Automatically Acquiring a Language Model for POS Tagging Using Decision Trees

Lluís Márquez and Horacio Rodríguez
LSI – UPC
c/ Jordi Girona 1-3
Barcelona 08034, Catalonia
{lmarquez,horacio}@lsi.upc.es

Abstract

We present an algorithm that automatically acquires a statistically-based language model for POS tagging, using statistical decision trees. The learning algorithm deals with more complex contextual information than simple collections of n-grams and it is able to use information of different nature. The acquired models are independent enough to be easily incorporated, as a statistical core of constraints/rules, in any flexible tagger. They are also complete enough to be directly used as sets of POS disambiguation rules. We have implemented a simple and fast tagger that has been tested and evaluated on the WSJ corpus with a remarkable accuracy. Comparative results are reported.

1 Introduction

In NLP, it is necessary to model the language in a representation suitable for the task to be performed. The language models more commonly used are based on two main approaches: first, the linguistic approach, in which the model is written by a linguist, generally in the form of rules or constraints (Voutilainen & Järvinen 95). Second, the automatic approach, in which the model is automatically obtained from corpora (either raw or annotated), and consists of n-grams (Garside et al. 87; Cutting et al. 92), rules (Hindle 89) or neural nets (Schmid 94). In the automatic approach we can distinguish two main trends: The low-level data trend collects statistics from the training corpora in the form of n-grams, probabilities, weights, etc. The high level data trend acquires more sophisticated information, such as context rules, constraints, or decision trees (Daelemans et al. 96; Márquez & Rodríguez 97; Samuelsson et al. 96). The acquisition methods range from supervised-inductive-learning-from-examples algorithms (Quinlan 93; Aha et al. 91) to genetic algorithm strategies (Losee 94), through the transformation-based error-driven algorithm used in (Brill 95). Still another possibility are the hybrid models, which try to join the advantages of both approaches (Voutilainen & Padró 97).

We present in this paper a supervised learning algorithm that induces, from annotated training corpora, a language model oriented to POS tagging. We describe the main characteristics of the algorithm, its sources of information and its basic future extensions.

The acquired language model, consisting basically of the distribution of tags and words in some relevant contexts, is represented as a set of statistical decision trees. Such trees can be straightforwardly interpreted as a set of POS disambiguation rules and therefore it can be used as a tagger by itself. Following this idea we have implemented a quite simple, fast and accurate tagger. We describe the basic tagging algorithm and some variants for further extensions. Experiments and results obtained on the Wall Street Journal (WSJ) corpus are also reported.

In addition, these decision trees can be translated into weighted rules or constraints expressing compatibility between tags and contexts, so it becomes easy to incorporate this model in any rule-based or hybrid flexible tagger such as (Padró 96). Some experiments are reported in this direction.

The paper is organized as follows. In section 2 we describe the language model acquisition algorithm, in section 3 we expose the tagging algorithm, and in sections 4 and 5 we describe the corpus used, the experiments performed and the results obtained. Conclusions and an overview of the future work can be found in sections 6 and 7.

2 Language Model Acquisition

Choosing, from a set of possible tags, the proper syntactic tag for a word in a particular context...
can be seen as a problem of classification. Decision trees, recently used in NLP basic tasks such as tagging and parsing (McCarthy & Lehnert 95; Daelemans et al. 96; Magerman 96), are suitable for performing this task.

A decision tree is a n-ary branching tree that represents a classification rule for classifying the objects of a certain domain into a set of mutually exclusive classes. The domain objects are described as a set of attribute-value pairs, where each attribute measures a relevant feature of an object taking a set of discrete, mutually incompatible values. Each non-terminal node of a decision tree represents a question on (usually) one attribute. For each possible value of this attribute there is a branch to follow. Leaf nodes represent concrete classes.

Classify a new object with a decision tree is simply following the convenient path through the tree until a leaf is reached.

Statistical decision trees only differ from common decision trees in that leaf nodes define a conditional probability distribution on the set of possible classes.

In our case we have a classification problem for each class of POS ambiguity, to which we will associate one statistical decision tree. The set of all these trees is what we call the language model.

Acquisition Algorithm

The algorithm we used for constructing the statistical decision trees is a non-incremental supervised learning-from-examples algorithm of the TDIDT (Top Down Induction of Decision Trees) family. It constructs the trees in a top-down way, guided by the distributional information of the examples (Quinlan 93). Briefly, the algorithm works as a recursive process that departs from considering the whole set of examples at the root level and constructs the tree in a top-down way branching at any non-terminal node according to a certain selected attribute. The different values of this attribute induce a partition of the set of examples in the corresponding subsets, in which the process is applied recursively in order to generate the different subtrees. The recursion ends, in a certain

...
whole set of ambiguity classes as a taxonomy with a DAG structure (see figure 1).

![Figure 1: A partial view of the ambiguity-class taxonomy](image)

The smoothing is done by considering a successive generalization of the examples included in the training set. The shadowed part in figure 1 corresponds to the generalization of JJ-NH-RB, a class with very few examples. It is possible to increase this set of examples in two stages by adding (with a proper weighting) firstly the examples of the four depth-2 classes: JJ-NH-RB-VB, IN-JJ-NH-RB, JJ-NH-NP-RB and JJ-NH-RB-UH; and secondly the examples of the two related classes at depth three: JJ-NH-RB-RP-VB and DT-JJ-NH-PDT-RB.

Experimental tests show that this method reports benefits in some cases.

**Attribute Selection Function**

We have tested several attribute selection functions of both families, but no significant differences could be observed between them. For the experiments reported in section 5 we used a attribute selection function due to López de Mántaras (López 91), belonging to the information-based family, which showed a slightly higher stability than the others. Roughly speaking, it defines a distance measure between partitions and selects for branching the attribute that generates the closest partition to the **correct partition**, namely the one that joins together all the examples of the same class.

We plan to improve the basic technique of attribute selection by means of reducing its locality. This could be done, at each step of branching, exploring deeper levels of resulting trees in order to evaluate the goodness of each attribute. However, it would lead to a severe increasing of the computational cost. Further experiments should be done for testing the real improving vs. the resulting overhead.

**Branching Strategy**

Usual TDIDT algorithms consider a branch for each value of the selected attribute. This strategy is not feasible when the number of values is big (or even infinite). In our case the greatest number of values for an attribute is 45—the tag set size—which is considerably big (this means that the branching factor could be 45 at every level of the tree). Some systems perform a previous recasting of the attributes in order to have binary-valued attributes and to deal only with binary trees (Magerman 96). This can always be done but the resulting features lose their intuition and direct interpretation, and explode in number. We have chosen a mixed approach which consists of splitting for all values and afterwards joining the resulting subsets into groups for which we have not enough statistical evidence of being different distributions. This statistical evidence is tested with a χ² test at a 5% level of significance, with a previous smoothing of data in order to avoid zero probabilities.

Experimental tests show that in this way more compact and predictive trees are obtained.

**Pruning the Tree**

Decision trees that correctly classify all examples of the training set are not always the most predictive ones. This is due to the phenomenon known as **over-fitting**. It occurs when the training set has a certain amount of misclassified examples, which is obviously the case of our training corpus (see section 4). If we force the learning algorithm to completely classify the examples then the resulting trees would fit also the noisy examples.

The usual solutions to this problem are: 1) Prune the tree, either during the construction process (Quinlan 93) or afterwards (Mingers 89b); 2) Smooth the conditional probability distributions using fresh corpus (Magerman 96).

Since another desirable goal is to have small trees we have implemented a pruning technique. In a first step the tree is completely expanded and afterwards is pruned following a minimal cost-complexity criterion (Breiman et al. 84). Roughly speaking this is a process that iteratively cut those subtrees producing only marginal benefits in accuracy, obtaining smaller trees at each

6In real cases the branching factor is much lower since not all tags appear always in all positions of the context.

7Of course, this can be done only in the case of statistical decision trees.
step. The trees of this sequence are tested using a comparatively small fresh part of the training set in order to predict which is the best tree on new examples. For that we simply choose the most accurate tree among all the trees with less nodes than a prefixed parameter.

Experimental tests have shown that the pruning process reduces tree sizes at about 50% and improves their accuracy in a 2–5%.

An Example

Finally, we present a real example of a decision tree branch learned for the class **IN-RB**.

![Figure 2: Example of a decision tree branch](image-url)

We can observe in figure 2 that each node in the path from the root to the leaf contains a question on a concrete attribute and a probability distribution. In the root it is the prior probability distribution of the class. In the other nodes it represents the probability distribution conditioned to the answers of the questions preceding the node. For example the second node says that the word as is more commonly a preposition than an adverb, but the leaf says that the word as is almost sure an adverb when it occurs immediately before another adverb and a preposition (this is the case of as much as, as well as, as soon as, etc.)

3 Tagging Algorithm

Algorithm

We have programmed a *reductionistic* tagger in the sense of constraint grammars (Voutilainen & Järvinen 95). In a initial step a word-form frequency dictionary constructed from the training corpus provides each input word with all possible tags with their associated lexical probability. After that an iterative process reduces the ambiguity (discarding low probable tags) at each step until a certain stopping criterion is satisfied. The whole process is represented in figure 3.

![Figure 3: The tagging process](image-url)

More particularly, at each step and for each ambiguous word the work to be done is:

1) Classify the word using the corresponding decision tree. 2) Use the resulting probability distribution to update the probability distribution of the word. 3) Discard the tags with *almost* zero probability, that is, those with probabilities lower than a certain *discard boundary* parameter.

After the stopping criterion is satisfied some words could still remain ambiguous. Then there are two possibilities: 1) Choose the most-likely tag for each still ambiguous word to complete disambiguate the text. 2) Accept the residual ambiguity (perhaps for treating it in successive stages).

Note that a unique iteration forcing the complete disambiguation is equivalent to use directly the trees as classifiers and results in a very efficient tagger, while performing several steps reduces progressively the efficiency but takes advantage of the statistical nature of the trees.

There are some decisions to take for each ambiguous word after each iteration of the algorithm.

- Firstly, how to update the current probability distribution with the new information resulting from the tree classification. We tried different options that included the sum, the product and the simply substitution for the new distribution. First experiments suggest that the best combination is the product.

- Secondly, when a certain tag is discarded due to its low probability, the ambiguity of the word is reduced and it may fall into another ambiguity class. So in the next iteration it
is possible to change the initial assigned tree for the tree representing the new more specific class. This seems an interesting idea, however we have not obtained good results up to the present.

Another important point is to determine an appropriate stopping criterion. First experiments seem to indicate than the performance increases up to a unique maximum and then softly decreases as the number of iterations increases (see results in section 5). If this was the general behaviour then the criterion will be easy and robust, but future research is needed to extract reliable conclusions.

Direction of tagging

We are very used to see taggers that disambiguate from left to right the input text. There is, in principle, no reason/limitation for following this criterion in our case. In fact any order could be considered. The most general approach is the one that defines a global scoring for each word in the input text and then proceeds, island-driven, from the most relevant words to the less relevant. We explored some simple scorings based on the prior error rate of the trees associated to the ambiguous words, but in the initial results we observed no significant differences between them. This led us to try to establish an upper bound for the achievable accuracy with the best scoring. For that we tested the tagger simulating ideal conditions, that is, when a word is about to be disambiguated its context is supposed to be completely ambiguous and correctly tagged. The accuracy rate raised from 97.29% to 97.37%, that is, less than 0.1 percent. The conclusion was that there is not much to win searching a good scoring criterion. So we returned to the simpler left-to-right tagging.

4 Description of the corpus

We used the Wall Street Journal corpus to train and test the system. We divided it in three parts: 1,100 Kw were used as a training set, 20 Kw as a model-tuning set and 50 Kw as a fresh test set.

The tag set size is 45 tags. 36.4% of the words in the corpus are ambiguous, and the ambiguity ratio is 2.44 tags/word over the ambiguous words, 1.52 overall.

We used a lexicon derived from training corpora, that contains all possible tags for a word, as well as their lexical probabilities. For the words in test corpora not appearing in the train set, we stored all possible tags, but no lexical probability (i.e. we assume uniform distribution).

The noise in the lexicon was filtered by manually checking the lexicon entries for the most frequent 200 words in the corpus to eliminate the tags due to errors in the training set. For instance the original lexicon entry (numbers indicate frequencies in the training corpus) for the very common word *the* was

```
the CD 1 DT 47715 JJ 7 NN 1 NNP 6 VBP 1
```

since it appears in the corpus with the six different tags: CD (cardinal), DT (determiner), JJ (adjective), NN (noun), NNP (proper noun) and VBP (verb-personal form). It is obvious that the only correct reading for *the* is determiner.

5 Experiments and results

The whole WSJ corpus contains 243 different ambiguity classes. From them, the acquisition algorithm learned a base of 194 trees covering 99.5% of the ambiguous words and requiring about 500 Kb of storage. For each tree, a prior estimation of its error rate was extracted directly from how well it fit the training/pruning corpus.

We implemented using PERL 5.0 a first TreeTagger prototype, with a tagging speed of about 300 words/sec running in a SUN ULTRA2 machine. We tested the tagger on the 50 Kw test set (non-covered ambiguous words were tagged with their most-likely tags) with the following results.

The performance of some of the learned trees is shown in table 1. The reported trees correspond to the ten most difficult classes which concentrate the 62.5% of the errors committed by a most-likely tagger (ML column). TT column shows the number of errors committed by our tree-tagger. The two remaining columns contain respectively the percentage of errors committed by each tree and its prior error rate.

On the one hand we can observe a remarkable reduction in the number of errors (56.4%). On the other hand we also observe that some trees do not obtain a great reduction from the most-likely baseline tagger. It is the case of the classes:

9That is, we assumed a morphological analyzer that provides all possible tags for unknown words.

10The 200 most frequent words in the corpus represent over half of it.
Comparative results are reported in table 3. IGTTree (Daelemans et al. 96) stands for a Memory-Based POS tagger which uses a tree representation of the training set of examples. HMM stands for a bigram-based HMM statistical tagger (Elworthy 92), while RL stands for a relaxation labeling-based tagger (Padró 96) using trigram information. HMM and RL taggers have been tested under the same conditions as our tagger. IGTTree results, reported in (Daelemans et al. 96), come from testing on 70 Kw of the WSJ corpus.

<table>
<thead>
<tr>
<th></th>
<th>ML</th>
<th>IT</th>
<th>tree-err</th>
<th>prior</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBD-VBN</td>
<td>261</td>
<td>94</td>
<td>0.47%</td>
<td>7.53%</td>
</tr>
<tr>
<td>NN-VB-VBP</td>
<td>258</td>
<td>54</td>
<td>4.02%</td>
<td>3.32%</td>
</tr>
<tr>
<td>VB-VBP</td>
<td>228</td>
<td>46</td>
<td>4.97%</td>
<td>3.06%</td>
</tr>
<tr>
<td>IN-RB-RP</td>
<td>210</td>
<td>164</td>
<td>9.14%</td>
<td>7.13%</td>
</tr>
<tr>
<td>DT-IN-RB-WDT</td>
<td>187</td>
<td>50</td>
<td>12.06%</td>
<td>6.06%</td>
</tr>
<tr>
<td>JJ-VBD-VBN</td>
<td>180</td>
<td>95</td>
<td>16.50%</td>
<td>18.75%</td>
</tr>
<tr>
<td>JJ-JJN</td>
<td>144</td>
<td>122</td>
<td>14.04%</td>
<td>16.36%</td>
</tr>
<tr>
<td>NN-VBG</td>
<td>116</td>
<td>58</td>
<td>40.81%</td>
<td>16.54%</td>
</tr>
<tr>
<td>NNS-VBD</td>
<td>81</td>
<td>44</td>
<td>6.39%</td>
<td>4.37%</td>
</tr>
<tr>
<td>JJ-JJ</td>
<td>73</td>
<td>48</td>
<td>10.83%</td>
<td>11.20%</td>
</tr>
<tr>
<td>NN-VB</td>
<td>67</td>
<td>12</td>
<td>1.63%</td>
<td>1.11%</td>
</tr>
<tr>
<td>Total</td>
<td>1806</td>
<td>787</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Number and percentages of error for the most difficult classes

IN-RB-RP, JJ-JN and JJ-RB. A further study of these particular classes is required here.

It is also noticeable the clear correlation between prior and real error rate, showing that the prior estimation is quite reliable.

Global results of the tagger are reported in table 2. It includes accuracy rates over the ambiguous words and overall, for a number of iterations ranging from 0 to 10 (note that 0 iterations is equivalent to a most-likely tagger). The discard boundary was set to 5%. Results correspond to a complete disambiguation.

<table>
<thead>
<tr>
<th>num-it.</th>
<th>ambiguous</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>83.59%</td>
<td>94.44%</td>
</tr>
<tr>
<td>1</td>
<td>91.65%</td>
<td>97.18%</td>
</tr>
<tr>
<td>2</td>
<td>91.98%</td>
<td>97.29%</td>
</tr>
<tr>
<td>3</td>
<td>91.86%</td>
<td>97.25%</td>
</tr>
<tr>
<td>4</td>
<td>91.84%</td>
<td>97.24%</td>
</tr>
<tr>
<td>5</td>
<td>91.71%</td>
<td>97.20%</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>91.64%</td>
<td>97.17%</td>
</tr>
</tbody>
</table>

Table 2: Results of our tagger

Figures in this table show that the maximum performance is achieved with two iterations and that the following decreasing is very soft. The residual ambiguity at this point is 1.026 tags/word (1.08 tags/word over the ambiguous). This means on the one hand that 2.75% of the words are not completely disambiguated, but on the other hand that 98.22% of the words preserve the correct tag.\(^{11}\)

\(^{11}\)This figure is significantly better than the one obtained in (Samuelsson & Voutilainen 97) when comparing a stochastic tagger with the linguistic tagger of the Constraint Grammar framework.

We could add to the comparative results the performance of Brill's tagger (Brill 92) also on the WSJ corpus, reporting a 3-4% error rate.

The main conclusion from above results is that our tagger performs better, in similar conditions, than a variety of the non linguistically motivated state-of-the-art taggers.

Although the improvement obtained might seem small, it must be taken into account that we are moving very close to the best achievable result with these techniques. First, some ambiguities, such as Noun–Adjective, can only be solved using semantic information. Second, the WSJ corpus contains noise (mistagged words) that affects both the training and the test sets. The noise in the training set produces noisy –and so less precise– models. In the test set, it produces a wrong estimation of accuracy, since correct answers are computed as wrong and vice-versa. See (Márquez & Padró 97) for a more detailed discussion.

Yet another test of the appropriateness of the tree model was done in a previous experiment with another tagger. The group of the 44 most representative trees (covering 84% of the examples) were translated into a set of context con-
straints and used to feed a relaxation–labelling–based tagger. See (Márquez & Padró 97) for details. Reported results, 97.09% accuracy when using alone and 97.39% when combining with a trigram model, showed that the addition of automatically acquired context constraints led to an improvement in the accuracy of the tagger.

6 Conclusions

We have presented an algorithm for automatically learning a language model for POS tagging based on statistical decision trees. It uses more complex contextual information than usual n-gram models and it can easily accept other kinds of information.

We have used this model for developing a fast and simple tagger tested on the WSJ corpus with a remarkable accuracy. Although it is difficult to compare the results to other works, since the accuracy varies greatly depending on the corpus, the tag set, and the lexicon or morphological analyzer used, it seems that our tagger achieves (and slightly overcomes) state-of-the-art performance in tagging domain. Some comparative results have been reported to support this conclusion.

Finally we have shown the independence of the acquired language model from the particular tagging algorithm, by translating the trees into a set of context constraints to feed a flexible relaxation–labelling–based tagger. Results obtained with this tagger are fairly good.

7 Further Work

Further work is still to be done in the following directions:

On the language model learning algorithm:

- Take into account morphological, semantic and other kinds of information.
- Consider more complex context features, such as non–limited distance or barrier rules in the style of (Samuelsson et al. 96).
- Use a more informed attribute selection function by exploring deeper levels during the tree construction.
- A study of the most difficult classes for the trees in order to find particular solutions.

On the tagging algorithm: the testing phase of our tagger is still at a preliminary stage. So several questions (e.g. the stopping criterion, combination of different trees, probability updating functions, parameters setting, etc.) must be deeply studied in order to obtain more reliable conclusions.

We also plan to apply soon our tagger to other languages, as Spanish and Catalan, and use the same techniques to acquire language models for word sense disambiguation, testing them by themselves and in cooperation with other algorithms (e.g. performing POS tagging and WSD jointly).

8 Acknowledgments

This research has been partially funded by the Spanish Research Department (CICYT’s ITEM project TIC96–1243–C03–02), by the EU Commission (EuroWordNet LE4003) and by the Catalan Research Department (CIRIT’s quality research group 1995SGR 00566).

Appendix A

| CC | Coordinating conjunction |
| CD | Cardinal number |
| DT | Determiner |
| IN | Preposition |
| JJ | Adjective |
| NN | Noun, singular |
| NNP | Proper noun, singular |
| NNS | Noun, plural |
| NNPS | Proper noun, plural |
| PDT | Predeterminer |
| RB | Adverb |
| RP | Particle |
| UH | Interjection |
| VB | Verb, base form |
| VBD | Verb, past tense |
| VBG | Verb, gerund |
| VBN | Verb, past participle |
| VBP | Verb, non-3rd ps. sing. present |
| VBZ | Verb, 3rd ps. sing. present |
| WDT | wh-determiner |

Table 4: Tag meanings

---

13Overcoming the bi/tri-gram models and properly cooperating with them and with a small set of linguistically motivated hand-written constraints.
References


