculate its density which is defined as follows

$$Density(G'_x) = \frac{|A'_x|}{|V'_x|(|V'_x| - 1)}.$$

Other important measure is the recall of *standard golden edges*. Given $y \in Y(x)$ a *standard golden dependency tree*[2], the recall is defined as follows

$$Recall(G'_x, y) = \frac{|A_y \cap A'_y|}{|A_y|}.$$

Next, we describe the basic features associated with an edge of a candidate graph.

3.3
Basic Edge Features

In order to perform the parsing task each lexical item in the input sentence x is represented as a list of linguistic properties like the word form, the *postag*, gender, number and other morpho-syntactic features. Following the canonical graph-based approach (40), $\Phi(x, y)$ is a feature representation defined over the sentence x an its parse tree y.

To construct and effective representation, $\Phi(x, y)$ is decomposed into local representations relative to less complex parts p of the dependency tree y,

$$\Phi(x, y) = \sum_{p \in y} \phi(x, p).$$

The most simple decomposition of the dependency structure is over edges. Standard decompositions include other types of parts like sibling arcs or grandparent arcs which are known as higher-order decompositions.

We are interested in adopt the arc-based model, in that way an edge (i, j) connecting the lexical items x_i and x_j is represented by a vector

$$\Phi(x, i, j) = (\phi_i(x, i, j), \ldots, \phi_M(x, i, j))$$

[2] *Standard golden dependency tree* refers to the structure that is taken as ground of truth.

of M binary features. Hence the joint feature vector is

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

which gives the frequency distribution of the local features ϕ in the dependency tree y.

The basic features associated to an edge (i, j) are defined by its context. In particular, we consider features relative to tokens and features relative to the relation between tokens. We concatenate the features relative to the word context with the features of the relation between i and j, Figure 3.3 shows this.

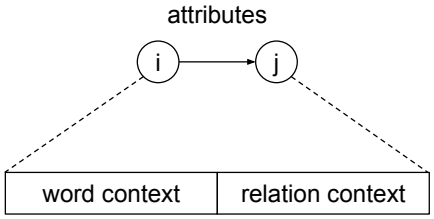

Figure 3.3: Edges feature representation.

Here we use the same set of features of (48). Next we detail the set of features that we use given the mentioned contexts.

3.3.1 Token Context

The set of basic features associated with a token are the features codified in the CoNLL format for dependency parsing task. Such features are word-form, lemma, coarse-grained *postag*, fine-grained *postag* and other syntactic and morphological features. In our case, given this basic set of features we also add derived features that we describe next and summarize in Table 3.5.

The features word-form and lemma have a large number of instances. In order to reduce the cardinality of the feature lemma we substitute it for the *lemma of the verb on the left* of every token.

Instead of using the two types of *postags* associated with a lexical item, we only use the coarse-grained *postag* that have a more compact tag set. Further, we add the head's *postag* and side features that are the output of the respective auxiliary predictors. In addition, we work with several counter features like: the *number of nouns and verbs that occurs to the left* of the dependent token, as well as the *number of nouns and verbs appearing on the right.*

Another important information that we use is the *chunk tag* information and the *clause tag.* Particularly, chunking refers to the natural language

Feature	Description
token	Word-form
POS	*postag* of the token
fts	Other morphological and syntactical attributes
Num. of n left	Number of nouns to the left
Num. of n right	Number of nouns to the right
Num. of v left	Number of verbs to the left
Num. of v right	Number of verbs to the right
Left v lemma	Lemma of the verb on the left
Chunk	*Chunk* tag for the token
Claused start	*Clause start* tag for the token
Clause end	*Clause end* tag for the token
Clause	*Clause* tag for the token
hPOS	Head's *postag*
hSide	Head's side

Table 3.5: Token context attributes.

processing task of breaking the sentence into nonoverlapping parts or chunks, ensuring that lexical items that are syntactically related belong to the same chunk. It have been proved that this information offers insights for the parsing process and helps to improve accuracy (17). The chunk attributes are obtained by the system of (21) and the used tag set is described in Appendix B.

The other important information is about sentence clause segmentation. This is a natural language processing task that identifies the clauses of a given sentence. This information is obtained using the system of (16). In Appendix B we describe the tag set for this task.

This give us a total of 14 attributes per token. Next, we present a mechanism to improve this representation by concatenating information about the surrounding tokens given a current one.

Window Format

In order to improve the information about the context of a lexical item, we represent it using a window context. This is the concatenation of the information of k tokens on the left, followed by the information of the current lexical item and followed by the information relative to the next k tokens. Here, we use $k = 3$, hence the window context has length 7. Also, in Figure 3.4 is shown how we represent the features of the word context of an edge (i, j), this is the concatenation of the window context of i with the window context of j.

In order to improve the representation of the current token we add 12 attributes that are summarized in Table 3.6.

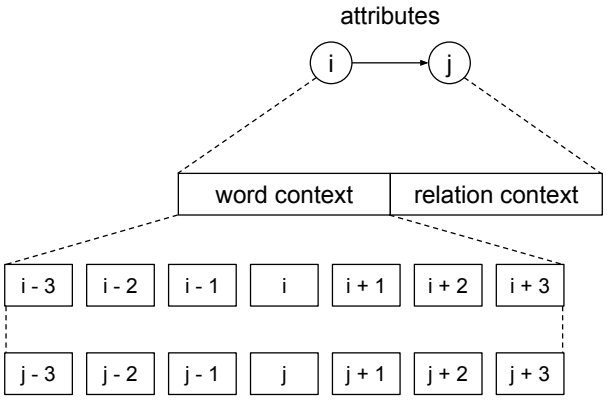

Figure 3.4: Feature representation of the word context.

Feature	**Description**
Num. of Start clause left	Number of tokens with start clause tag to the left
Num. of Start clause right	Number of tokens with start clause tag to the right
Num. of End clause left	Number of tokens with end clause tag to the left
Num. of End clause right	Number of tokens with end clause tag to the right
Num. of NP chunks left	Number of noun chunks to the left
Num. of NP chunks right	Number of noun chunks to the right
Num. of VP chunks left	Number of verbal chunks to the left
Num. of VP chunks right	Number of verbal chunks to the right
Num. of PP chunks left	Number of prepositional chunks to the left
Num. of PP chunks right	Number of prepositional chunks to the right
Num. of tokens left	Number of tokens to the left
Num. of tokens right	Number of tokens to the right

Table 3.6: Additional token context attributes for window format.

We used a window context of size 7, then we have 110 features describing a token. Therefore, we have 220 features as the result of concatenating the window contexts of i and j for an edge (i, j).

Feature	Description
Num. of Start clause tags	Number of start clauses between i and j
Num. of End clause tags	Number of end clauses between i and j
Num. Ck Noun	Number of start noun chunks between i and j
Num. Ck Verb	Number of start verbal chunks between i and j
Num. Ck Prep	Number of start preprositional chunks between i and j
Num. POS Art	Number of articles between i and j
Num. POS Conj	Number of conjunctions between i and j
Num. POS Noun	Number of nouns between i and j
Num. POS Nprop	Number of proper nouns between i and j
Num. POS Prep	Number of prepositions between i and j
Num. POS Verb	Number of verbs between i and j
Num. Tok Pai	Number of tokens between i and j

Table 3.7: Attributes of the relation context between the token i and its candidate head j.

3.3.2 Relation Context

We describe the features relative to the edge (i, j), these are basically counters for properties of the tokens between i and j. A summary of this set of features is presented in Table 3.7.

3.4 Chapter Conclusions

As shown in (48) and (13), their token-based approach to dependency parsing divides the task into three less complex subtasks, those are multiclass predictors for each of the heads properties: side, *postag* and distance relative to the dependent token. We shown how to decompose parsing into subtasks and use the side and *postag* prediction as input to the parsing task but in the context of structured prediction and not a multiclass approach as stated in the works that we are following. Taking this into account, we define the machine learning task in question and detailed the set of features describing

dependency relations between the lexical items of the input sentence. The next chapters describe the learning methods used in this work to solve dependency parsing based on the modeling details described here.

4
Structure Perceptron

The canonical graph-based approach builds a model in which the parameters are over dependency subgraphs and learns such parameters to globally score correct dependency trees above incorrect ones. Based on this, the parsing task can be modeled as a structured linear prediction problem. Then, the intention is to define the scoring function that helps discriminate between correct structures and incorrect ones. To fulfill this goal we need to define a scoring function s over the dependency trees that can be inferred from the proposed filtered candidate graph G'_x. Such candidate graph contains the syntactic structure underlying the input sentence $x = (x_1, \ldots, x_n)$.

A step towards a successful modeling of DP as a structured linear prediction problem is to decompose the joint feature representation vector $\Phi(x, y)$ over less complex parts of the dependency tree. This has the main goal of expressing the score of the candidate structure as a linear combination of the input features. The most simple decomposition of Φ is over edges. Following this line of thinking, the scoring function is factorized in terms of edges too which is known as the arc-based model.

Finally, we present the structured learning algorithm that learns the parameters of the mentioned model. Here we are interested in using structure perceptron as the classification algorithm. Also we pay special attention to regularization techniques, which are well known for helping on the improvement of the performance of the obtained models. In addition, we are interested in the promotion of sparsity on the obtained models because it is known that sparse linear models perform simultaneously feature selection and the estimation of the parameters. The result is a more compact and light-weight model which helps with the control of a potential overfitting. Then it can be employed as a powerful regularization technique.

The remainder of this chapter is organized as follows, first we give details of how the learning is performed with dependency trees. Then, we specify the edge-based scoring function that serves to discriminate candidates to dependency tree. Finally, we describe the structure perceptron and the corresponding extensions that need to be included in order to fulfill the proposed goals.

4.1 Dependency Tree Learning

DP refers to the task of finding the syntactic dependency structure underlying a given sentence $x = (x_0, \ldots, x_n)$. In order to model this task as a structured linear prediction problem, the classical graph-based approach recasts the parsing problem as a maximum branching problem (11; 15). The idea is to use a learning algorithm to learn the parameters of a scoring function s over less complex parts of a dependency tree. When such parts coincide with the candidate head-dependent edges (i, j), the scoring function $s(x, i, j)$ is called edge-based model.

In this case, the *prediction problem* $F(x)$ is to predict a dependency tree for an input sentence x, which coincides with finding the maximum-score tree among the valid rooted trees

$$F(x; w) = arg\, max_{y \in Y(x)} s(x, y; w),$$

where $s(x, y; w) = \sum_{(i,j) \in y} s(x, i, j; w)$ is the score of a candidate tree $y \in Y(x)$. This is the well studied maximum branching problem that can be efficiently solved by Chu-Liu-Edmonds' algorithm (11; 15). There is also an improved implementation of this algorithm by Tarjan (66).

By processing a given sample of correct sentence-tree pairs, the supervised learning algorithm learns a scoring function that generalizes for unseen sentences. The scoring function takes the form

$$s(x, i, j; w) = \sum_{m=1}^{M} w_m \cdot \phi_m(x, i, j) = \langle w, \Phi(x, i, j) \rangle,$$

where $w = (w_1, \ldots, w_M)$ is the weight vector that defines a dependency parsing model, $\Phi(x, i, j) = (\phi_1(x, i, j), \ldots, \phi_M(x, i, j))$ is the feature vector of M binary features that describe the relation between (i, j) and $\langle \cdot, \cdot \rangle$ is the scalar product operator.

This means that the score of a dependency tree has the form

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

Thus, the learning problem consists in the search of the feature weights that make $F(x)$ accurate on the training data and, moreover, with good generalization performance on unseen data. In this work, we use the averaged structure perceptron algorithm (12) with a large margin extension (18; 38; 70) as the main training algorithm.

The perceptron is an online learning algorithm, hence it is sensitive to the order that training examples are received. To tackle this disadvantage we include an extension to the learning algorithm that shuffles the order of the training examples at the beginning of every epoch.

In our approach we include an extension to structure perceptron for inducing model sparsity (23). We use this technique to regularize the attributes domains and following (48), we introduce the dropout extension in the structure predictor. Also, we present an extension to structure perceptron that allows the inclusion of features with nonlinear pattern, which is performed with *Entropy-Guided Feature Generation* algorithm (17; 19; 60; 47).

4.2 Structure Perceptron

The structure perceptron algorithm (12) is analogous to its univariate counterpart (59). Given a training sample D of correct sentence-tree pairs, the algorithm generates a sequence $w^0 = 0, w^1, \ldots, w^m$ of models.

At each iteration t, the structure perceptron draws a training instance $(x, y) \in D$ and performs two major steps:

(1) a prediction $\widehat{y} = F(x)$ is made using the current model w^t;

(2) $w^{t+1} \leftarrow w^t + \Phi(x, y) - \Phi(x, \widehat{y})$.

This algorithm is presented in Figure 4.1 in the context of dependency parsing.

Input: $D = \{(x, y)\}$ binary data set
Output: w
$w \leftarrow 0$
while no convergence
  for each $(x, y) \in D$
    $\widehat{y} \leftarrow arg\, max_{\bar{y} \in Y(x)} \sum_{(i,j) \in \bar{y}} s(x, i, j; w)$
    $w \leftarrow w + \Phi(x, y) - \Phi(x, \widehat{y})$
return w

Figure 4.1: Structure Perceptron algorithm.

Note that, when the current model makes a correct prediction $\widehat{y} = y$, the model does not change, that is $w_{t+1} \leftarrow w_t$. When the prediction is wrong, the update rule favors the correct output y over the predicted one $\widehat{y}$. Regarding binary feature functions, for instance, the update rule increases the weights of features that are present in y but missing in $\widehat{y}$ and decreases the weights of features that are present in $\widehat{y}$ but not in y. The weights of features that

are present in both y and $\widehat{y}$ are not changed. A simple extension of Novikoff's theorem (56) shows that the structure perceptron is guaranteed to converge to a zero loss solution, if one exists, in a finite number of steps (2; 12).

4.2.1 Averaged Models

The structure perceptron algorithm with an averaging strategy is proposed by (12). This is a known strategy, used even with the binary perceptron, which makes the algorithm more robust. In Figure 4.2, we present the pseudo-code of the averaged structure perceptron.

Input: $D = \{(x, y)\}$ binary data set
Output: w
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
  for each $(x, y) \in D$
    $\widehat{y} \leftarrow arg\, max_{\overline{y} \in Y(x)} \sum_{(i,j) \in \overline{y}} s(x, i, j; w)$
    $w^{t+1} \leftarrow w^t + \Phi(x, y) - \Phi(x, \widehat{y})$
    $t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} w^k$

Figure 4.2: Averaged Structure Perceptron algorithm.

It is very similar to the algorithm presented in Figure 4.1. Given that the algorithm executes for T iterations, the averaged structure perceptron builds a sequence of models $w^0, \ldots, w^T$. Instead of returning the last model w^T, like the ordinary structure perceptron, it returns the average among all models built, that is, $w = \frac{1}{T} \sum_{k=1}^{T} w^k$. Each update in the structure perceptron algorithm has a large potential impact on the model parameters. Thus, the averaged algorithm is more robust to noisy examples and usually performs significantly better than the nonaveraged version.

4.2.2 Shuffled Training Examples

Structure perceptron is an online algorithm that can be affected by the order in which the examples are processed. In order of reduce the bias caused by the specific order of the training examples, we propose to randomly shuffle the training examples at the beginning of every training epoch. In Figure 4.3, we present the pseudo-code with the proposed modification.

Input: $D = \{(x, y)\}$ binary data set
Output: w
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
  for each $(x, y) \in Shuffle(D)$
    $\widehat{y} \leftarrow arg\, max_{\bar{y} \in Y(x)} \sum_{(i,j) \in \bar{y}} s(x, i, j; w)$
    $w^{t+1} \leftarrow w^t + \Phi(x, y) - \Phi(x, \widehat{y})$
    $t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} w^k$

Figure 4.3: Shuffled Structure Perceptron algorithm.

In particular, the function $Suffle(D)$ returns the training examples in the order given by a permutation of $|D|$ numbers. This modification turns the structure perceptron more robust.

4.3 Large Margin Classifiers

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. In this work, we use a large-margin generalization of the structure perceptron that is based on the well known margin rescaling technique for structural support vector machines. For a training instance $(x, y) \in D$, instead of $F(x)$, we use the following *loss-augmented* prediction problem in step 1 of the structure perceptron learning algorithm

$$F_\ell(x) = arg\, max_{\bar{y} \in Y(x)} s(x, \bar{y}) + \ell(y, \bar{y}),$$

where $\ell(\cdot, \cdot) \geq 0$ is a given loss function that measures the difference between a candidate tree and the correct one.

In other words,

$$F_\ell(x, y; w) = arg\, max_{\bar{y} \in Y(x)} \left[\left(\sum_{(i,j) \in \bar{y}} s(x, i, j; w) \right) + C \cdot \ell(y, \bar{y}) \right],$$

where C is the constant that balance the weight between the loss function and the learned edges weights w.

We use the most common loss function for dependency trees, which just counts, for all tokens within the input sentence, how many head tokens have

been *incorrectly* assigned in the predicted tree, which is given by

$$\ell(y, \bar{y}) = \sum_{(i,j)\in\bar{y}} 1[(i,j) \notin y].$$

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

$$F_\ell(x, y; w) = arg\, max_{\bar{y}\in Y(x)} \sum_{(i,j)\in\bar{y}} \Big(s(x, i, j; w) + C \cdot 1[(i,j) \notin y] \Big).$$

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

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

and then we have that

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

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 4.4.

Input: $D = \{(x, y)\}$ binary data set,
C loss constant
Output: w
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
for each $(x, y) \in Shuffle(D)$
$\widehat{y} \leftarrow arg\, max_{\bar{y}\in Y(x)} \sum_{(i,j)\in\bar{y}} \Big(\langle w^t, \Phi(x, i, j)\rangle + C \cdot 1[(i,j) \notin y] \Big)$
$w^{t+1} \leftarrow w^t + \sum_{(i,j)\in y} \Phi(x, i, j) - \sum_{(i,j)\in\widehat{y}} \Phi(x, i, j)$
$t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} w^k$

Figure 4.4: Large margin Structure Perceptron algorithm for dependency parsing.

The unique modification to the structure perceptron algorithm is in the 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 (x, y) implies a model update whenever the current model does not respect the following margin constraint

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

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

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

In order to enhance the feature representation by introducing features with nonlinear pattern we modify the structured perceptron. Next we detail such modifications.

4.4 Feature Induction

Many structure learning problems are highly nonlinear on the available input features, which is the case of dependency parsing. Therefore, when training a linear model for a structured prediction problem, 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.

In this work, we used *Entropy-guided Feature Generation* (EFG) (19; 17) 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) (61), which generalizes Transformation-Based Learning (7) by automatically generating rule templates.

In (17), they proposed Entropy-Guided Structure Learning (ESL) framework, which integrates in a general structure learning framework EFG as a preprocessing step. As an instance of this general structure learning framework we proposed the Structure Sparse Perceptron described above; the pseudo code of the new extension is shown in Figure 4.5.

Input: $D = \{(x, y)\}$ binary data set,
C loss constant
Output: w
$\Phi(D) \leftarrow EFG(D)$
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
for each $(x, y) \in Shuffle(D)$
$\widehat{y} \leftarrow arg\, max_{\bar{y} \in Y(x)} \sum_{(i,j) \in \bar{y}} \Big(\langle w^t, \Phi(x, i, j) \rangle + C \cdot 1[(i, j) \notin y] \Big)$
$w^{t+1} \leftarrow w^t + \sum_{(i,j) \in y} \Phi(x, i, j) - \sum_{(i,j) \in \widehat{y}} \Phi(x, i, j)$
$t \leftarrow t + 1$
return $\frac{1}{t} \sum_{k=1}^{t} w^k$

Figure 4.5: Large margin Structure Perceptron algorithm for dependency parsing with feature induction.

EFG automatically derives a set of basic feature conjunctions, which are denoted *feature templates*. Each generated template is used to instantiate a feature in $\Phi(x, y)$, hence we are increasing the dimensions of the feature space. For the purpose of this document, we refer to this increment as the expansion step of the joint feature representation vector.

By considering this modification in the perceptron algorithm we are obtaining a structured model that is linear on the derived feature representation, which corresponds to a nonlinear combination of the basic features. Hence, we are introducing features with nonlinear pattern into a linear algorithm.

Next, we describe the process of the *feature templates* generation that is built-in the expansion step.

4.4.1 Template Generation

The first step of our automatic template generation method is to train a decision tree on a dataset, where each example comprises the basic features related to one *local decision variable* of the prediction problem. Local decision variables correspond to edges, since the prediction problem is to choose some edges from all candidate edges between tokens. Thus, given a sentence in the input dataset, we generate a decision tree example for each candidate edge. The

binary decision variable indicates whether an edge is included in the correct dependency tree or not. Hence, the decision tree algorithm learns the output tree by predicting whether an edge is correct or not. Figure 4.6 illustrates a decision tree which is used to extract feature templates.

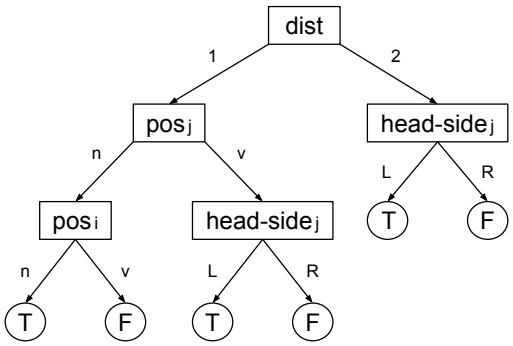

Figure 4.6: Decision tree example.

From the learned decision tree, the proposed method uses a very simple tree decomposition scheme to extract templates. The decomposition process is based on a depth-first traversal of the decision tree and thus can be recursively described as follows. For each visited node, a new template is created by conjoining the node feature with its parent template.

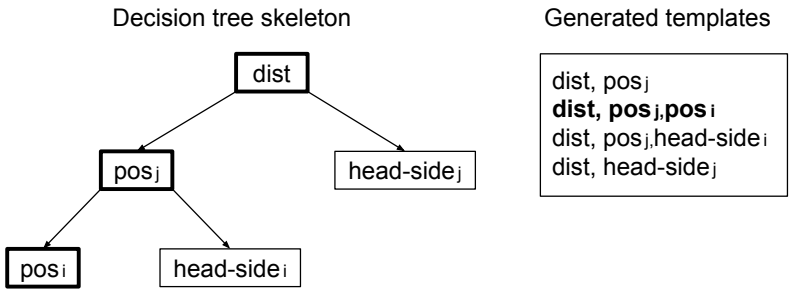

Figure 4.7: Decision tree skeleton example and the corresponding templates. We highlighted the template "dist, pos_j, pos_i" in the decision tree skeleton to illustrate the corresponding path on the tree.

The tree in the left side of the Figure 4.7 uses four basic features (rectangular nodes): `dist` is the absolute distance between the head and the dependent token, `head_side_j` is the side of the head token relative to the token j, `pos_i` and `pos_j` are the part-of-speech respectively for the token i and j of the dependent and the head token, respectively. The round nodes are the decision variable values (`T` for true and `F` for false). The generated templates are listed in the right side of the figure. In other words, we create a template for each path from the root to every other decision tree node, ignoring the feature values at the edges and, thus, using only the node features.

4.5 Sparser Perceptron Models

Sparse linear models have emerged as a powerful framework to deal with various supervised estimation tasks, in machine learning as well as in statistics and signal processing. These models basically seek to predict an output by linearly combining only a small subset of the features describing the data (28).

Here we are concerned with model selection: which features should be used to define the prediction score? We want to take advantage of high-dimensional inputs while selecting values of w that achieve high accuracy not only on training data, but also to generalize well on new data. The fact is that models with few features or sparse models are desirable for several reasons like compactness, interpretability, good generalization (35).

For the particular case of perceptron, to simultaneously address this variable selection and the linear model estimation, Goldberg and Elhadad have proposed a modification on the prediction step, inspired in the number of times that a feature participates in models updates. On they own words:

"*Our sparse variant of the perceptron algorithm is based on the intuition that only relevant features should end up in the final parameter vector, and that all features are irrelevant until proven otherwise. For a feature to prove its relevance, it needs to participate in a minimum number of updates.*"

Input: $D = \{(x, y)\}$ binary data set,
C loss constant,
L update counters threshold
Output: w
$\Phi(D) \leftarrow EFG(D)$
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
  for each $(x, y) \in Shuffle(D)$
    $\widehat{y} \leftarrow arg\, max_{\bar{y} \in Y(x)} \sum_{(i,j) \in \bar{y}} s'_\ell(x, y, i, j; w, u, L)$
    $w^{t+1} \leftarrow w^t + \Phi(x, y) - \Phi(x, \widehat{y})$
    $u \leftarrow u + 1[\Phi(x, y) \neq \Phi(x, \widehat{y})]$
    $t \leftarrow t + 1$
  for all i **s.t** $u_i < L$
    $u_i \leftarrow 0$
return $\frac{1}{t} \sum_{k=1}^{t} w^k$

Figure 4.8: Structure Sparse Perceptron algorithm with consideration of large margin.

Specifically, to include this extension on the prediction step of perceptron

we maintain a vector u with the number of updates of w in which every feature has participated. Also, we use a threshold L to establish when a feature became relevant. In other words, L is the lower bound of number of updates of w required by an attribute in order to participate in the activation rule of the learning algorithm. Here the u vector acts as a mask because the $\langle \cdot, \cdot \rangle$ not consider binary features $\phi_i, i \in [1, M]$ with updates below L.

As the main question is the selection of attributes that participate in a minimum number of updates of w we modify the original arc-based scoring function

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

to activate the mask when ϕ_i has attained L, therefore the modified scoring function is

$$s'(x, i, j; w, u, L) = \langle w[u \geq L], \Phi(x[u \geq L], i, j) \rangle,$$

if we consider a large margin, the new scoring function has the form

$$s'_\ell(x, y, i, j; w, u, L) = \langle w[u \geq L], \Phi(x[u \geq L], i, j) \rangle + C \cdot 1[(i, j) \notin y].$$

In order to consider s'_ℓ as the scoring function the modified prediction function is

$$F'_\ell(x, y, i, j; w, u, L) = arg\, max_{\bar{y} \in Y(x)} \sum_{(i,j) \in \bar{y}} s'_\ell(x, y, i, j; w, u, L),$$

which is a maximum branching problem just as the original prediction problem, but with modified edge weights. In this scenario, we are able to use Chu-Liu-Edmonds as inference method.

We also need to consider in the update step of the structure perceptron algorithm how to update the information on the vector u. Our update rule is simple, augment the counter in one for the binary feature $\phi_i, i \in [1, M]$ if $\phi_i(x, y) \neq \phi_i(x, \widehat{y})$. In Figure 4.8, we present the pseudo-code of the structure perceptron integrating the outlined extensions in previous sections.

When an epoch ends, the algorithm performs an additional step, that is to reset the update counters of the nonselected features. The consideration of this extension also introduces another meta-parameter that must be calibrated in validation step, the threshold constant L.

In NLP context, features are usually binary functions indicating the presence of some condition which results in a very high-dimensional feature space were only few of the features are active for each observed instance is expected that a lot of features are irrelevant for classification. Particularly, (48) observed a great amount of irrelevant features in their token based approach to dependency parsing.

4.5.1 Dropout

We present a modification made to the structure sparse perceptron proposed by (48). Dropout is introduced with the main goal of promoting regularization through eliminating correlated attributes. This allows a better control over the number of selected attributes. This modification is an adaptation for sparse perceptron of the idea of dropout of neural networks, proposed by (26).

Input: $D = \{(x, y)\}$ binary data set,
C loss constant,
L update counters threshold,
$P_{dropout}$ probability of dropout
Output: w
$\Phi(D) \leftarrow EFG(D)$
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
  for each $(x, y) \in Shuffle(D)$
    $\widehat{y} \leftarrow arg\, max_{\overline{y} \in Y(x)} \sum_{(i,j) \in \overline{y}} s'_\ell(x, y, i, j; w, u, L)$
    $w^{t+1} \leftarrow w^t + \Phi(x, y) - \Phi(x, \widehat{y})$
    for all i **s.t** $\Phi(x, y) \neq \Phi(x, \widehat{y})$
      $r \leftarrow random[0, 1)$
      if $r \geq P_{dropout}$
        $u_i \leftarrow u_i + 1$
    $t \leftarrow t + 1$
  for all i **s.t** $u_i < L$
    $u_i \leftarrow 0$
return $\frac{1}{t} \sum_{k=1}^{t} w^k$

Figure 4.9: Structure Sparse Perceptron algorithm with consideration of large margin and dropout.

Specifically, we modify our u update rule to not always increment the counter of $\phi_i(x, y), i \in [1, M]$, if the later participates in an update of the current model w. We consider a probability of dropping out the feature ϕ_i, respect to a given probability $P_{dropout}$. The consideration of this extension also

introduces another meta-parameter that must be calibrated in validation step, the probability of dropout $P_{dropout}$. In Figure 4.9, we present the pseudo-code of the structure perceptron integrating the outlined extensions in previous sections.

With the introduction of dropout extension, we can make a fine grane selection of attributes, through variations of the $P_{dropout}$. In the original algorithm, we do not get such fine grained selection of attributes because the threshold of minimum updates is an integer.

4.6 Chapter Conclusions

Due to its flexibility, robustness and simplicity, the perceptron algorithm is one of the most popular linear discriminant methods used to learn complex representations in the scenario of structure prediction. Here we presented how the learning of dependency trees can be done with structure perceptron and the extensions that perform feature induction and selection. This algorithm can be modified to produce sparse models and combined with an induction method to include features with nonlinear pattern.

5
S-IFIS Structured Learning with Incremental Feature Induction and Selection

Incremental Feature Induction and Selection (IFIS) (48) is a framework to create regularized nonlinear models with high performance using a linear algorithm. This approach integrates decision trees, support vector machines and feature selection via multiclass sparse perceptrons. It was empirically evaluated in dependency parsing with a token-based approach, in other words using a multiclass prediction approach. Also, they propose to use other algorithms to substitute the components of the framework. For example, for structured prediction they propose ESL (16; 17) or Structured Support Vector Machines to substitute SVM as main classifier.

Inspired by this framework and to give continuity to the work on parsing of the LEARN lab, we propose an approach to parsing in a structured context by performing cycles of compression and expansion of the feature representation vector Φ. We describe the incremental scheme of compression and expansion of Φ that integrates decision trees and structured perceptron with the extensions outlined in Chapter 4. Finally, we describe some modeling remarks and the parametrization of the obtained learning algorithm.

5.1
Attributes Representation

In natural language processing, generally, attributes are represented symbolically, or as categorical attributes. An example of categorical attribute is the attribute *word*. A sentence in natural language, as Portuguese, is represented as a sequence of words and every word can assume a value of a set called *vocabulary*. For example, the sentence *A casa é amarela* is represented as

$$word[1] = A$$

$$word[2] = casa$$

$$word[3] = é$$

$$word[4] = amarela$$

The categorical representation is appropriate for certain types of classification algorithm based on information theory, as is the case of decision trees.

However, algorithms based on mathematical models need numeric attributes to represent categorical attributes. So, the original corpus of natural language which is represented in a categorical manner must be *binarized* to be used by SVM and the sparse perceptron. Each pair of attribute and value is assigned a unique index binary attribute. After selecting attributes, the domains of categorical attributes are compressed, in accordance with the selected binary attributes. When an attribute value is discarded by the perceptron, it is encoded as a value *dummy* in the input of the decision tree. The Table 5.1 illustrates the transformation of the domains of the attributes after regularization of the features *word* and *lemma* for the initialization step in the scheme outlined before.

Attribute	**Value**	**Binary Attribute**	**Selected Attribute**	**Attribute**	**Value**
word	conseguem	5	yes	word	conseguem
word	sempre	716	yes	word	sempre
word	tenha	3	yes	word	tenha
word	casa	201	no	word	"*dummy*"
word	Brasil	43	no	word	"*dummy*"
lemma	eleger	72	yes	lemma	eleger
lemma	merecer	137	yes	lemma	merecer
lemma	renovar	82	yes	lemma	renovar
lemma	avaliar	78	no	lemma	"*dummy*"
lemma	prezar	302	no	lemma	"*dummy*"

Table 5.1: Example of the compression of the domains of categorical attributes *word* and *lemma* after the initialization step of our proposed method.

Given a categorical attribute *word* that has a vocabulary V dimension $|V| = n$, the binarized representation of this attribute is given n binary indicator functions defined as

$$x_i = \begin{cases} 1 & \text{if } V_i = V_j \\ 0 & \text{if } V_i \neq V_j \end{cases}$$

where $0 < i, j \geq n$.

5.2 Learning Algorithm

Our main goal is to generate regularized nonlinear models with high performance by incremental cycles of compression and expansion of the feature joint vector Φ. Next, we describe how the compression and expansion steps are performed.

Since the Structured Sparse Perceptron (SSPerc) jointly performs feature selection and the estimation of the model by combining a small set of features, we are interested in the selected features. The goal is to reduce the dimensions of the feature space by taking into account only the set of selected features, which are the features that exceed the minimum updates threshold L. Hence we are interested in the update counters that SSPerc maintains. We propose to modify the algorithm to return $K \leftarrow \{k \mid u_k \geq L\}$. This modification corresponds to the compression step because its intention is to reduce the dimensions of the feature vector Φ. The modified algorithm is presented in Figure 5.1.

Input: $D = \{(x, y)\}$ binary data set,
C loss constant,
L update counters threshold,
$P_{dropout}$ probability of dropout
Output: w
K
$w^0 \leftarrow 0$
$t \leftarrow 0$
while no convergence
  for each $(x, y) \in Shuffle(D)$
    $\widehat{y} \leftarrow arg\, max_{\overline{y} \in Y(x)} \sum_{(i,j) \in \overline{y}} s'_\ell(x, y, i, j; w, u, L)$
    $w^{t+1} \leftarrow w^t + \Phi(x, y) - \Phi(x, \widehat{y})$
    for all i **s.t** $\Phi(x, y) \neq \Phi(x, \widehat{y})$
      $r \leftarrow random[0, 1)$
      if $r \geq P_{dropout}$
        $u_i \leftarrow u_i + 1$
    $t \leftarrow t + 1$
  for all i **s.t** $u_i < L$
    $u_i \leftarrow 0$
$w \leftarrow \frac{1}{t} \sum_{k=1}^{t} w^k$
$K \leftarrow \{k \mid u_k \geq L\}$
return (w, K)

Figure 5.1: Modified Structure Sparse Perceptron algorithm with consideration of large margin and dropout.

The original algorithm includes the feature generation component based on Entropy-Guided Feature Generation (EFG) algorithm, which is included as a preprocesing step in SSPerc. The induction corresponds with the expansion step because it is responsible for the increment of the dimensions of the feature vector Φ. For a better comprehension of the incremental scheme, we have removed this preprocessing step with the goal of describing in details how this

is accomplished.

Our incremental scheme works using cycles of compression and expansion of the set of attributes and uses as main classifier the Structured Perceptron (SPerc). After any modification made on the feature vector Φ, we must learn a model with the new representation. In our approach this is achieved by SPerc. In this case, our SPerc coincides with SSPerc with $L = 0$ and $P_{dropout} = 1$. The pseudo-code of the proposed algorithm is shown in Figure 5.2.

Input: *cycles* stop condition,
$D = \{(x, y)\}$ binary data set,
C loss constant,
L update counters threshold,
$P_{dropout}$ probability of dropout
Output: w
$K \leftarrow StructuredSparsePerceptron(D, C, L, P_{dropout})$
$D \leftarrow CompressDataSet(D, K)$
$w \leftarrow StructuredSparsePerceptron(D, C, 0, 1)$
$c \leftarrow 0$
while $c < cycles$
$D_{categorical} \leftarrow ToCategorical(D)$
$decisionTree \leftarrow C_{4.5}(D_{categorical})$
$templates \leftarrow ExtractTemplates(decisionTree)$
$D \leftarrow ExpandDataSet(D, templates)$
$K \leftarrow StructuredSparsePerceptron(D, C, L, P_{dropout})$
$D \leftarrow CompressDataSet(D, K)$
$w \leftarrow StructuredSparsePerceptron(D, C, 0, 1)$
$c \leftarrow c + 1$
return w

Figure 5.2: Incremental scheme.

The first step towards an expansion of the feature representation is performed with the preparation of the data set that is used in the templates generation stage. In this case, the local decision variables correspond to edges, since the prediction problem is to choose some edges from all candidate edges between tokens. Thus, given a sentence in the input dataset, we generate a decision tree example for each candidate edge. The *binary* decision variable indicates whether an edge is included in the correct dependency tree or not. Then, the decision tree algorithm learns the output tree by predicting whether an edge is correct or not. Next, the templates are extracted from the paths of the trained decision tree. The expansion is completed when the generated templates are instantiated as derived features in Φ.

Here we are basically performing conjointly the feature generation and selection for a structured problem, and then, we learn a regularized and non linear model using SPerc. In order to improve the first template generation step, we only consider the set of the most informative basic features. For that reason the method is initialized with a compression step performed by the SSPerc.

Next, we outline some modeling considerations regarding to the incremental scheme that we propose here. Such remarks are about the counters initialization in the compression step and concerning to the conversion between categorical and binary data.

5.2.1 Modeling Remarks

At each cycle, attributes that have been selected remain throughout subsequent processing. For this behavior, from cycle 1, the vector u is initialized with counter L for all attributes selected in the previous cycle, such that in the perceptron learning those attributes starts already active. In cycle 0, the input parameter U_0 receive $\mathbf{0}$.

5.3 Experiments Parametrization

Each experiment processed by the incremental scheme has a configuration profile that specifies all the required parameters. Next, we describe the input parameters of the method.

The first group of parameters are for the stop conditions of the built-in algorithms. The parameter *cycle* is used as stop condition for the incremental scheme that we propose here, as well the parameter *epochs* is used as the stop condition of the perceptrons algorithms that are built-in the scheme.

The parameter C, used in the large margin extension of SPerc, is a constant to balance the weight between the loss function and the learned edge weights.

The second group of parameters are related with the compression step, this only with the SSPerc, the parameter L is the lower bound of the number of updates of the weight vector in which a feature must participate. The $P_{dropout}$ parameter, this is the probability of dropping a feature ϕ, was introduced for a fine-grain selection of attributes.

Finally, the parameters related with the expansion step, that is related to the decision tree learning. The extraction of the decision tree models is defined

by a minimum length and a maximum length of the paths taken in the tree. Typical sizes templates range from two to four (17; 61).

5.4 Implementation Remarks

Following the line of research at the LEARN lab on Multilingual Coreference Resolution, this section gives details regarding the implementation of the proposed solution in order to be reused in the development of a coreference analyzer.

The implementation of the proposed solution is developed using C# language in Visual Studio Community 2015. Since the key components of our solution are the data preparation stage and the learning algorithm, we show the classes involved in the respective components on Figures 5.3 and 5.4

In the data preparation stage, we have the classes that model the proposed filter, that from the sentence representation builds a sparse graph. Then, we have the classes representing a graph, the generic abstract class *Graph* and the generic class *DirectedGraph*.

Specifically, *CandidateBuilder* filters the edges of candidates graphs and *GoldenBuilder* builds the standard golden dependency trees. We also have a class to create the derived features, like counters.

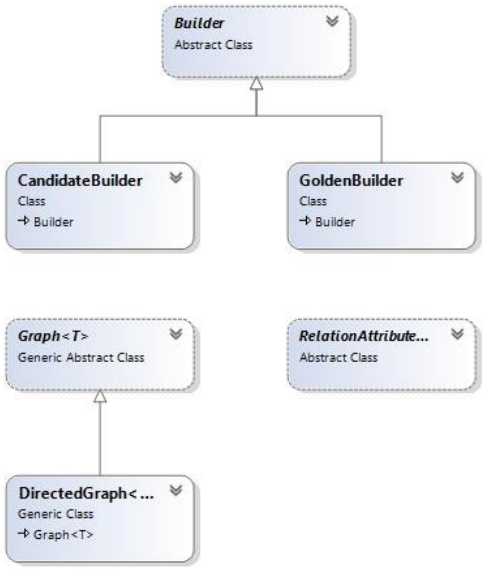

Figure 5.3: Classes involved in the data preparation stage.

Regarding to the classes involved in the incremental scheme, we have a class *Templates* to represent a template, that is a conjunction of basic features. It is related to the feature generation component that is implemented in the class *FeatureInduction*.

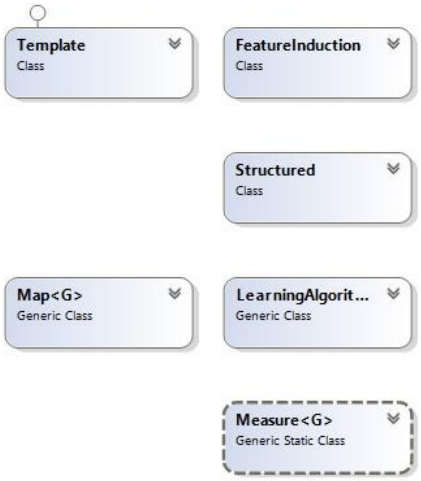

Figure 5.4: Classes involved with the learning algorithm.

Our feature induction implementation is based in the Accord.NET Framework [1], that is a .NET machine learning framework completely written in C#. From this framework, we use the classes related to decision tree learning. Among them we use *Codification*, *DecisionVariable*, *DecisionTree* and *C45Learning*.

The implementation of the Structured Perceptron and its respective extensions are specified in the class *Structured*. We also place there the implementation of Tarjan for the algorithm of Chu-Liu-Edmonds.

The attributes representation is specified in the generic class *Map*. It contains the methods that perform the compression and the expansion of the feature representation vector. It also has the procedures that apply the binary codification over categorical data. Finally, the incremental scheme is implemented in the class *LearningALgorithm*, which follows the pseudocode described here. Our evaluation methods are specified in the class *Measures*.

5.5 Chapter Conclusions

The incremental scheme is a based on cycles of compression and expansion of the feature representation vector Φ. It combines decision trees with structured perceptron and extensions. Such extensions are the core of our method. The Structured Sparse Percetron performs the selection of the attributes and the Entropy-Guided Feature Generation performs the induction of feature with non linear pattern given a set of basic features. Therefore, we obtain a procedure that generates regularized and nonlinear models, that

[1] http://accord-framework.net

are desirable because they speed up the training, are compact and with great generalization power.

6
Empirical Evaluation

Dependency parsing refers to the task of finding the syntactic structure underlying a sentence. It involves the prediction of the dependency tree describing the sentence structure. Hence, it can be modeled as a structured problem.

The CoNLL 2006 Shared Tasks have created a standard for: task formalization, data sets extracted from available treebanks and evaluation metrics. This standarization has served to set up a common input ground for the parsers and their further evaluation. In the years following this set up, several machine learning approaches adopt this standard to evaluate their performances.

Here, our goal is to empirically evaluate the proposed incremental scheme. To accomplish this, we propose to test our scheme by using the Portuguese data set from the CoNLL 2006 Shared Task and the standard metrics proposed in that competition. This chapter presents the experimental setup, *corpus* statistics and performance results of our proposed machine learning approach to DP.

First, the Portuguese *corpus* is described and analyzed, as well as the evaluation metrics. Then, the application of the proposed machine learning solution is described. Finally, we report the achieved performance on the experiments, their modeling particularities and specific parameter setting.

6.1
Corpus

In 2006, 2007, 2008 and 2009, DP related tasks have been part of the Conference on Natural Language Learning Shared Task. For the first two years, the task has been to solve the dependency parsing problem itself for a wide range of languages. For the remaining two years, a joint task on syntactic and semantic dependencies has been proposed. Since this work is focused on syntactic dependencies for Portuguese, it is mainly concerned with the task of the first year.

In 2006, *corpora* for thirteen languages were made available, namely: Arabic, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish, Turkish and Bulgarian (8). However, only four of those *corpora* are still publicly available, namely, the Dutch, Danish, Portuguese

and Swedish. Here, we are interested in the Portuguese *corpus*, which is the *BOSQUE* part of the *Floresta Sin(c)táctica* (1; 22).

By the occasion of the competition, the Portuguese treebank has been converted into a dependency treebank, since it is originally a phrase structure treebank. This *corpus* provide the following features: word form, token position, lemma of the word, coarse-grained part-of-speech, fine-grained part-of-speech and a list of set-valued syntactic and morphological features. Table 6.1 provides statistical information about the Portuguese *corpus*.

Statistic	**Result**
Number of Tokens	212 545
Number of Sentences	9 359
Tokens per sentence	22.8
Number of different coarse *postags*	15
Number of different fine *postags*	21
Percentage of punctuation tokens	14.2%
Percentage of nonprojective relations[1]	1.3%
Percentage of sentences with at least one nonprojective relation[1]	22.2%

Table 6.1: Portuguese *corpus* Statistics.

Information regarding the set of part-of-speech tags for coarse-grained and fine-grained features is presented in Appendix A. This *corpus* has been partinioned into a training and a test set, by the occasion of the conference. This work follows the same division. In Table 6.2 we provide basic statistics about these partitions.

Partition	**Sentences**	**Tokens**
Train	9 071	206 678
Test	288	5 867

Table 6.2: Partitions of Portuguese *corpus*.

Additionally, we use 10% of the training data as a validation set, in order to choose the values for our algorithm meta-parameters.

6.2 Evaluation Metrics

To evaluate dependency parsers the three most common metrics are the *labeled attachment score* (LAS), the *unlabeled attachment score* (UAS) and the *label accuracy* (LA). LAS is the percentage of tokens where the system correctly predicts both its head and the relation type that the token holds

[1]Including nonscoring tokens

with its head. UAS is the percentage of tokens where the system correctly predicts its head, whereas LA is the percentage of tokens with correct relation type.

For both the CoNLL 2006 (8) and the CoNLL 2007 (51) shared tasks, LAS is used as the main evaluation metric. Nevertheless, systems results are reported for all three metrics. One difference between the competitions on those years was about including or not punctuation marks as scoring tokens.

In this work, we use UAS as the evaluation metric, since our concern here is the prediction of correct token heads when learning unlabeled dependency trees. Following the rules of the competition that were held in 2006, we don't consider punctuation marks as scoring tokens.

6.3 Portuguese Dependency Parsing

In the past decade, DP has attracted a lot of attention. Fast progress has been made on improving the performance of dependency parsers. Here, we are interested on reviewing the systems that show higher performance on the Portuguese CoNLL-2006 dataset. We compare them with our proposed solution. Furthermore, we use the *unlabeled attachment score* (UAS) to score the reviewed systems. UAS is defined as the percentage of tokens that are correctly attached to their head tokens, that is, the percentage of correct arcs in the predicted tree. Here, we don't consider punctuation marks to calculate UAS.

In 2005, the MST Parser system proposed by (40), achieved state-of-the-art performance on different datasets. In the next year, the CoNLL-2006 Shared Task (8) were devoted to multilingual DP. By applying an extension of the MST Parser that uses *second-order features* and showing a 91.36% of UAS, (41) achieved the best performance.

MST Parser'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 (43). (43) proposed an approximation algorithm to this problem and showed that the second-order model outperforms the first-order one, even using approximated prediction.

As far as we know, the best performing system on the Portuguese CoNLL-2006 dataset is the dual decomposition system proposed by (31). This system introduces a new algorithm to perform approximated prediction with second- and third-order features. However, the second-order features used in this

Parser	Year	Learning Algorithm	Basic Features	UAS
Koo et al.	2010	MIRA	3rd order	93.03
Martins et al.	2013	MIRA	2nd order	92.71
Fernandes	2012	ESL	1st order	92.66
Motta	2014	IFIS	1st order	92.01
Mcdonald et al.	2006	MIRA	2nd order	91.36
Crestana	2010	ETL	1st order	89.74
This Work	**2016**	**S-IFIS**	1st order nonregularized	**92.96**
This Work	**2016**	**S-IFIS**	1st order regularized	**92.83**

Table 6.3: Portuguese Dependency Parsing best reported parsers.

system are slightly different from the features used in MST Parser, 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 (43).

More recently, also based on dual decomposition algorithm, (34) attained the second-best result for this task with the Portuguese data set. Their second-order model obtains 92.71% of UAS. Furthermore, dual decomposition has shown to lead to a certificate of optimality for the vast majority of the sentences.

The major goal of this work is to advance this line of research of the *Laboratorio de Engenharia de Algoritmos e Redes Neurais* (LEARN) research group. For that sake, we compare our approach with the parsers developed in this group. The schemes proposed by (48) and (13) are token based, whereas the parser proposed by (17) is graph based. This graph based parser uses the well known edge-based scoring function. The decomposition of the score of a candidate dependency tree is based on its edges. Hence, it belongs to the category of parsers that use first-order feature decomposition.

The *Entropy-guided Feature Generation* (EFG) has been proposed as a component of the *Entropy Guided Transformation Learning* (ETL) (60). EFG promotes the induction of new features with nonlinear pattern. (13) applies a token-based approach and uses ETL to build a system for the DP task. This system shows a 89.74 % of accuracy.

The token-based approach of (48) attains 92.01% of accuracy. It uses a supervised machine learning approach that incrementally induces and selects feature conjunctions derived from basic features. This approach integrates decision trees, support vector machines and sparse perceptrons. The resulting

machine learning framework is named IFIS – Incremental Feature Induction and Selection. The work of (48) uses EFG to model the feature generation component and formulates the prediction task as a multiclass problem.

EFG has been successfully integrated with the structured perceptron algorithm leading to a structured framework called *Entropy-Guided Structure Learning* (ESL) (17). ESL has been used to develop a Portuguese DP system. This system shows a 92.66% accuracy. This is the previous best result, regarding DP parsers developed at the LEARN lab. In Table 6.3, we summarize the best reported results for the Portuguese DP task, on the evaluation scenario of the CoNLL 2006 Shared Task. Also, we compare the results of the proposed incremental scheme against the best reported systems.

The performance of our method is evaluated in two scenarios. First, we report the result of a model that is trained without any regularization – this is our nonregularized model. It attains 92.96% of accuracy, which is our best result. Second, we report our best performance when using regularization techniques. This regularized model shows a 92.83% accuracy. Both models outperform the second best reported result, being also very close to the state-of-art system (31).

6.4 Results

This section is devoted to the experimental results achieved with the incremental scheme. The evaluation of the method is performed using the Portuguese *corpus* provided by the CoNLL 2006 Shared Task. In Table 6.4, we specify the partition that we use to conduct the experiments. We randomly select 10% of the available training data to build a development set, which is used for the parameter setting. All results are reported using the UAS metric. Here we do not take into account punctuation marks as scoring tokens.

Partition	Sentences	Tokens
Train	8 164	185 974
Development	907	20 704
Test	288	5 867

Table 6.4: Partition of the Portuguese corpus used to perform the experiments.

Since this is a task that consumes many resources, particularly memory, our experiments were conducted in the Microsoft Azure cloud platform, using a configuration for computing memory and CPU intensive tasks. We use an A10 virtual machine with 8 cores and 56 GB of RAM. The A10 and A11 virtual machines feature Intel Xeon E5 processors and are ideal for high-performance

clusters, modeling and simulations, video encoding and other computer or network intensive applications.

Also we carried out part of our experiments in a D12 v2 virtual machine with 4 cores and 28 GB of RAM. As well as the A10 configuration, the Dv2-series instances are based on the latest generation 2.4 GHz Intel Xeon E5-2673 v3 (Haswell) processor and with Intel Turbo Boost Technology 2.0, which can go up to 3.2 GHz. Dv2-series and D-series are ideal for applications that demand faster CPUs, better local disk performance, or higher memories. Both virtual machines were utilizing Microsoft Windows Server 2012 R2 operating system.

The incremental scheme integrates several machine learning techniques to predict the syntactic structure of a given sentence. In particular, it uses a decision tree to generate templates in order to include features with nonlinear pattern. Also, it uses a sparse perceptron to perform the selection of the most informative attributes and a structure perceptron as the main classifier.

The proposed method is always trained using the arcs filtering procedure. For each instance, we provide *standard golden information*[1] regarding the head to be used by the arcs filter. In this case we provide the correct *postag* and side of the candidate head. In order to test the performance of the obtained model, the development and test data sets are filtered with predicted information regarding the head. This information is obtained from the outputs of the predictors of head *postag* and head side developed by (48). Particularly, the coarse-grained tag set specified in Appendix A was used as the class set for the predictor of head *postag*.

We believe that the proposed filter improves the accuracy of all the learning algorithms comprised in the incremental scheme. This is because it reduces the number of unpromissing candidate arcs, what makes more expressive the number of standard golden edges. Therefore, we tested the performance of the scheme with filtered candidate structures and compared it against a model trained with unfiltered structures. In this case, the parser developed by (17) helped us in the comparison. It corresponds with the nonregularized version of our scheme when it is trained with unfiltered candidate structures. To accomplish this, we trained a model that only performs feature induction and does not include any regularization extension.

In a second group of experiments we tested the performance of models trained with regularization extensions. We showed the sensibility of the method to different values of the parameter *update counters threshold* and to dropout

[1] *Standard golden information* is a term that refers to the correct information or 100% accurate. In other words to information that is taken as ground of truth.

intensity.

The last experiment measures the impact of arcs filtering method based on the quality of auxiliary predictors. Such predictors provide information related to the head, particularly its *postag* and its side. As the method uses predictions that are not 100% accurate to filter candidate edges, it is interesting to test it in ideal conditions. In this case, we tested a model with standard golden information. We compared this set up against a data set that uses predicted information in the filtering. In this scenario, we can evaluate the system gain if we improve the accuracy of the auxiliary predictors.

6.4.1 Impact of Candidate Edges Filter

For the task of dependency parsing, we incorporate a filter of candidate edges, which we believe improves the accuracy of the learning algorithms comprised in the incremental method. As our incremental scheme is based on cycles of compression and expansion of the feature representation of the training examples, we measure the power of the proposed filter in the initialization cycle of the method without any regularization extension. We hope this experiment shows an improvement in performance of the Structured Perceptron (SPerc) used as main classifier.

The training is performed on the initialization cycle with different values of the margin to measure the sensibility of the accuracy against this parameter. As stop condition of the SPerc we use 10 epochs of training. The Figure 6.1 shows on the left the results for the training set and the results on the right are for the development data set.

The best result for the development data set was obtained when the margin is equal to 400. The performance of the corresponding model was measured on the test data set and 91.93% of accuracy was obtained. This result compared against (13) and (41) shows that, in this scenario, our method performs better. Hence, we concluded that just the application of the filter benefits the parser's performance. In other words, the main classifier obtains a better result when the number of unpromising edges is reduced.

This improvement is due to the fact that we moved from training with complete candidate graphs, in which the relation between unpromising edges and the number of edges in a dependency tree is large, to a better balanced scenario. In Table 6.5 we show statistics regarding the filtered structures that we are using to the empirical evaluation of our method.

The mean density obtained from the filtered candidate structures points outs that such structures are very sparse. Another important measure associ-

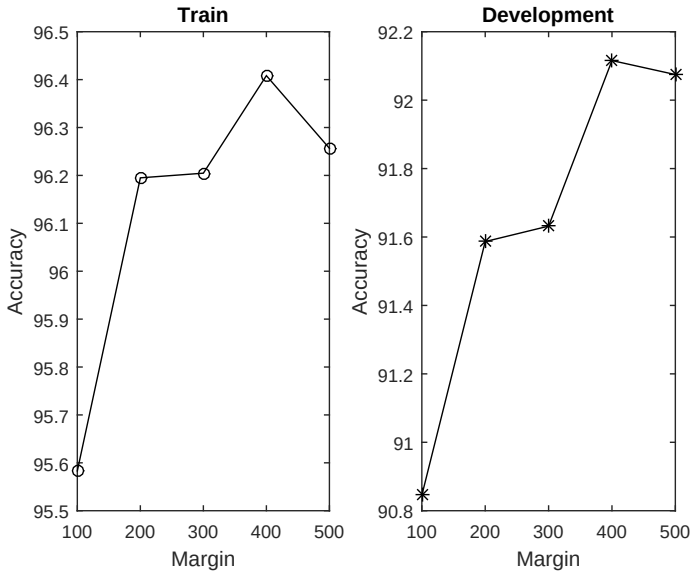

Figure 6.1: Sensibility of the accuracy against different values of the margin when performing the initialization cycle. On the left we show the results for the training data set and on the right for the development data set.

Partition	Mean Density	Mean Recall (%)
Train	0.14	100
Development	0.15	93.51

Table 6.5: Mean density and recall of standard golden edges for the chosen partition.

ated with the quality of the filter is the mean recall of *standard golden edges*[2], this measures the percent of correct edges in the candidate graph. In the training set the recall of golden edges is 100% because we used standard golden information to filter the structures. On the other hand, the mean recall in the development set is 93.51%, due to the fact that the information used in the filtering stage is obtained from auxiliary predictors that are not 100% accurate. This means that the measure of recall is an upper bound for performance. However, as this experiment demonstrated, the filter can benefit the main classifier by improving its accuracy. Next, we investigated the power of the filter when combined with feature induction.

Impact on the Feature Induction

We believe that the accuracy of the decision tree used to generate the templates in the feature induction step is improved when using the proposed filter. To prove this claim, we add to the initialization cycle another cycle with

[2] *Standard golden edge* refers to an edge that belongs to the structure that is taken as ground of truth.

feature induction; all this without any regularization extension. We compared the result with the model of (17), which coincides with our model when it is trained with unfiltered candidate structures. The training is carried out with different values of the margin, to measure the sensibility of the accuracy to this parameter. We use 10 epochs as stop condition of the SPerc algorithm. Following (17) and (61), we use templates with lengths in the range from 2 to 4. The Figure 6.2 shows the results of this model for the training and development sets.

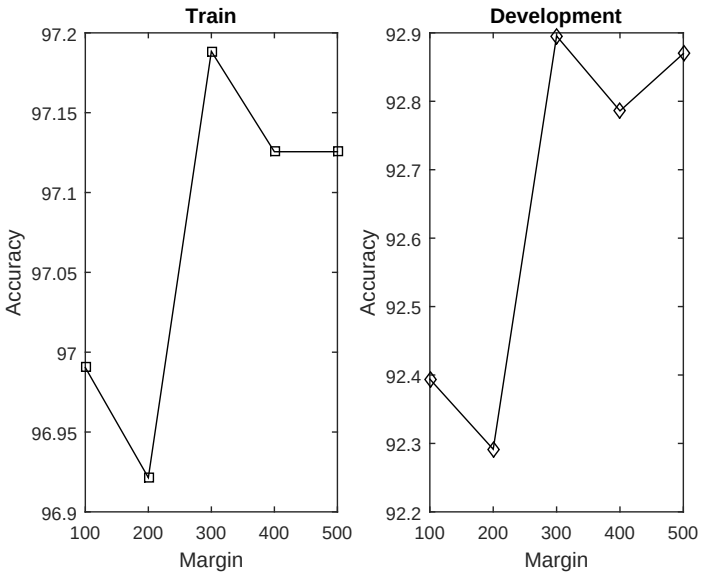

Figure 6.2: Sensibility of the accuracy against different values of the margin when performing the initialization cycle and a cycle of feature induction. On the left we show the results for the training data set and on the right for the development data set.

In this scenario and for the development data set, the best model was obtained when the margin is equal to 300. We measured its performance on test data set and **92.96%** of accuracy was obtained. This result compared against the unfiltered case, (17), shows an improvement of performance. This is because the decision tree performs the training with a reduced number of unpromising edges and a more representative number of edges that belong to a dependency tree. In other words, we are promoting the balance of the number of examples in the classes used to train the decision tree. Therefore, we concluded that the filter helps to improve the performance of two of the main components of the incremental scheme: the decision tree algorithm and the SPerc. This model outperforms the second best reported result (34), being very close to the state-of-art system (31).

A highlight on the statistics of the selection of the model, when using

the development data set, is its size. In Table 6.6 we detail such statistics. In the initialization cycle, the learning of the model's weights starts with 328 665 binary attributes. After the induction are obtained 7 templates. When such templates are instantiated the number of binary attributes grows to 428 625, which represents an increment of 23.33%.

Another particularity is that the number of nonzeros of the final model, the obtained after the induction, is 87 483. This means that our embedded extension of regularization, models of averages, is working; although its size is still regarded as large.

Cycle	Initial Attributes	Nonzeros
0	328 665	86 482
1	428 625	87 483

Table 6.6: Statistics of the nonregularized model obtained with the margin parameter set to 300.

Another peculiarity, is regarding the time spent training this model. In this particular case, the training of the decision tree algorithm took 6 hours to be completed because of the large amount of decision variables that were used. In total the experiment took 6 hours and 30 minutes to complete.

In this scenario, it is necessary to use regularization techniques. Our goals are to reduce training time, to select the most useful features for the model's learning and to promote compactness which is known to improve the generalization power. We use two regularization techniques introduced in the SPerc: dropout and *update counters threshold*. Next, we report the results of the regularized model.

6.4.2 Impact of Regularization

Sparse linear models have become a powerful tool due to its ability to jointly perform feature selection and the estimation of the model. It is interesting to select only the features that should be used to build the model. In the scenario of feature induction the amount of binary attributes that are generated by the templates can lead to a potential overfitting of the training data. Also, it might introduce a large number of irrelevant attributes as was shown in (48). Hence, to control overfitting and the size of the model, we introduce regularization techniques based on sparse linear models. Our main goal is to promote more compact models, which are desirable because they facilitate interpretation, they take less time to be trained and occupy less computational resources.

Up to this point we have tested our method without any extension of regularization. We are interested on investigating the efficacy of two particular regularization techniques. First, the *update counters threshold* technique, which was introduced in SPerc for the promotion of sparseness. Second, the dropout technique, that as shown in (48), is an adaptation for the perceptron algorithm of the dropout used in neural networks. The intention is to smooth and refine the selection performed by the counters threshold. So, the compression component is based on Structured Sparse Perceptron (SSPerc).

First, we present a group of experiments referring to the setting of the threshold parameter and its contribution to accuracy improvement. In a second stage we investigate the impact of the dropout when combined with *update counters threshold*.

Update Counters Threshold

Inspired on the number of times that a feature participates in updates of the model, (23) proposed the extension that we are referring here as *update counters threshold*. The intention is to establish a minimum bound of necessary updates that a feature must accomplish in order to become relevant for the learning of the model. To include this extension the scoring function is modified with the main goal of use the update counters as a mask, this is to include ϕ_i in the scalar product only if it has participated in minimum L updates. For example, if we have $(\phi_1(x,i,j),\ldots,\phi_3(x,i,j))$ basic features for the edge (i,j) and $u=[1,2,3]$ with $L=2$ the score of the edge is $w_2{*}\phi_2(x,i,j){+}w_3{*}\phi_3(x,i,j)$ because ϕ_1 does not meet the constraint of minimum updates.

Our method is based on cycles of compression and expansion of the feature representation. The compression is performed with the selection of the most informative attributes, which helps to control the size of the model. The expansion is performed with the induction of features with nonlinear pattern. Our main concern here is to investigate the impact of the *update counters threshold* extension as part of the compression step. We based our experiments on the value of the margin for the nonregularized model. So we adjusted the threshold parameter using the margin equal to 300. Also, we used 10 epochs of training as stop condition for the SPerc and the SSPerc.

The application of the arcs filter helped to improve performance, so we wanted to test its combination with this regularization technique. We performed the initialization cycle guided by the update counters threshold extension and in Figure 6.3 we depicted the results for the training and the development sets.

In this scenario, the best result for the development set is obtained when

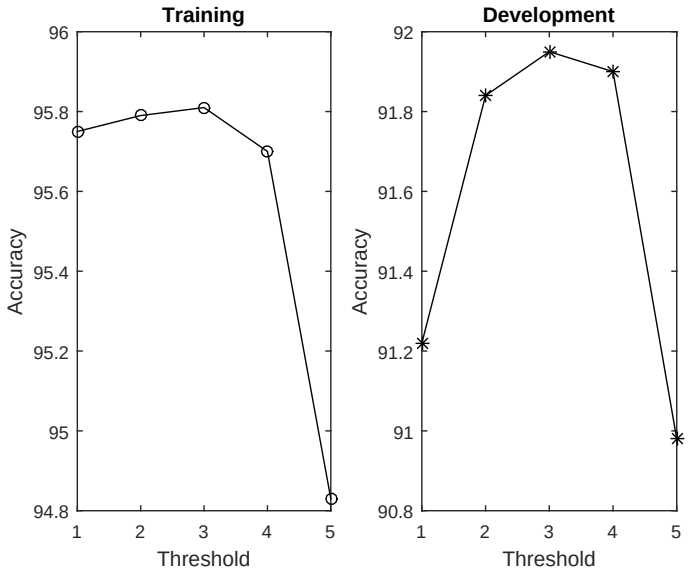

Figure 6.3: Sensibility of the accuracy against different values of update counters threshold. The obtained results are for the initialization cycle, on the left is shown the training set results and to the right the development results.

the threshold has value 3. Using the generated model on test data set we obtain 91.00% of accuracy. As expected, its nonregular counterpart performs better. When compared with the best results reported, this model only outperforms (13). This let us conclude that only the attributes selection combined with arcs filter is not sufficient to describe the phenomenon of interest. In this case, is required a clever mechanism to introduce more features to help in the learning.

We combined this setup for the initialization with a cycle of induction and selection. As the method is initialized with a compression step, the induction is made over the selected attributes. Hence, this helps with the refinement of the primary selection because we are adding templates built only with relevant features. We claim that this configuration improves the accuracy of the final model. With this configuration we obtained competitive results. In Figure 6.4 is shown the results for the training and development sets.

The best results are attained when the threshold parameter is set to 5. Using this model on the test data set we obtain 92.65% of accuracy. This is because the induction helps to refine the initial selection which improves the final accuracy. This model has a better performance than (48), (41) and (13) approaches.

In this scenario of an initialization cycle with compression and a full cycle with compression and expansion using the update counters extension active, we obtain the results reported in Table 6.7 regarding to the regularization

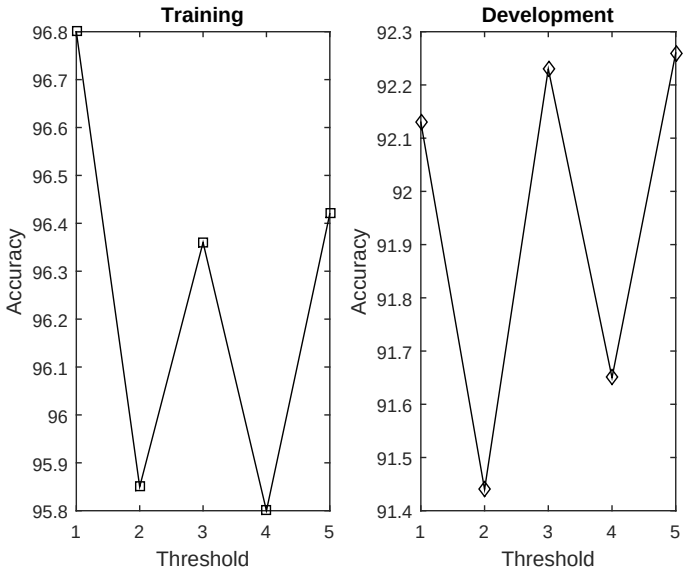

Figure 6.4: Sensibility of the accuracy against different values of update counters threshold. The obtained results are for the initialization cycle and another cycle of induction and compression, on the left is shown the training set results and to the right the development results.

power.

Cycle	Initial Attributes	Nonzeros	Regularization (%)
0	328 665	9 295	97.17
1	106 317	12 285	88.27

Table 6.7: Statistics of the regularized model obtained with parameters of update counters threshold in 5.

The obtained results showed that our method has a great regularization power. It discarded 97.17% of the basic features that served as input for the method in the initialization cycle. As well, after the induction it compressed the model in 88.27%. This model obtained a good result when compared against its nonregularized counterpart.

Regarding the time spent on the training of this model, we can say that the most time consuming component, when compared against its nonregularized counterpart, is the training of the decision tree algorithm. In this context this component takes a mean time of training of 20 minutes. The experiment took 45 minutes to be completed. Therefore, when compared against the nonregularized model we have reduced the time of training from 6 hours to 45 minutes. So, we concluded that the employed regularization technique drops dramatically the size of the final model and the time spent on training, which in turn helps to improve performance.

Next, we combined this technique with the dropout technique proposed by (48) to investigate the impact of regularization in this scenario.

Combining Update Counters Threshold with Dropout

In order to refine the selection made by the *update counters threshold* extension is proposed dropout. The goal of dropout is to promote regularization through the elimination of highly correlated attributes. This allows a better control over the number of selected attributes. It is based on the probability of discarding a feature even if it participates in an update of the model. Hence, it allows a fine selection of the attributes through the variation of this probability. For example, low values of the dropout probability cause low increments of the update counters. On the other hand, values near 1 obtain similar results to the *update counters threshold* extension.

An important remark is that there is no purpose of training with dropout and without the threshold extension. This is because it only intervenes in the update rule of the attributes update counters. In other words, if the threshold is 0 this means that every attribute participates in the score of the edges, hence is irrelevant if we increase the update counters or not.

The conducted experiments showed that the best result for the development set was obtained with values of the threshold in 3 and dropout in 0.9. We used as stop condition of the perceptrons 10 epochs and a margin constant of 300. For the template generation we consider templates lengths between 2 and 4. We obtained **92.83%** of accuracy on the test set with the model trained with the proposed parameter setting.

This model outperforms the second best result (34), being also very close to the state-of-art system from (31). As expected, its nonregularized counterpart has a better performance. However, it outperforms the regularized model obtained with *update counters threshold* technique. This might be explained in the number of attributes or the percent of regularization that the model suffers when such techniques are employed.

Regarding the statistics on model's size, in the scenario of the initialization with compression and a cycle of induction and selection, are discarded 96.17% of the total of binary attributes. The Table 6.8 reports such statistics.

Cycle	Initial Attributes	Nonzeros	Regularization (%)
0	328 665	14 138	95.69
1	142 327	17 746	87.53

Table 6.8: Statistics of the regularized model obtained with parameters of update counters threshold in 3 and dropout in 0.9.

The size of this model is larger when compared against the model trained with *update counters threshold* technique. Also, it attained a better performance. This increase in the size of the model helped to improve performance which means that we refined the selection made by *update counter threshold* technique. Regarding the time spent in the training of this algorithm, we can say that it has similar measure when compared with the model without the dropout extension. Hence, we concluded that with this combination we have improved the trade-off between accuracy and model's size.

6.4.3 Impact of Standard Golden Information on Auxiliary Predictors

Up to this point we have demonstrated the power of the proposed filter of candidate edges when combined with several machine learning techniques. Until now we always have used predicted information to evaluate performance. The discussion now turns to the scenario of getting more accurate information to filter our candidate structures.

As we trained our models with *standard golden information*, it is interesting to test our best model in this scenario too. Our main goal here is to investigate the impact in accuracy if an improvement is made in the filtering stage. In other words, this will give us a measure of the gain in accuracy that is obtained if the recall of *standard golden edges* is improved.

The predictors used in the filtering step are not 100% accurate, Table 6.9 shows the accuracy of the used auxiliary predictors. When the filter is applied with predicted information to the development dataset, it leads to a recall of 93.51% which sets an upper bound for accuracy in such dataset. Hence, we wanted to test the performance of our best model when the upper bound set by the recall of *standard golden edges* is 100%. Such condition is achieved when the candidate structures are filtered with *standard golden information*.

Predictor	**Accuracy (%)**
Head POS	95.67
Head Side	98.90

Table 6.9: Accuracy of the auxiliary predictors used in the filtering step.

When predicted information is used, our best model which is the non-regularized one, attains 92.86% of accuracy in the development dataset. To conduct our experiment we used the development dataset because it allows us to relate the results with the recall. In this case we have filtered the dataset with *standard golden information* and tested the model. In this scenario we

obtained 94.67% of accuracy, Table 6.10 summarizes the results for the development dataset when predicted and *standard golden information* is used.

Filtering Information	**UAS (%)**
Predicted	92.89
Standard golden	94.67

Table 6.10: Accuracy of the nonregularized model on the development dataset. First, we used predicted information to filter the candidates structures. Also, we include the obtained result when *standard golden information* is used in the filtering step.

When *standard golden information* is used in the filtering stage we envisioned a clear improvement in the accuracy. We had a 1.88% of improvement, allowing us to outperform the state-of-art system from (31). Hence, we concluded that if we work towards the improvement of the auxiliary predictors used in the filtering stage, the results of the general task are going to be benefited. Next, we performed an error analysis of the auxiliary predictors and discussed a solution to circumvent the errors of such predictors.

6.5 Error Analysis

We observe that the division into less complex subtasks to solve the parsing problem benefits the learning algorithms involved in the components of our incremental scheme. As shown in the empirical evaluation, the best results are obtained from the combination of the candidate edges filter and the feature induction.

The candidate edges filter is used as a preprocessing step, hence any error made in this stage is propagated through the subsequent components of the proposed method. In this case the recall of *standard golden edges* obtained after the application of the filter sets an upper bound for performance. For example, after the filtering stage the recall in the development dataset is 93.51%, which means that the performance on this dataset can't be above this measure.

Our filter is based on the outputs of the auxiliary predictors of head *postags* and head side. In this section is discussed an error analysis made on the head *postags* auxiliary predictor used to filter the candidate edges. We focus on this predictor because it has 95.67% of accuracy and is outperformed by the head side predictor. Our goal here is to propose a solution to circumvent the errors of such predictor and improve the final accuracy of the method. The Table 6.11 shows the most common errors made by the head *postags* predictor.

When analyzing the errors of the head *postags* predictor we found that the most confusing items are verbs and nouns. Analyzing the mistaken

	v	n	prop	prp	adj	root	pron	adv	Total
v	7 401	176	10	34	12	42	4	13	294
n	152	6 543	36	22	16	9	1	2	251
prop	35	104	1 163	5	0	1	2	1	148
prp	55	42	8	3 157	7	1	2	3	120
adj	25	57	2	1	248	0	1	2	88
root	41	4	0	0	0	862	0	0	45
pron	19	15	4	2	0	2	67	1	43
adv	25	5	1	2	1	1	0	98	36

Table 6.11: Partial confusion matrix of the errors made by the head *postags* predictor. Here we show the most frequent mistakes made by the predictor in question.

predictions regarding nouns, we can say that in 60.55% of the cases it predicts verbs rather than nouns. On the other hand, the predictor confuses nouns with verbs in 59.89% of the mistakes made when analyzing verbs.

Inspired on this analysis, we want to pass for the structured perceptron the task of correcting the hard cases in which the multiclass predictor fails, that is on differentiating verbs from nouns. As future work, we propose to increase the recall of *standard golden edges* by not to filter edges linking dependent tokens with verbs or nouns. Therefore, is expected an increase on the density of the candidate structures. We expect to see an improvement on the performance of the proposed learning algorithm.

6.6 Discussion

Machine learning approaches to dependency parsing have used several strategies, from token based to structure learning. Following the line of research of the *Laboratorio de Engenharia de Algoritmos e Redes Neurais*, we propose an integration and enhancement of the machine learning approaches to dependency parsing proposed by (48), (17) and (13).

First, we decompose the main task into less complex subtasks. These are, the prediction of the *postag* of the head token given a dependent and the prediction of the side of the head token given the relative position in the sentence of the dependent. This two subtasks correspond to multiclass prediction problems.

In order to find the correct head token for a given dependent, we can define a measure of distance based on previous information, that is the number of tokens between the head and the dependent that have the same *postag* as the head token. Given such measure of distance between the head token and the dependent, we propose to solve in terms of encountering the correct distance

between the dependent and its head token based on the information of the head side and *postag*. This corresponds to the distance subtask proposed in (48) and (13).

To integrate the mentioned subtasks into a problem of structured prediction of the distance, we filter the unpromissing edges based on the information of the auxiliary predictors of head *postag* and head side. Here, we show the power of the filter when compared against the best reported results of the literature. We concluded that only the application of the filter helps improve the accuracy of the predictor. This improvement is due to the significant reduction on the number of candidate arcs. Instead of working with all the arcs as candidates, we filter just a few as candidates. As a result, the relation between unpromissing arcs and corret arcs in the candidate arcs graph is drastically reduced. This fact reduces noise, improving learnability.

In the second group of experiments we obtained the best result of our model, 92.96% of accuracy. We shown that the combination of the filter with feature induction is powerful. This is because the decision tree performs the training with a reduced number of unpromising edges and a more representative number of edges that belong to a dependency tree. In other words, we are promoting the balance of the number of examples in the classes used to train the decision tree. Therefore, the filter helps improve the performance of two of the main components of the incremental scheme: the decision tree algorithm and the Structured Perceptron.

As feature induction introduces a large number of binary attributes and can produce overfitting, we conducted a serie of experiments to obtain a regularized model. This is because the promotion of more compact models simplify interpretation, reduce training time and consume less computational resources. It turns out that our regularized model quality is comparable to the nonregularized version, although a little smaller. Particularly, it reduces the number of nonzeros in the final model in at least 80%, this is respect the number of nonzeros of the nonregularized model.

Also, we present some statistics about the structures obtained after the arc filtering. We found that the performance of the predictor is limited by the recall of standard golden edges of the filtered candidate structures. For example, we can not have a performance better that 93.51% on the development dataset. Inspired on this, in the last experiment we tested our best model using the development data set filtered with *standard golden information*. This experiment showed that the performance of the incremental scheme can be benefited from an improvement on the accuracy of the auxiliary predictors.

Finally, we concluded that the integration of our arc filter procedure with decision trees and structured perceptron is powerful when applied to the Portuguese dependency parsing task. Our resulting system is competitive with the state-of-the-art for the task.

6.7 Chapter Conclusions

Here we reported the empirical evaluation performed to the proposed solution. We have observed its superiority when compared against state-of-art approaches. The obtained results demonstrate that the filter of candidate edges benefits the incremental scheme and that our approach improves the performance of the dependency parsers.

7
Conclusions

Several techniques have been proposed to automatically solve the syntactic analysis problem. In this dissertation, we review the main topics referring to dependency-based syntactic analysis. These topics provide the theoretical foundation and the representation used to describe the structure underlying such analysis.

Additionally, we examined the work on data-driven approaches to dependency parsing. We highlight the efforts made by the NLP community on the CoNLL Shared Tasks. These shared tasks establish a standard for evaluating and comparing parsers. Particularly, we focused on the work on graph-based approaches to dependency parsing and their main characteristics when compare against their counterpart, the transition-based parsers.

Following a line of research of *Laboratorio de Engenharia de Algoritmos e Redes Neurais* (LEARN), we proposed an integration and enhancement of the machine learning approaches to dependency parsing proposed by (48), (17) and (13). The token-based approaches of (48) and (13) decompose the task into less complex subtasks related to the head token. They build a multiclass predictor for three head properties, namely: side, *postag* and relative distance to the dependent token. In this case, the measure of distance is the number of tokens, between the head and the dependent, that have the same *postag* as that of the head token.

In this work, we model dependency parsing as a structured prediction. The main goal is to identify, for each token, which token in the sentence is its corresponding head token. One of our major contributions is the filtering of candidate arcs, using the head side and head *postag* predictors. Taking this into account, we defined the machine learning subtask that is the focus of our research. Also, we described a set of features that capture dependency relations between the lexical items in the input sentence.

Due to its flexibility, robustness and simplicity, the perceptron algorithm is one of the most popular linear discriminant methods. It is also adapted to learn complex representations, such as those required for structured prediction. In this dissertation, we proposed the learning of dependency trees with the structured perceptron and its *dropout* and *sparse vector* extensions. Such extensions are the core of our proposal. We also combined this algorithm with

an induction method, which provides automatically features that represent nonlinear patterns.

Additionally, we proposed an incremental scheme that is based on cycles of compression and expansion of the feature representation vector. It combines decision trees with the proposed structured perceptron extensions. The Structured Sparse Percetron performs the selection of the attributes, whereas the Entropy-Guided Feature Generation performs the induction of features with nonlinear pattern from a given set of basic features. Therefore, we have a procedure that generates regularized and nonlinear models. As a consequence of these size reduced models, we obtained a speed up in the training phase. We reduced in 88% the time spent on training which in turn saves computational resources and improves learnability. On the other hand, these compact models show better generalization power.

In the empirical evaluation of our model, we observe that only the application of the arc filter helps improve the accuracy of the predictor. This improvement is due to the significant reduction of the number of candidate arcs, after we apply our arc filtering procedure.

Our Portuguese Dependency Parsing system achieves a 92.96% of accuracy on its nonregularized version. It is obtained with the initialization cycle using the basic features and just one cycle of feature induction. Our findings indicate that the combination of the arc filter with feature induction is powerful.

Our regularized model achieves a 92.83% accuracy. Nevertheless, it also shows a striking reduction of 96.17% in the number of binary features used by the model. Furthermore, it reduces the learning time in almost 90%, when compared to its nonregularized version. It is obtained with the initialization cycle using the basic features and a full cycle of compression and expansion. The full cycle uses the *dropout* and the *update counters threshold* extensions.

Future Work

We observe that the task decomposition into less complex subtasks that are sequentially executed helps to solve the dependency parsing problem. The learning algorithms involved in our incremental learning scheme efficiently use the information from the previous subtasks. As shown in the empirical evaluation, the best results are obtained by combining the candidate edges filter and the feature induction. Nevertheless, performance has the quality of the recall of standard golden edges as an upper bound. For example, the recall quality in the development dataset has 93.51% as an upper bound, because we

use predicted information for the arc filtering procedure. Hence, when tested in such dataset, our method can not perform better than that. By examing our head *postag* predictor confusion matrix, we observed that the most difficult head *postags* to discriminate are verbs and nouns. Therefore, we think that the result of the system accuracy would be improved if we don't include the arcs which are hard to discriminate for the head *postag* predictor in the filtering stage. On the other hand, following a line of research at the LEARN lab on Multilanguage Coreference Resolution (57; 20), we foresee the reuse of our parser implementation as a key component in the development of coreference resolution systems.

8
Bibliography

AFONSO, S. et al. Floresta sintá (c) tica: A treebank for portuguese. In **LREC**. [S.l.: s.n.], 2002. 6, 2.3.1, 3.1, 6.1

ALTUN, Y. et al. Hidden markov support vector machines. In **ICML**. [S.l.: s.n.], 2003. vol. 3, p. 3–10. 4.2

ATTARDI, G. Experiments with a multilanguage non-projective dependency parser. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Tenth Conference on Computational Natural Language Learning**. [S.l.], 2006. p. 166–170. 2.3.2

BICK, E. The parsing system palavras. **Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework**, University of Arhus, 2000. 2.3.1

BIKEL, D. M.; CASTELLI, V. Event matching using the transitive closure of dependency relations. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers**. [S.l.], 2008. p. 145–148. 2.3

BOHNET, B. Comparing advanced graph-based and transition-based dependency parsers. In **Proceedings of the International Conference on Dependency Linguistics (Depling)**. [S.l.: s.n.], 2011. p. 282–289. 2.3.2, 2.3.3

BRILL, E. Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. **Comput. Linguist.**, MIT Press, Cambridge, MA, USA, vol. 21, p. 543–565, dec. 1995. ISSN 0891-2017. Available from Internet: <http://dl.acm.org/citation.cfm?id=218355.218367>. 4.4

BUCHHOLZ, S.; MARSI, E. Conll-x shared task on multilingual dependency parsing. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Tenth Conference on Computational Natural Language Learning**. [S.l.], 2006. p. 149–164. 6, 2.3.1, 2.3.3, 2.3.4, 3.1, 6.1, 6.2, 6.3

BUYKO, E.; HAHN, U. Evaluating the impact of alternative dependency graph encodings on solving event extraction tasks. In ASSOCIATION FOR COMPUTA-

TIONAL LINGUISTICS. **Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing**. [S.l.], 2010. p. 982–992. 2.3

CARRERAS, X. Experiments with a higher-order projective dependency parser. In **EMNLP-CoNLL**. [S.l.: s.n.], 2007. p. 957–961. 2.3.3

CHU, Y.-J.; LIU, T.-H. On shortest arborescence of a directed graph. **Scientia Sinica**, SCIENCE PRESS 16 DONGHUANGCHENGGEN NORTH ST, BEIJING 100717, PEOPLES R CHINA, vol. 14, no. 10, p. 1396, 1965. 1.2, 4.1

COLLINS, M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10**. [S.l.], 2002. p. 1–8. 4.1, 4.2, 4.2, 4.2.1

CRESTANA, C. E. M. **A Token Classification Approach to Dependency Parsing**. Master thesis — Pontifícia Universidade Católica do Rio de Janeiro, 2010. 1, 1.5, 1, 2.3.4, 3, 3.1, 3.2, 3.4, 6.3, 6.3, 6.4.1, 6.4.2, 6.4.2, 6.6, 7

CULOTTA, A.; SORENSEN, J. Dependency tree kernels for relation extraction. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics**. [S.l.], 2004. p. 423. 2.3

EDMONDS, J. Optimum branchings. **Journal of Research of the National Bureau of Standards B**, vol. 71, no. 4, p. 233–240, 1967. 1.2, 4.1

FERNANDES, E.; SANTOS, C. dos; MILIDIÚ, R. A machine learning approach to portuguese clause identification. In PARDO, T. et al. (Ed.). **Computational Processing of the Portuguese Language**. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, vol. 6001). p. 55–64. ISBN 978-3-642-12319-1. 3.3.1, 5

FERNANDES, E. L. R. **Entropy Guided Feature Generation for Structure Learning**. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, 2012. 1, 1.5, 4, 1.6, 2.3.3, 2.3.4, 3.3.1, 4.1, 4.4, 5, 5.3, 6.3, 6.3, 6.4, 6.4.1, 6.4.1, 6.6, 7

FERNANDES, E. R.; BREFELD, U. 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: [s.n.], 2011. 4.1

FERNANDES, E. R.; MILIDIÚ, R. L. Entropy-guided feature generation for structured learning of portuguese dependency parsing. In **Computational Processing of the Portuguese Language**. [S.l.]: Springer, 2012. p. 146–156. 4, 2.3.3, 4.1, 4.4

FERNANDES, E. R.; SANTOS, C. N. dos; MILIDIÚ, R. L. Latent trees for coreference resolution. **Computational Linguistics**, MIT Press, 2014. 7

FERREIRA, G. C. D. N. **A Machine Learning Approach for Portuguese Text Chunking**. Master thesis — Pontifícia Universidade Católica do Rio de Janeiro, 2011. 3.3.1

FREITAS, C.; ROCHA, P.; BICK, E. Floresta Sintá(c)tica: Bigger, thicker and easier. In TEIXEIRA, A. et al. (Ed.). **Computational Processing of the Portuguese Language**. [S.l.: s.n.], 2008. (Lecture Notes in Computer Science, vol. 5190), p. 216–219. 6, 2.3.1, 3.1, 6.1

GOLDBERG, Y.; ELHADAD, M. Learning sparser perceptron models. Tech. Rep.[Online]. Available: http://www. cs. bgu. ac. il/˜ yoavg/publications, 2011. 5, 2.3.3, 4.1, 4.5, 6.4.2

GÓMEZ-RODRÍGUEZ, C.; NIVRE, J. Divisible transition systems and multiplanar dependency parsing. **Computational Linguistics**, MIT Press, vol. 39, no. 4, p. 799–845, 2013. 2

GUYON, I. et al. Feature extraction: foundations and applications. Springer, 2008. 1.4

HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. **arXiv preprint arXiv:1207.0580**, 2012. 4.5.1

HUDSON, R. A. **English word grammar**. [S.l.]: Basil Blackwell Oxford, 1990. 2.1

JENATTON, R.; AUDIBERT, J.-Y.; BACH, F. Structured variable selection with sparsity-inducing norms. **The Journal of Machine Learning Research**, JMLR. org, vol. 12, p. 2777–2824, 2011. 4.5

JOHANSSON, R.; NUGUES, P. Dependency-based syntactic-semantic analysis with propbank and nombank. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Twelfth Conference on Computational Natural Language Learning**. [S.l.], 2008. p. 183–187. 2.3.2

KOLLER, D.; TASKAR, B.; GUESTRIN, C. Max-margin markov networks. **Advances in Neural Information Processing Systems (NIPS)**, Citeseer, vol. 17.

KOO, T. et al. Dual decomposition for parsing with non-projective head automata. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing**. [S.l.], 2010. p. 1288–1298. 2.3.3, 2.3.4, 6.3, 6.3, 6.4.1, 6.4.2, 6.4.3

KÜBLER, S.; MCDONALD, R.; NIVRE, J. Dependency parsing. **Synthesis Lectures on Human Language Technologies**, Morgan & Claypool Publishers, vol. 1, no. 1, p. 1–127, 2009. 2.2

LEI, T. et al. Low-rank tensors for scoring dependency structures. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. [S.l.], 2014. 2.3.3

MARTINS, A. F.; ALMEIDA, M. B.; SMITH, N. A. Turning on the turbo: Fast third-order non-projective turbo parsers. 2013. 2.3.4, 6.3, 6.3, 6.4.1, 6.4.2

MARTINS, A. F. et al. Structured sparsity in structured prediction. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Conference on Empirical Methods in Natural Language Processing**. [S.l.], 2011. p. 1500–1511. 2.3.3, 4.5

MARTINS, A. F. et al. Online learning of structured predictors with multiple kernels. 2011. 2.3.3

MARUYAMA, H. Structural disambiguation with constraint propagation. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 28th annual meeting on Association for Computational Linguistics**. [S.l.], 1990. p. 31–38. 2.1

MCALLESTER, D.; HAZAN, T.; KESHET, J. Direct loss minimization for structured prediction. In **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2011. 4.1

MCDONALD, H. Z. R. Enforcing structural diversity in cube-pruned dependency parsing. 2014. 2.3.4

MCDONALD, R.; CRAMMER, K.; PEREIRA, F. Online large-margin training of dependency parsers. In **Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics**. [S.l.: s.n.], 2005. (ACL'05), p. 91–98. 2.3.1, 2.3.3, 2.3.4, 3.2, 3.3, 6.3

MCDONALD, R.; LERMAN, K.; PEREIRA, F. Multilingual dependency analysis with a two-stage discriminative parser. In **In Proceedings of the Conference on Computational Natural Language Learning (CoNLL**. [S.l.: s.n.], 2006. p. 216–220. 2.3.4, 6.3, 6.4.1, 6.4.2

MCDONALD, R.; NIVRE, J. Analyzing and integrating dependency parsers. **Computational Linguistics**, MIT Press, vol. 37, no. 1, p. 197–230, 2011. 2, 2.3, 3.2

MCDONALD, R.; PEREIRA, F. Online learning of approximate dependency parsing algorithms. In **In Proc. of EACL**. [S.l.: s.n.], 2006. p. 81–88. 2.3.3, 2.3.4, 6.3, 6.3

MCDONALD, R.; SATTA, G. On the complexity of non-projective data-driven dependency parsing. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 10th International Conference on Parsing Technologies**. [S.l.], 2007. p. 121–132. 2.3.3

MCDONALD, R. T.; NIVRE, J. Characterizing the errors of data-driven dependency parsing models. In **EMNLP-CoNLL**. [S.l.: s.n.], 2007. p. 122–131. 2, 2.3.1, 2.3.2

MEL'ČUK, I. A. **Dependency syntax: theory and practice**. [S.l.]: SUNY press, 1988. 2.1, 2.2

MILIDIÚ, R. L. et al. Phrase chunking using entropy guided transformation learning. In **ACL**. [S.l.: s.n.], 2008. p. 647–655. 4, 2.3.3, 4.1

MOTTA, E. N. **Indução e seleção incrementais de atributos no aprendizado supervisionado**. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2014. 1, 1.5, 3, 1.6, 2.3.3, 2.3.4, 3, 3.1, 3.1, 3.1.1, 3.1.2, 3.2, 3.3, 3.4, 4.1, 4.5, 4.5.1, 5, 6.3, 6.3, 6.4, 6.4.2, 6.4.2, 6.4.2, 6.6, 7

MURPHY, K. P. Machine learning: a probabilistic perspective. MIT press, 2012. 1.4

NG, A. Y. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In ACM. **Proceedings of the twenty-first international conference on Machine learning**. [S.l.], 2004. p. 78. 1.5

NILSSON, J.; RIEDEL, S.; YURET, D. The conll 2007 shared task on dependency parsing. In SN. **Proceedings of the CoNLL shared task session of EMNLP-CoNLL**. [S.l.], 2007. p. 915–932. 2.3.1, 6.2

NIVRE, J. Dependency grammar and dependency parsing. **MSI report**, vol. 5133, no. 1959, p. 1–32, 2005. 2, 2.1, 2.3

NIVRE, J. Non-projective dependency parsing in expected linear time. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Joint**

Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1. [S.l.], 2009. p. 351–359. 2.3.2

NIVRE, J.; HALL, J.; NILSSON, J. Memory-based dependency parsing. 2008. 2.3.2

NIVRE, J. et al. Labeled pseudo-projective dependency parsing with support vector machines. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Tenth Conference on Computational Natural Language Learning**. [S.l.], 2006. p. 221–225. 2.3.1, 2.3.2

NOVIKOFF, A. B. **On convergence proofs for perceptrons**. [S.l.], 1963. 4.2

PRADHAN, S. et al. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Joint Conference on EMNLP and CoNLL-Shared Task**. [S.l.], 2012. p. 1–40. 7

QUIRK, C.; MENEZES, A.; CHERRY, C. Dependency treelet translation: Syntactically informed phrasal smt. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics**. [S.l.], 2005. p. 271–279. 2.3

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, vol. 65, no. 6, p. 386, 1958. 4.2

SANTOS, C. N. d.; MILIDIú, R. L.; INFORMáTICA, P. U. C. do Rio de Janeiro. Departamento de. **Entropy guided transformation learning**. 2007. 4, 2.3.3, 4.1, 6.3

SANTOS, C. N. dos; MILIDIú, R. L. Entropy guided transformation learning. In **Foundations of Computational Intelligence (1)**. [S.l.]: Springer, 2009. p. 159–184. 4, 2.3.3, 4.4, 5.3, 6.4.1

SGALL, P.; HAJICOVÁ, E.; PANEVOVÁ, J. **The meaning of the sentence in its semantic and pragmatic aspects**. [S.l.]: Springer Science & Business Media, 1986. 2.1

SHEN, D.; KLAKOW, D. Exploring correlation of dependency relation paths for answer extraction. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics**. [S.l.], 2006. p. 889–896. 2.3

SMITH, N. A.; MARTINS, A. F. Linguistic structure prediction with the sparseptron. **XRDS: Crossroads, The ACM Magazine for Students**, ACM, vol. 19, no. 3, p. 44–48, 2013. 2.3.3

STEVENSON, M.; GREENWOOD, M. A. Comparing information extraction pattern models. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Workshop on Information Extraction Beyond The Document**. [S.l.], 2006. p. 12–19. 2.3

TARJAN, R. E. Finding optimum branchings. **Networks**, Wiley Online Library, vol. 7, no. 1, p. 25–35, 1977. 4.1

TAUB-TABIB, H. et al. **Template Kernels for Dependency Parsing**. [S.l.: s.n.], 2014. 2.3.3

TESNIÉRE, L. **Eléments de syntaxe structurale**. [S.l.]: Librairie C. Klincksieck, 1959. 1, 1.1, 2.1

TITOV, I.; HENDERSON, J. A latent variable model for generative dependency parsing. In **Trends in Parsing Technology**. [S.l.]: Springer, 2010. p. 35–55. 2.3.2

TSOCHANTARIDIS, I. et al. Large margin methods for structured and interdependent output variables. **Journal of Machine Learning Research**, vol. 6, p. 1453–1484, 2005. 4.1

XU, P. et al. Using a dependency parser to improve smt for subject-object-verb languages. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics**. [S.l.], 2009. p. 245–253. 2.3

YAMADA, H.; MATSUMOTO, Y. Statistical dependency analysis with support vector machines. In **Proceedings of IWPT**. [S.l.: s.n.], 2003. vol. 3, p. 195–206. 2.3.2

ZHANG, H.; MCDONALD, R. Generalized higher-order dependency parsing with cube pruning. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning**. [S.l.], 2012. p. 320–331. 2.3.4

ZHANG, K.; SU, J.; ZHOU, C. Regularized structured perceptron: A case study on chinese word segmentation, pos tagging and parsing. In **EACL**. [S.l.: s.n.], 2014. p. 164–173. 2.3.3

ZHANG, Y.; CLARK, S. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the Conference on Empirical Methods in Natural Language Processing**. [S.l.], 2008. p. 562–571. 2.3.2

ZHANG, Y. et al. Greed is good if randomized: New inference for dependency parsing. 2014. 2.3.4

ZHANG, Y.; NIVRE, J. Transition-based dependency parsing with rich non-local features. In ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2**. [S.l.], 2011. p. 188–193. 2.3.2

A
Portuguese Corpus Part-Of-Speech Tag Set

The Portuguese dataset of the CoNLL 2006 Shared Task has information about Part-Of-Speech in two levels: coarse-grained and fine-grained. Here we are interested in the first tag set because is more compact and less detailed.

Coarse-grained *postag*	**Grammatical class**
adj	adjective
adv	adverb
art	article
conj	conjunction
in	interjection
n	noun
num	numeral
pron	pronoun
prop	proper noun
prp	preposition
v	verb

Table A.1: Portuguese Part-of-Speech Tags of CoNLL 2006 Shared Task

B
Chunk and Clause Tag Sets

Next, we describe the tags used to mark clauses limits and chunks, which are used as basic attributes in the dependency analysis task.

B.1
Clause Tags

The clause limits is a syntactic information that is available in the BOSQUE corpus. Here, we use parentheses to mark the clause limits. A sentence example from the BOSQUE corpus is shown in Figure B.1.

(Ninguém percebe (que ele quer (impor sua presença)) .)

Figure B.1: A sentence annotated with information of the limits of clauses, indicated by parentheses.

The tags used to mark the clause limits in a sentence are described in Table B.1. The column *Start* encodes the binary attribute that indicate if a clause start in the current *token*. The column *End* indicates if at least a clause ends in the current *token*. Finally, the column *Clause* indicates the clauses of the sentence, using parenthesis, like in the previous example.

Token	***postag***	***Start***	***End***	***Sentence***
Ninguém	pron-indp	S	X	(S*
percebe	v-fin	X	X	*
que	conj-s	S	X	(S*
ele	pron-pers	X	X	*
quer	v-fin	X	X	*
impor	v-inf	S	X	(S*
sua	pron-det	X	X	*
presença	n	X	E	*S)S)
.	.	X	E	*S)

Table B.1: Clause tags

The tag set representing the clause limits of a sentence are used as basic features in dependency parsing task.

B.2
Chunk Tags

In the Table B.2 we describe the tags that encode the chunks, used in dependency parsing task.

Information of the *token* chunk	**Tag**
Beginning of a Nominal chunk	B-NP
Beginning of a Prepositional chunk	B-PP
Beginning of a Verbal chunk	B-VP
Inside of a Nominal chunk	I-NP
Inside of a Prepositional chunk	I-PP
Inside of a Verbal chunk	I-VP
Out of any chunk *chunk*	O

Table B.2: Chunk tags