Assigning Deep Lexical Types Using Structured Classifier Features for Grammatical Dependencies

João Silva
University of Lisbon
Dept. Informatics, Faculty of Sciences
Campo Grande, Lisboa, Portugal
jsilva@di.fc.ul.pt

António Branco
University of Lisbon
Dept. Informatics, Faculty of Sciences
Campo Grande, Lisboa, Portugal
antonio.branco@di.fc.ul.pt

Abstract

Deep linguistic grammars are able to provide rich and highly complex grammatical representations of sentences, capturing, for instance, long-distance dependencies and returning a semantic representation. These grammars lack robustness in the sense that they do not gracefully handle words missing from their lexicon. Several approaches have been explored to handle this problem, many of which consist in pre-annotating the input to the grammar with shallow processing machine-learning tools. Most of these tools, however, use features based on a fixed window of context, such as n-grams. We investigate whether the use of features that encode discrete structures, namely grammatical dependencies, can improve the performance of a machine learning classifier that assigns deep lexical types. In this paper we report on the design and evaluation of this classifier.

1 Introduction

Parsing is one of the fundamental tasks in Natural Language Processing and a critical step in many applications. Many of the most commonly used parsers rely on probabilistic approaches. These parsers are obtained through data-driven approaches, by inferring a probabilistic language model over a dataset of annotated sentences. Though these parsers always produce some analysis of their input sentences, they do not go into deep linguistic analysis.

Deep grammars, also referred to as precision grammars, seek to make explicit information about highly detailed linguistic phenomena and produce complex grammatical representations for their input sentences. For instance, they are able to capture long-distance dependencies and produce the semantic representation of a sentence. Although there is a great variety of parsing methods (see (Mitkov, 2004) for an overview), all CKY-based algorithms require a lexical look-up initialization step that, for each word in the input, returns all its possible categories.

From this it follows that if any of the words in a sentence is not present in the lexicon—an *out-of-vocabulary* (OOV) word—a full parse of that sentence is impossible to obtain. Given that novelty is one of the defining characteristics of natural languages, unknown words will eventually occur. Hence, being able to handle OOV words is of paramount importance if one wishes to use a grammar to analyze unrestricted texts.

Another important issue is that of lexical ambiguity. That is, words that may bear more than one lexical category. The combinatorial explosion of lexical and syntactic ambiguity may hinder parsing due to increased requirements in terms of parsing time and memory usage. Thus, even if there were no OOV words in the input, being able to assign syntactic categories to words prior to parsing may be desirable for efficiency reasons.

For the shallower parsing approaches, such as plain constituency parsing, it suffices to determine the part-of-speech of words, so pre-processing the input with a POS tagger is a common and effective way to tackle either of these problems. However, the linguistic information contained in the lexicon of a

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 62–71,
Jeju, Republic of Korea, 12 July 2012.

deep grammar is much more fine-grained, including, in particular, the subcategorization frame (SCF) of the word, which further constraints what can be taken as a well-formed sentence by imposing several restrictions on co-occurring expressions.

Thus, what for a plain POS tagger corresponds to a single category is often expanded into hundreds of different distinctions, and hence tags, when at the level of detail required by a deep grammar. For instance, the particular grammar we will be using for the study reported in this paper—a grammar following the HPSG framework—has in its current version a lexicon with roughly 160 types for verbs and nearly 200 types for common nouns.

While the deep grammar may proceed with the analysis knowing only the base POS category of a word, it does so at the cost of vastly increased ambiguity[1] which may even allow the grammar to accept ungrammatical sentences as valid. This has lead to research that specifically targets annotating words with a tagset suitable for deep grammars.

Current approaches tend to use shallow features with limited context (e.g. n-grams). However, given that the SCF is one of the most relevant pieces of information that is associated with a word in the lexicon of a deep grammar, one would expect that features describing the inter-word dependencies in a sentence would be highly discriminative and help to accurately assign lexical types. Accordingly, in this paper we investigate the use of structured features that encode grammatical dependencies in a machine-learning classifier and how it compares with state-of-the-art approaches.

Our study targets Portuguese, a Romance language with a rich morphology, in particular in what concerns verb inflection (see for instance, (Mateus et al., 2003) for a detailed account of Portuguese grammar and (Branco et al., 2008) for an assessment of the issues raised by verbal ambiguity).

Paper outline: Section 2 provides an overview of related work, with a focus on supertagging, and introduces tree kernels as a way of handling structured classifier features. Section 3 introduces the particular deep grammar that is used in this work and how it supports the creation of the corpus that provides the data for training and evaluation of the classifier. The classifier itself, and the features it uses, are described in Section 4. Section 5 covers empirical evaluation and comparison with other approaches. Finally, Section 6 concludes with some final remarks.

[1]For instance, a common noun POS tag could be taken as being any of the nearly 200 common noun types existing in the lexicon of the grammar we use in this paper.

2 Background and Related Work

The construction of a hand-crafted lexicon for a deep grammar is a time-consuming task requiring trained linguists. More importantly, such lexica are invariably incomplete since they often do not cover specialized domains and are slow to incorporate new words.

Accordingly, much research in this area has been focused on automatic lexical acquisition (Brent, 1991; Briscoe and Carroll, 1997; Baldwin, 2005). That is, approaches that try to discover all the lexical types a given unknown word may occur with, thus effectively creating a new lexical entry. However, at run-time, it is still up to the grammar using the newly acquired lexical entry to choose which of those lexical types is the correct one for each particular occurrence of that word; and, ultimately, one can only acquire the lexicon entries for those words that are present in the corpus. Thus, any system that is constantly exposed to new text—e.g. parsing text from the Web—will eventually come across some unknown word that has not yet been acquired. Moreover, such words must be dealt with on-the-fly, since it is unlikely that the system can afford to wait until it has accumulated enough occurrences of the unknown word to be able to apply offline lexicon acquisition methods.

In the work reported in the present paper we use a different approach, closer to what is known as supertagging, where we assign on-the-fly a single lexical type to a word.

2.1 Supertagging

POS tagging is a task that relies only on local information (e.g. the word and a small window of context) to achieve a form of syntactic disambiguation. As such, POS tags are commonly assigned prior to parsing as a way of reducing parsing ambiguity by restricting words to a certain syntactic category. Less ambiguity leads to a greatly reduced search space and, as a consequence, faster parsing.

Supertagging, first introduced by Bangalore and Joshi (1994), can be seen as a natural extension of this idea to a richer tagset, in particular to one that includes information on subcategorization frames.

In (Bangalore and Joshi, 1994) supertagging was applied to the Lexicalized Tree Adjoining Grammar (LTAG) formalism. As the name indicates, this is a lexicalized grammar, like HPSG, but in LTAG each lexical item is associated with one or more trees, the elementary structures, which localize information on dependencies, even long-range ones, by requiring that all and only the dependents be present in the structure.

The supertagger in (Bangalore and Joshi, 1994) assigns an elementary structure to each word using a simple trigram model. The data for training was obtained by taking the sentences of length under 15 words in the Wall Street Journal together with some other minor corpora, and parsing them with XTAG, a wide-coverage grammar for English based on LTAG. In addition, and due to data-sparseness, POS tags were used in training instead of words.

Evaluation was performed over 100 held-out sentences from the Wall Street Journal. For a tagset of 365 elementary trees, this supertagger achieved 68% accuracy, which is far too low to be useful for parsing.

In a later experiment, the authors improved the supertagger by smoothing model parameters and adding additional training data (Bangalore and Joshi, 1999). The larger dataset was obtained by extending the corpus from the previous experiment with Penn Treebank parses that were automatically converted to LTAG. The conversion process relied on several heuristics, and though it is not perfect, the authors found that the issues concerning conversion were far outweighed by the benefit of increased training data.

The improved supertagger increased accuracy to 92% (Bangalore and Joshi, 1999). The supertagger can also assign the n-best tags, which increases the chances of it assigning the correct supertag at the cost of leaving more unresolved ambiguity. With 3-best tagging, it achieved 97% accuracy.

A supertagger was also used by Clark and Curran (2007), in their case for a Combinatory Categorial Grammar (CCG). This formalism uses a set of logical combinators to manipulate linguistic construction tough, for our purposes here, it matters only that lexical items receive complex tags that describe the constituents they require to create a well-formed construction.

The set of 409 lexical categories to be assigned was selected by taking those categories that occur at least 10 times in sections 02–21 of a CCG automatic annotation of Penn Treebank (CCGBank).

Evaluation was performed over section 00 of CCGBank, and achieved 92% per word accuracy.

As with the LTAG supertagger, assigning more than one tag can greatly increase accuracy. However, instead of a fixed n-best number of tags—which might be to low, or too high, depending on the case at hand—the CCG supertagger assigns all tags with a likelihood within a factor β of the best tag. A value for β as small as 0.1, which results in an average of 1.4 tags per word, is enough to boost accuracy up to 97%.

Supertagging for HPSG: There has been some work on using supertagging together with the HPSG framework. As with other works on supertagging, it is mostly concerned with restricting the parser search space in order to increase parsing efficiency, and not specifically with the handling of OOV words.

Prins and van Noord (2003) present an HMM-based supertagger for the Dutch Alpino grammar. An interesting feature of their approach is that the supertagger is trained over the output of the parser itself, thus avoiding the need for a hand-annotated dataset.

The supertagger was trained over 2 million sentences of newspaper text parsed by Alpino. A gold standard was created by having Alpino choose the best parse for a set of 600 sentences. The supertagger, when assigning a single tag (from a tagset with 2,392 tags), achieves a token accuracy close to 95%.

It is not clear to what extent these results can be affected by some sort of bias in the disambiguation module of Alpino, given that both the sequence of lexical types in the training dataset and in the gold standard are taken from the best parse produced by Alpino.

Matsuzaki et al. (2007) use a supertagger with the Enju grammar for English. The novelty in their work comes from the use of a context-free grammar (CFG) to filter the tag sequences produced by

the supertagger before running the HPSG parser. In this approach, a CFG approximation of the HPSG is created. The key property of this approximation is that the language it recognizes is a superset of the parsable supertag sequences. Hence, if the CFG is unable to parse a sequence, it can be safely discarded, thus further reducing the amount of sequences the HPSG parser has to deal with.

The provided evaluation is mostly concerned with showing the improvement in parsing speed. Nevertheless, the quality of the supertagging process can be inferred from the accuracy of the parse results, which achieved a labeled precision and recall for predicate-argument relations of 90% and 86%, respectively, over 2,300 sentences with up to 100 words in section 23 of the Penn Treebank.

Dridan (2009) tests two supertaggers, one induced using the TnT tagger (Brants, 2000) and another using the C&C supertagger (Clark and Curran, 2007), over different datasets. For simplicity, we will only refer to the results of TnT over a dataset of 814 sentences of tourism data.

The author experiments with various tag granularities in order to find a balance between tag expressiveness and tag predictability. For instance, assigning only POS—a tagset with only 13 tags—is the easiest task, with 97% accuracy, while a highly granular supertag formed by the lexical type concatenated with any selectional restriction present in the lexical entry increases the number of possible tags to 803, with accuracy dropping to 91%.

2.2 Support-Vector Machines and Tree Kernels

Support-vector machines (SVM) are a well known supervised machine-learning algorithm for linear binary classification. They are part of the family of kernel-based methods where a general purpose learning algorithm is coupled with a problem-specific kernel function (Cristianini and Shawe-Taylor, 2000).

For the work presented in this paper we wish to apply the learning algorithm over discrete tree-like structures that encode grammatical dependencies (see Figure 1 for an example). A suitable kernel for such a task is the tree kernel introduced by Collins and Duffy (2002), which uses a representation that implicitly tracks all subtrees seen in the training data.

This representation starts by implicitly enumerating all subtrees that are found in the training data. A given tree, T, is then represented by a (huge) vector where the n-th position counts the number of occurrences of the n-th subtree in T.

Under this representation, the inner product of two trees gives a measure of their similarity. However, explicitly calculating such an operation is prohibitively expensive due to the high dimensions of the feature space. Fortunately, the inner product can be replaced by a rather simple kernel function that sums over the subtrees that are common to both trees (see (Collins and Duffy, 2002) for a proof).

3 Grammar and Base Dataset

The deep linguistic grammar used in this study is LXGram, a hand-built HPSG grammar for Portuguese (Branco and Costa, 2008; Branco and Costa, 2010).

We used this grammar to support the annotation of a corpus. That is, the grammar is used to provide the set of possible analyses for a sentence (the parse forest). Human annotators then perform manual disambiguation by picking the correct analysis from among all those that form the parse forest.[2] This grammar-supported approach to corpus annotation ensures that the various linguistic annotation layers—morphological, syntactic and semantic—are consistent.

The corpus that was used is composed mostly by a subset of the sentences in CETEMPúblico, a corpus of plain text excerpts from the Público newspaper.

After running LXGram and manually disambiguating the parse forests, we were left with a dataset consisting of 5,422 sentences annotated with all the linguistic information provided by LXGram.

4 Classifier and Feature Extraction

For training and classification we use SVM-light-TK (Moschitti, 2006), an extension to the widely-used SVM-light (Joachims, 1999) software for SVMs that adds a function implementing the tree kernel introduced in Section 2.2. With SVM-light-TK one can

[2] In our setup, two annotators work in a double-blind scheme, where those cases where they disagree are adjudicated by a third annotator. Inter-annotator agreement is 0.86.

directly provide one or more tree structures as features (using the standard parenthesis representation of trees) together with the numeric feature vectors that are already accepted by SVM-light.

Given that the task at stake is a multi-class classification problem but an SVM is a binary classifier, the problem must first be binarized (Galar et al., 2011). For this work we have chosen a one-vs-one binarization scheme, where multiple classifiers are created, each responsible for discriminating between a pair of classes. This divides a problem with n classes into $n(n-1)/2$ separate binary problems (i.e. one classifier for each possible class pairing). Each classifier then performs a binary decision, voting for one of the two classes it is tasked with discriminating, and the class with the overall largest number of votes is chosen.

The dataset, having been produced with the help of a deep grammar, contains a great deal of linguistic information. The first step is thus to extract from each sentence the relevant features in a format that can be used by SVM-light-TK.

Since we are aiming at discriminating between deep lexical types, which, among other information, encode the SCF of a word, the dependency structure associated with a word is expected to be a piece of highly relevant information. We start by extracting the dependency representation of a sentence from the output of LXGram.[3] The dependency representation that is obtained through this process consists of a list of tuples, each relating a pair of words in the sentence through a grammatical relation.

The example in Figure 1 shows the dependency representation of the sentence "a o segundo dia de viagem encontrámos os primeiros golfinhos" (Eng.: by the second day of travel we found the first dolphins).[4] Note that each word is also annotated with its lexical type, POS tag and lemma, though this is not shown in the example for the sake of readability.

For a one-vs-one classifier tasked with discriminating between types A and B we are concerned with finding instances of type A to be taken as positive examples and instances of type B to be taken as negative examples.

Take, for instance, the word "encontrámos" from the example in Figure 1. Its lexical type in this particular occurrence is verb-dir_trans-lex, the type assigned to transitive verbs by LXGram. A one-vs-one classifier tasked with recognizing this type (against some other type) will take this instance as a positive example.

However, the full dependency representation of the sentence has too many irrelevant features for learning how to classify this word. Instead, we focus more closely on the information that is relevant to determining the SCF of the word by looking only at its immediate neighbors in the dependency graph: its dependents and the word it depends on.

This information is encoded in two trees, shown in Figure 2, which are the actual features given to SVM-light-TK.

One tree, labeled with H as root, is used to represent the word and its dependents. The target word is marked by being under an asterisk "category" while the dependents fall under a "category" corresponding to the relation between the target word and the dependent. The words appears as the leafs of the tree, with their POS tags as the pre-terminal nodes.[5]

The second feature tree, labeled with D as root, encodes the target word—again marked with an asterisk—and the word it is dependent on. In the example shown in Figure 2, since the target word is the main verb of the sentence, the feature tree has no other nodes apart from that of the target word.

5 Evaluation

The following evaluation results were obtained following a standard 10-fold cross-validation approach, where the folds were taken from a random shuffle of the sentences in the corpus.

We compare the performance of our tree kernel (TK) approach with two other automatic annotators, TnT (Brants, 2000) and SVMTool (Giménez and Màrquez, 2004).

TnT is a statistical POS tagger, well known for its efficiency—in terms of training and tagging speed—and for achieving state-of-the-art results despite having a quite simple underlying

[3] The details of this process are outside the scope of the current paper and will be reported elsewhere.

[4] Relations in the example: ADV (adverb), C (complement), DO (direct object), PRED (predicate), SP (specifier) and TMP (temporal modifier).

[5] POS tags in the example: V (verb), PREP (preposition) and CN (common noun).

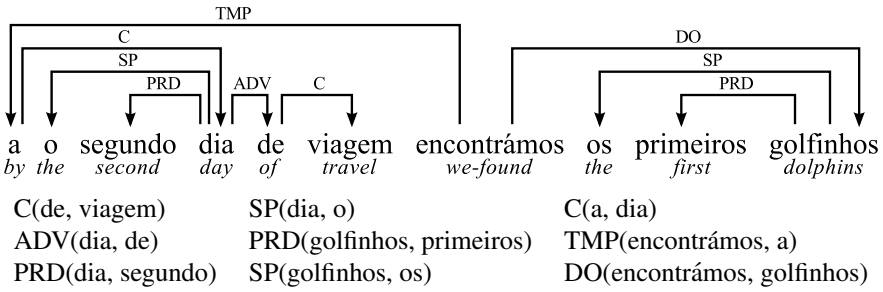

Figure 1: Dependency representation

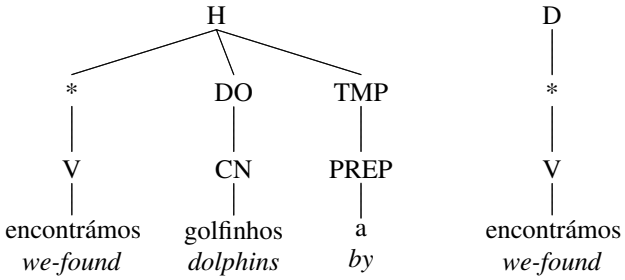

Figure 2: Features for SVM-light-TK

model. It is based on a second-order hidden Markov model extended with linear smoothing of parameters to address data-sparseness issues and suffix analysis for handling unknown words. TnT was used as a supertagger in (Dridan, 2009), where it achieved the best results for this task, and is thus a good representative for this approach to supertagging. We run it out-of-the-box using the default settings.

SVMTool is another statistical sequential tagger which, as the name indicates, is based on support-vector machines. It is extremely flexible in allowing to define which features should be used in the model (e.g. size of word window, number of POS bigrams, etc.) and the tagging strategy (left to right, bidirectional, number of passes, etc). In fact, due to this flexibility, it is described as being a tagger generator. It beat TnT in a POS tagging task (Giménez and Màrquez, 2004), so we use it in the current paper to evaluate whether that lead is kept in a supertagging task. We used the simplest settings, "M0 LR", which uses Model 0 in a left to right tagging direction.[6]

[6]See (Giménez and Màrquez, 2006) for an explanation of these settings.

The type distribution in the dataset is highly skewed. For instance, from the number of common noun types that occur in this corpus, the two most frequent ones are enough to account for 57% of all the common noun tokens. Such skewed category distributions are usually a problematic issue for machine-learning approaches since the number of instances of the more rare categories is too small to properly estimate the parameters of the model.

For many types there are not enough instances in the dataset to train a classifier. Hence, the evaluation that follows is done only for the most frequent types. For instance, top-10 means picking the 10 most frequent types in the corpus, training one-vs-one classifiers for those types, and evaluating only over tokens with one of those types. In addition, we show only the evaluation results of verb types, for which SCF information is more varied and relevant.

Table 1 show the accuracy results for each tool over the top-10, top-20 and top-30 most frequent verb types.

Comparing both sequential supertaggers, one finds that SVMTool is consistently better than TnT, which is in accordance with the results for POS tagging reported in (Giménez and Màrquez, 2004).

Our TK approach beats both supertaggers when

	TnT	SVMTool	TK
top-10	92.98%	94.22%	94.71%
top-20	91.53%	92.39%	90.21%
top-30	91.42%	92.38%	88.70%

Table 1: Accuracy over frequent verb types

dataset	sentences	tokens	unique	oov
base	5,422	51,483	8,815	10.0%
+ Público	10,727	139,330	18,899	7.6%
+ Wiki	15,108	205,585	24,063	6.6%
+ Folha	21,217	288,875	30,204	6.0%

Table 2: Cumulative size of datasets

looking at the top-10 verb types, but falls behind as soon as the number of types under consideration increases. This seems to point towards data-sparseness issues, an hypothesis we test by automatically extending the dataset, as discussed next.

5.1 Experiments with an Extended Dataset

The extended datasets were created by taking additional sentences from the Público newspaper, as well as sentences from the Portuguese Wikipedia and from the Folha de São Paulo newspaper, preprocessing them with a POS tagger, and running them through LXGram.

Such an approach is only made possible because LXGram, like many other modern HPSG grammars, includes a stochastic disambiguation module that automatically chooses the most likely analysis among all those returned in the parse forest, instead of requiring a manual choice by a human annotator (Branco and Costa, 2010). The authors do not provide a complete evaluation of this disambiguation module. Instead, they perform a manual evaluation of a sample of 50 sentences that indicates that this module picks the correct reading in 40% of the cases.

If this ratio is kept, 60% of the sentences in the extended datasets will have an analysis that is, in some way, the wrong analysis, though it is not clear how this translates into errors in the lexical types that end up being assigned to the tokens. For instance, when faced with the rather common case of PP-attachment ambiguity, the disambiguation module may choose the wrong attachment, which will count as being a wrong analysis though most lexical types assigned to the words in the sentence may be correct.

To evaluate this, we tested the disambiguation module over the base dataset, where we know what the correct parses are, and found that the grammar picks the correct parse in 44% of the cases. If we just look at whether the lexical types are correct, the grammar picks a sentence with fully correct types in 68% of the cases.

LXGram displayed a coverage of roughly 30%, and allowed us to build progressively larger datasets as more data was added. The cumulative sizes of the resulting datasets are shown in Table 2. The Table also shows the ratio of OOV words, which was determined by taking the average of the ratio for each of the 10 folds (i.e. words that occur in a fold but not in any of the other 9 folds).

We can now evaluate the tools over the four progressively larger datasets and plot their learning curves. In the following Figures, the errors bars represent a 95% confidence interval.

All learning curves in the following Figures tell a somewhat similar story.

The lead that SVMTool has over TnT when looking only at the base corpus is kept in the extended corpora. Both sequential supertaggers only start to benefit from the increased dataset at the final stage, when sentences from Folha de São Paulo are added. Before that stage the added data seems to be slightly detrimental to them, possibly due to them being sensitive to noise in the automatically generated data.

The learning curves give credence to the hypothesis put forward earlier that our TK approach was being adversely affected by data-sparseness issues when classifying a greater number of verb types, and that it has much to gain by an increase in the amount of training data.

For the top-10 verb types, for which there is enough data in the base dataset, TK starts ahead from the outset and significantly increases its margin over the two supertaggers.

For the top-20 and top-30 verb types, TK starts behind but its accuracy raises quickly as more data are added, ending slightly ahead of SVMTool when running over the largest dataset.

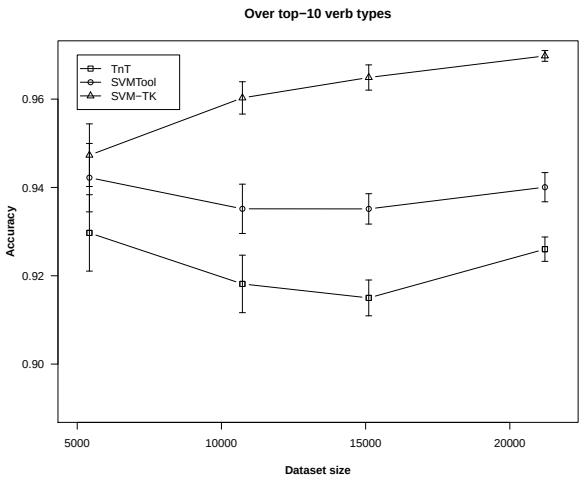

Figure 3: Learning curves (over top-10 verb types)

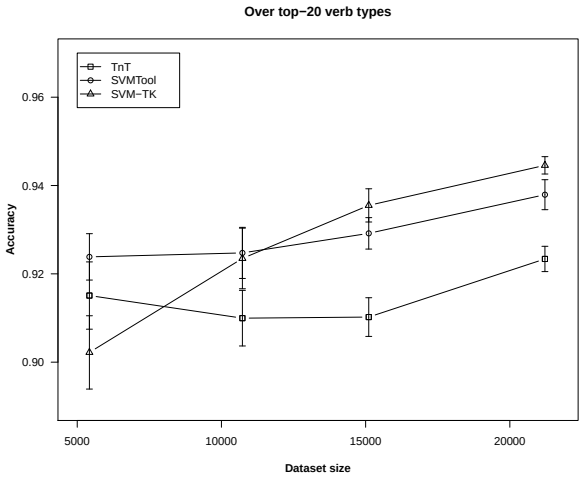

Figure 4: Learning curves (over top-20 verb types)

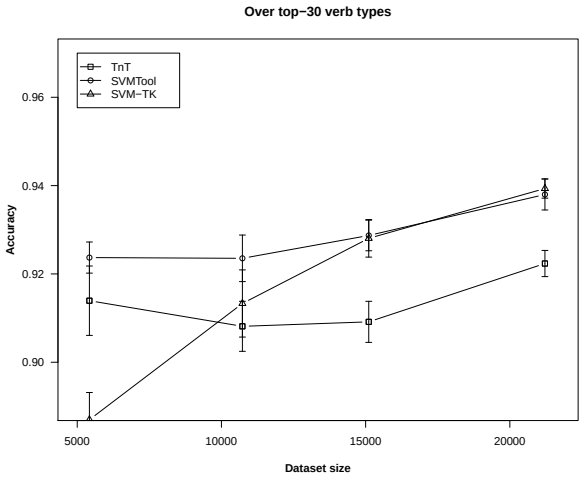

Figure 5: Learning curves (over top-30 verb types)

dataset	accuracy
base	87.24%
+ Público	82.67%
+ Wiki	82.30%
+ Folha	83.92%

Table 3: MaltParser labeled accuracy

5.2 Running over Predicted Dependencies

In the previous section, we were concerned with evaluating the classifier itself. Accordingly, the features used by the classifier were the gold dependencies in the corpus. However, on a running system, the features used by the classifier will be automatically generated by a dependency parser. To evaluate this setup, we used MaltParser (Nivre et al., 2007).

Like the other tools, the parser was run out-of-the-box. The 10-fold average labeled accuracy scores for each dataset shown in Table 3 can thus be seen as a lower bound on the achievable accuracy. Despite this, the performance over the base dataset is extremely good, on par with the best scores achieved for other languages (cf. (Nivre et al., 2007)). However, performance drops sharply when automatically annotated data is used, only beginning to pick up again when running over the largest dataset.

As expected, the noisy features that result from the automatic process have a detrimental effect on the accuracy of the classifier. For the same set of experiments reported previously, the accuracy of the SVM-TK classifier when running over predicted dependencies tends to trail 2.0–2.5% points behind that of the classifier that uses gold dependencies, as shown in Table 4.

6 Concluding Remarks

In this paper we reported on an novel approach to assigning deep lexical types. It uses an SVM classifier with a tree kernel that allows it to seamlessly work with features encoding discrete structures representing the grammatical dependencies between words.

Evaluation over the top-10 most frequent verb types showed that the grammatical dependencies of a word, which can be seen as information on its SCF, are very helpful in allowing the classifier to accurately assign lexical types. Our classifier clearly im-

	top-10		top-20		top-30	
dataset	gold	pred.	gold	pred.	gold	pred.
base	94.71%	93.14%	90.21%	88.66%	88.70%	87.01%
+ Público	96.02%	93.83%	92.34%	90.35%	91.32%	88.97%
+ Wiki	96.48%	93.95%	93.54%	91.29%	92.80%	90.21%
+ Folha	96.98%	94.55%	94.46%	92.26%	93.93%	91.50%

Table 4: SVM-TK classifier accuracy over gold and predicted features

proves over TnT, which had displayed the best supertagging performance in other studies.

When running the classifier for a greater number of verb types, data-sparseness issues led to a drop in performance, which motivated additional experiments where the dataset was extended with automatically annotated data. This allowed us to plot learning curves that show that our approach can maintain a lead in accuracy when given more training data.

Running the classifier over predicted features shows an expected drop in performance. However, we anticipate that using larger corpora will also be effective in raising these scores since additional training data not only improve the classifier, but also the underlying parser that provides the dependencies that are used as features.

References

Timothy Baldwin. 2005. Bootstrapping deep lexical resources: Resources for courses. In Timothy Baldwin, Anna Korhonen, and Aline Villavicencio, editors, *Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition*, pages 67–76.

Srinivas Bangalore and Aravind Joshi. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th Conference on Computational Linguistics (COLING)*, pages 154–160.

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

António Branco and Francisco Costa. 2008. A computational grammar for deep linguistic processing of Portuguese: LX-Gram, version A.4.1. Technical Report DI-FCUL-TR-08-17, University of Lisbon.

António Branco and Francisco Costa. 2010. A deep linguistic processing grammar for Portuguese. In *Proceedings of the 9th Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR)*, LNAI, pages 86–89. Springer.

António Branco, Francisco Costa, and Filipe Nunes. 2008. The processing of verbal inflection ambiguity: Characterization of the problem space. In *Proceedings of the 21st Encontro Anual da Associação Portuguesa de Linguística (APL)*, pages 2577–2583.

Thorsten Brants. 2000. TnT — a statistical part-of-speech tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st North American Chapter of the Association for Computational Linguistics*, pages 224–231.

Michael Brent. 1991. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 209–214.

Ted Briscoe and John Carroll. 1997. Automatic extraction of subcategorization from corpora. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 356–363.

Stephen Clark and James Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33:493–552.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 263–270.

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press.

Rebecca Dridan. 2009. *Using Lexical Statistics to Improve HPSG Parsing*. Ph.D. thesis, University of Saarland.

Mikel Galar, Alberto Fernandéz, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. 2011. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study in one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44:1761–1776.

Jesús Giménez and Lluís Màrquez. 2004. SVMTool: A general POS tagger generator based on support vector

machines. In *Proceedings of the 4th Language Resources and Evaluation Conference (LREC)*.

Jesús Giménez and Lluís Màrquez, 2006. *SVMTool: Technical Manual v1.3*. TALP Research Center, LSI Department, Universitat Politecnica de Catalunya.

Thorsten Joachims. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA.

Maria Helena Mira Mateus, Ana Maria Brito, Inês Duarte, Isabel Hub Faria, Sónia Frota, Gabriela Matos, Fátima Oliveira, Marina Vigário, and Alina Villalva. 2003. *Gramática da Língua Portuguesa*. Caminho, 5th edition.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1671–1676.

Ruslan Mitkov, editor. 2004. *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

Alessando Moschitti. 2006. Making tree kernels practical for natural language learning. In *Proceedings of the 11th European Chapter of the Association for Computational Linguistics*.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Robbert Prins and Gertjan van Noord. 2003. Reinforcing parser preferences through tagging. *Traitment Automatique des Langues*, 44:121–139.