Doubling DOP*

Data Oriented Parsing based on Double-DOP and DOP*

Benno Kruit
10576223

Sara Veldhoen
10545298

Abstract

This paper investigates two models of the Data Oriented Parsing approach to natural language syntax. We assess the theoretical and practical differences between these models by comparing the grammars they derive.

1 Introduction

A common approach to natural language syntax, is to view the structure of sentences as constituent trees. Such trees can be described by a *Context Free Grammars* (CFGs), such that all trees are built up from rules that each describe the production (children nodes) of a single node (parent) in the tree. When building an empirical model of observed parse trees, these rules are extended with probabilities to form a *probabilistic CFG* (PCFG). This gives the trees that are 'generated' by these rules their own probability, which makes it a statistical model of a distribution over natural language syntax.

The simple rules of a CFG cannot describe all linguistic phenomena, such as long distance dependencies. Grammars can be enriched by Markovisation, to include deeper levels in the tree.

1.1 DOP

Data-Oriented Parsing (DOP), as first introduced in(**?**), takes a different approach. It models the language with a Probabilistic Tree Substitution Grammar (PTSG). The trees in the treebank are taken apart, which results in *fragments* of arbitrary depth. A fragment is a connected subgraph of a tree such that it corresponds to context-free productions in that tree, i.e. each node must have either have children with the same labels as in the original tree, or no children at all. Note that a level-one fragment corresponds to a CFG rule. Its *symbolic grammar* refers to the set of fragments (that receive a non-zero weight) in a grammar.

Fragments can be combined in a *derivation* to build syntactic structures. A step in a derivation is a composition, denoted by the symbol $\circ$. We follow the convention to only allow left-most derivations. This means that the left-most non-terminal node in a fragment f_1 must correspond to the root node of f_2 in order to derive $f_3 = f_1 \circ f_2$

For each fragment, the probability is estimated by counting how often it occurs in the treebank, compared to others with the same root. The probability of a derivation is the product of the probability of the fragments. Note that a single tree can be the result of different derivations. Therefore probability of a tree is the sum of the probabilities of all its derivations.

1.2 Theoretical issues

It has been argued that DOP (in its original formulation) is biased and inconsistent (**?**), both assumed to be bad properties of an estimator in general. As we will see, bias is not necessarily a bad thing. In fact, (**?**) proves that any non-overfitting estimator is biased. Furthermore, he shows that it is possible to define a DOP-estimator that is consistent.

1.3 Practical issues

In its original formulation, DOP takes the trees apart in all possible ways. The number of fragments is exponential in the length of the sentences, thus the size of the symbolic grammar would be far too huge to be computationally feasible. Different approaches have been taken to reduce the symbolic grammar, e.g. by sampling or by applying a smart algorithm. This appears to be far from trivial.

1.4 Outlook

Section 2 elaborates the notions of consistency and bias and their relation to overfitting. In section 3, we outline two approaches that tackle the reduction of the symbolic grammars: Double-DOP and DOP*. This report focuses on a comparison of these approaches. Theoretically, they differ in that DOP*, unlike Double-DOP, has been proven to be consistent (**?**). We investigate the differences between the grammars produced by Double-DOP DOP*. The algorithms can be decomposed into two parts. We also analyze the impact of the partial choices by mutually using these parts.

Section 4 offers a detailed comparison of the two methods as well as a description of the experiments we conduct. In section 5, we present our findings and provide an analysis.

2 Statistics: Consistency and Bias

Linguistic studies of syntax mostly concern *competence* models, which describe the which structures appear in a language. In contrast, a *performance* model of language is an estimate of the probability of observing a parse tree in language use. It treats language as a statistical distribution over syntactic structures.

Let Ω be the set of all possible parse trees. The distribution P_Ω then describes the language, where $P_\Omega(t)$ is the probability of observing a tree $t \in \Omega$. Using a sample of parse trees from the language, an *estimator* EST builds a statistical model. A parser then uses that statistical model to predict the correct parse tree of sentences. A sample $X \in \Omega^n$ from the language is called a *corpus* or *treebank* of size n, which makes $\textsc{est}(X)$ an estimator trained on a sample. If $\mathcal{M}$ is the set of probability distributions over Ω, then $P_\Omega \in \mathcal{M}$ and $\textsc{est}(X) \in \mathcal{M}$.

In theory, an estimator should make exactly the right estimations of probabilities if it's given an infinite amount of data. That is to say, it should *converge* to the true distribution. If an estimator converges in the limit, that estimator is *consistent*. However, given a finite amount of data, the estimator will probably not generate the correct distribution. The distance between the true distribution P^* and an estimate P is called the *loss* of that estimate. The loss can be defined in different ways, but the most popular is the *mean squared difference*:

$$\mathcal{L}(P, P^*) = \sum_{t \in \Omega} P^*(t)(P^*(t) - P(t))^2$$

From a true distribution, it's possible to calculate the expected loss of an estimator trained on a treebank of a certain size. This is the *risk* or *error* of that estimator given a sample size and a distribution. With these definitions, it is possible to define estimator consistency when sampling $X \in \Omega^n$ from P_Ω:

$$\lim_{n \to \infty} \mathbf{E}[\mathcal{L}(\textsc{est}(X), P_\Omega)] = 0$$

In its original formulation, DOP was defined using a *relative frequency estimate*.

Bias is good.

3 Existing Models: Double-DOP and DOP*

In this section, we outline two approaches to constrain the extraction of fragments: Double-DOP and DOP*. Furthermore, we discuss the similarities and dissimilarities for these two approaches.

3.1 Double-DOP

In the following, we discuss Double-DOP as it was presented in (**?**). In this model, no unique fragments are extracted from the dataset: if a construction occurs in one tree only, it is probably not representative for the language. This is carried out by a dynamic algorithm using tree-kernels. It iterates over pairs of trees in the treebank, looking for fragments they have in common. In fact, only the largest shared fragment is stored.

The symbolic grammar that is the output of this algorithmis not guaranteed to derive each tree in the training corpus. Therefore all one-level fragments, consitutuing the set of PCFG-productions, are also added.

The Double-DOP model describes an extraction method for determining the symbolic grammar. However, it was also implemented with different estimators. The estimation is done in a second pass over the treebank, gathering frequency counts for the fragments in the gramar.

The best maximizing objective appears to be MCP (max consituent parse), which maximizes a weighted average of expected labeled recall and precision. The latter cannot be maximized directly, but is approximated by minimizing the mistake rate. Parameter λ that rules the linear interpolation was empirically found to be optimal at 1.15. Efficient calculation of MPC is possible with dynamic programming, but it can also be approximated with a list of k-best derivations.

In addition, empirical results show that the relative frequency estimate outperforms the other estimates tested, i.e. Equal Weights Estimate (first adjust counts proportional to the size of the symbolic grammar, this value proportional to fragments with the same root), and Maximum Likelihood (optimizing to maximum likelihood for the training data with an Inside-Outside algorithm).

3.2 DOP*

In DOP* (**?**), a rather different approach is taken called held-out estimation. The treebank is split in two parts, the extraction corpus (EC) and a held-out corpus (HC). An initial set of fragments is extracted from the EC, containing all the fragments from its trees. The weights are then determined so as to to maximize the likelihood of HC, under the assumption that this is equivalent to maximizing the joint probability of the *shortest derivations* of the trees in HC. All fragments that do not occur in such a derivation are removed from the symbolic grammar. Note that some trees in HC may not be derivable at all. Furthermore, a tree could have several shortest derivations.

Consistency and bias DOP* was claimed to be the first consistent (non-trivial) DOP-estimator, (**?**) provides a consistency proof. On the other hand DOP* is biased, but Zollmann shows how bias actually arises from generalization: no non-overfitting DOP estimator could be unbiased. Bias is therefore not prohibited but on the contrary a desirable property of an estimator.

In (**?**) it is argued that there is a problem with the consistency proof given for DOP*, as well as the non-consistency proof for other DOP-estimators by (**?**). Zuidema points out that these proofs use a frequency-distribution test, whereas for DOP a weight-distribution test would be more appropriate.

4 Comparison

DOP* and Double-DOP differ both in the set of fragments they extract and their estimation of the weights. To investigate the exact differences, we will view both steps separately.

Note that the DOP* extraction needs another decision: in many cases, there are several shortest derivations possible. From now on, we add all fragments that occur in one of these shortest derivations to the symbolic grammar. Of course we need to adjust the weights (e.g. divide the frequency counts by the number of shortest derivations) so that no full tree gets a higher impact on the PTSG. We will keep to the original formulation of DOP* in case no derivation is possible, i.e. not including any fragments for this tree.

Extraction Double-DOP uses tree kernels to find the maximal overlapping fragments of pairs of trees, which are added to the symbolic grammar. We will call this the *maximal-overlap* method. DOP* iteratively finds the shortest derivation of one tree given all the fragments of a set of trees, herafter the *shortest-derivation* method.

It is easy to see that the DOP* extraction method does not depend on the corpus split: we can also try to find the shortest possble derivation using fragments from all the other trees. Likewise Double-DOP could be implemented using a split, comparing pairs that consist of a tree from each part of the corpus.

We will implement both extraction methods in a 1 vs the rest manner. In this way, we can analyse how the resulting symbolic grammars differ. The analysis will comprise the size of the resulting symbolic grammar and the relative number of fragments of certain depth. Furthermore, we might be able to find interesting patterns by manually looking at the fragments that were extracted by one of the systems only.

Estimation Double-DOP determines the weights of the fragments in the symbolic grammar in a separate run over the treebank, to obtain exact counts. We use the relative frequency estimate to assign weights to the fragments. DOP* on the other hand counts the occurrence in shortest derivations of the fragments, and normalizes relative to counts of fragments with the same rout.

The estimation of Double-DOP thus uses the whole treebank, whereas DOP* uses the split. We compare these approaches by using either a split or the whole set of trees for both estimators. We will plot the fragments according to the weights assigned by the estimators, such that the differences can stand out. Furthermore, we can compare the grammars by having them parse a testset and determine their performance, e.g. the F1-score for correctly predicted parses.

Example This example clarifies how the grammars that result from Double-DOP and DOP* can actually differ. Figure 1 shows a small artificial treebank with four trees. In figure 2 all fragments are displayed that are in the symbolic grammar after applying the maximal overlap or shortest derivation extraction to this corpus. In table 1 it can be seen where these fragments originate from. Moreover, the table gives the weights assigned by both methods. Note that in the maximal overlap case, all PCFG rules in the treebank are added as well. The weight estimates are the relative frequencies of the fragments in the treebank: $p(f) = \frac{count(f)}{\sum_{f' \in F_{root(f)}} count(f')}$(**?**). As for the shortest derivation extraction, the weights are determined as the relative frequency of occurring in shortest derivations: $p(f) = \frac{r_c}{\sum_{k \in \{1 \ldots N\}: root(f_k) = root(f_j)} r_k}$(**?**).

Note the remarkable differences in the weight distributions. For example, f1 gets a weight of 0.5 in the maximal overlap approach, and zero in the shortest derivation case. Of course, the sparsity of the data contributes much to these extreme variations. However, the observed differences encourage us to investigate these two approaches into more depth.

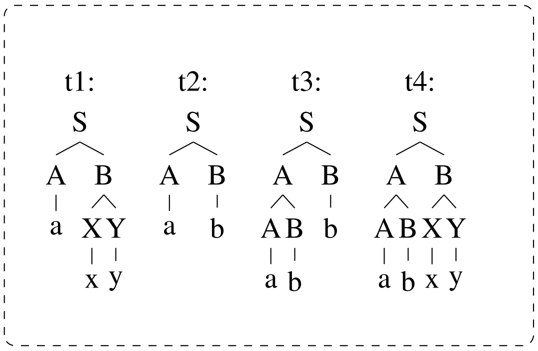

Figure 1: A toy treebank

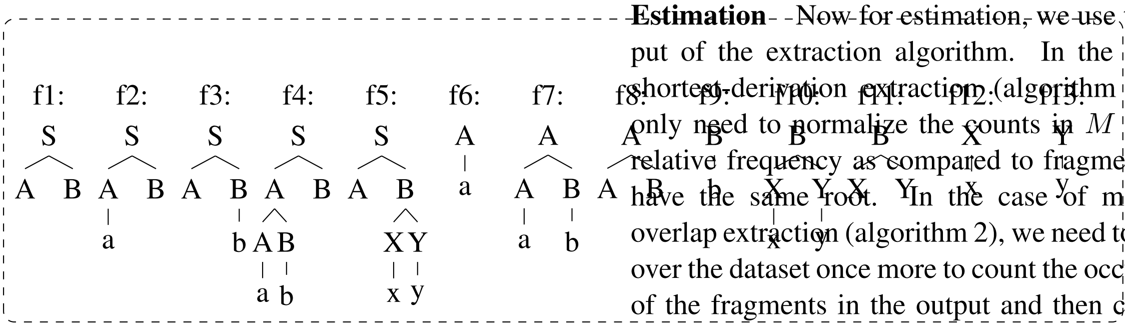

Figure 2: All extracted fragments

	Maximal overlap	Weight	Shortest derivation
f1	(t1,t3),(t2,t4)	4/12	-
f2	(t1,t2)	2/12	1b, 2a
f3	(t2,t3)	2/12	2b, 3b
f4	(t3,t4)	2/12	3a, 4b
f5	(t1,t4)	2/12	1a, 4a
f6	(t1,t3),(t1,t4),(t2,t3),(t2,t4)	4/6	1a, 2b
f7	-	0	3b, 4a
f8	- PCFG rule	2/6	-
f9	(t2,t3),(t2,t4),(t3,t4)	4/6	2a, 3a
f10	-	0	1b, 4b
f11	- PCFG rule	2/6	-
f12	- PCFG rule	2/2	-
f13	- PCFG rule	2/2	-

Table 1: The weights assignment according to both methods in a one vs. the rest manner

4.1 Implementation

Extraction Algorithm 1 performs shortest-derivation extraction (as in DOP*) in an efficient way, resembling the tree kernel approach in the iterative comparison of one tree with others. Namely, we iterate over the trees in HC (lines 2-8): create a list F of all of its fragments that occur in the rest of the treebank (line 3) and increment the count of the fragments in F that occur in the shortest derivation(s). In this way, we do not need to store all fragments in EC as in the initial step of the original DOP* algorithm.

Algorithm 2 performs the maximal-overlap extraction (as in Double-DOP). For the comparison of trees to find the maximal overlap (line 4), the tree kernel approach from (**?**) is used.

Estimation Now for estimation, we use the output of the extraction algorithm. In the case of shortest-derivation extraction (algorithm 1), we only need to normalize the counts in M to their relative frequency as compared to fragments that have the same root. In the case of maximal-overlap extraction (algorithm 2), we need to iterate over the dataset once more to count the occurences of the fragments in the output and then compute their relative frequencies.

[1] For this dataset, two shortest derivations exist for each tree. We refer to them with the following variables: 1a = f5, f6; 1b = f2, f9; 2a = f2, f8; 2b = f3, f6; 3a = f4, f8; 3b = f3, f7; 4a = f5, f7; 4b = f4, f9

Algorithm 1 Shortest derivation extraction in a one vs the rest manner

Data: a treebank TB

Result: a map of tree fragments and corresponding counts in shortest derivations

```
Initialize $M$, a map from fragments to counts, empty
for $t \in TB$ do
    $F \leftarrow Frag(t) \cap \bigcup_{t' \in TB/\{t\}} frag(t')$
    $D \leftarrow$ the shortest derivation(s) of $t$ using fragments in $F$
    for $f \in D$ do
        add $f$ to $M$ if $f \notin M$
        $M[f] \leftarrow M[f] + 1$ ▷ Anticipating the estimation step
    end for
end for
```

Algorithm 2 Maximal overlap extraction in a one vs the rest manner

Data: a treebank TB

Result: a set of tree fragments

```
Initialize $M$, a map from fragments to weights, empty
for $t \in TB$ do
    for $t' \in TB/\{t\}$ do
        $f \leftarrow$ maximal overlapping fragment(s) of $t$ and $t'$
        add $f$ to $M$ if $f \notin M$
    end for
end for
```

5 Results

6 Conclusion