Durham Research Online

Deposited in DRO:
14 November 2016

Version of attached file:
Published Version

Peer-review status of attached file:
Peer-reviewed

Citation for published item:

Further information on publisher’s website:

Publisher’s copyright statement:

Additional information:

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in DRO
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full DRO policy for further details.
Deterministic Choices in a Data-driven Parser

Sardar Jaf1, Allan Ramsay1

1The University of Manchester, Faculty of Engineering and Physical Sciences, School of
Computer Science, Manchester, United Kingdom
{sardar.jaf, allan.ramsay}@manchester.ac.uk

Abstract. Data-driven parsers rely on recommendations from parse models, which are generated from a set of training data using a machine learning classifier, to perform parse operations. However, in some cases a parse model cannot recommend a parse action to a parser unless it learns from the training data what parse action(s) to take in every possible situation. Therefore, it will be hard for a parser to make an informed decision as to what parse operation to perform when a parse model recommends no/several parse actions to a parser. Here we examine the effect of various deterministic choices on a data-driven parser when it is presented with no/several recommendation from a parse model.

1 Introduction

One of the main components of a data-driven parser is a parse model, which recommends parse operations to a parser when processing sentences. It is not guaranteed that a parse model can cover every possible situation during parsing and hence it may be unable to recommend a parse operation or it may recommend several operations in a given situation. Therefore, when a parse model recommends no/several operations to a parser, it will be hard for the parser to determine what operation to perform. In Section 3 we will describe a basic shift-reduce parser while in Section 4 we will describe our parser. In Section 6 we will identify several deterministic choices that a data-driven shift-reduce parser may take. We will examine the effect of these deterministic choices on the parsing performance in terms of efficiency and accuracy. In Section 8.1, we will examine the effect of various deterministic choices when running our parser deterministically, and in Section 8.2 we will examine the effect of the deterministic choices on our parse when running it non-deterministically.

2 Dataset

We have used the Penn Arabic Treebank (PATB) (Maamouri and Bies, 2004) part 1 version 3 for training and testing our dependency data-driven parser, which is a re-
implementation of the arc-standard version of MaltParser (Nivre et al., 2010; Kuhlmann and Nivre, 2010; Nivre et al., 2006). We have converted the phrase structure trees of the PATB to dependency structure trees using the standard conversion algorithm for transforming phrase structure trees to dependency trees, as described by Xia and Palmer (2001). In order to perform a 5-fold validation, we have systematically generated five sets of testing data and five sets of training data from the treebank, where the testing data is not part of the training data. The training data contains approximately 3853 sentences. The average length of sentences is 29 words and the total number of testing sentences in each fold is about 970 sentences.

3 A Shift-reduce Parser

A basic shift-reduce parsing algorithm performs one out of three operations at any parse transitions: SHIFT, LEFT-ARC or RIGHT-ARC. These operations are applied to a queue of words which have not yet been looked at and a stack of words which have been inspected but have not yet been assigned a syntactic role.

The SHIFT operation moves the head of the queue to the top of the stack. The LEFT-ARC and RIGHT-ARC operations establish head-dependent relations (in dependency parsing) between the head item of the queue and the top item on the stack. The LEFT-ARC and the RIGHT-ARC operations are applied to one node in a queue of input strings and one node on the stack. The LEFT-ARC operation makes the first node in the queue the parent of the top node on the stack while the RIGHT-ARC operation makes the top node on the stack the parent of the first node in the queue and rolls back the item on the top of the stack to the queue.

Our parser implementation is similar to the arc-standard algorithm of MaltParser (Kuhlmann and Nivre, 2010), which takes a deterministic approach to parsing natural language text where a support vector machine (SVM) (Chang and Lin, 2001) classifier is used for learning parse operations from a dependency treebank. The classifier helps the parser to predict the most likely correct parse operation when it is presented with a non-deterministic choice between multiple parse operations. As Nivre (2008) states, in this kind of implementation the parser derives a single parse analysis by incrementally selecting a parse operation, which makes the parsing process very simple and efficient. Moreover, by using an appropriate classifier, a good parsing accuracy is achievable (Nivre, 2008, p. 514).

The original arc-standard algorithm uses a deterministic approach to parsing natural language texts. The parser follows suggestions made by a parse model to perform a specific parse action (SHIFT, LEFT-ARC, or RIGHT-ARC) at each parse step. Performing the wrong parse action at a particular step during parsing will have a knock on effect on subsequent parsing steps. Hence, the error propagation can be substantial. Using a non-deterministic approach, where the parser is presented with multiple actions to take, allows the parser to recover from a previous mistake if this is subsequently identified.
4 DNDParser

Our parser contrasts with MaltParser in the way it is non-deterministic but with some deterministic features. We will call our parser DNDParser, which is short for deterministic and non-deterministic dependency data-driven parser. At each parse step, we generate a state for SHIFT, LEFT-ARC, and RIGHT-ARC, and we will assign different scores to each state. The score of each state is computed by using two different scores: (i) a score that is based on the recommendation made by the parse model. For example, we give a score of 1 for a SHIFT operation if it is recommended by the parse model, otherwise we give it a score of 0 (and the same applies to LEFT-ARC and RIGHT-ARC). (ii) We add the score from (i) to the score of the current state (which is the state that the new parse state is generated from). The sum of these two scores is assigned to the newly generated parse state(s). We can rank a collection of parse states by using their scores and then process the state with the highest score, which we consider the most plausible state. The various states generated by our parser is described in the following section.

5 Assigning Scores to Parse States

We extend the LEFT-ARC and RIGHT-ARC operations of the shift-reduce algorithm to allow more variations of the reduce operations, such as LEFT-ARC($n$) and RIGHT-ARC($n$) where $n$ is any positive numbers. In this way, our parser generates one or more parse states from a given state based on following situations:

- If the queue consists of one or more items and the stack is empty then the parser produces one state by performing SHIFT. For example, if the queue consists of items such as [1, 2, 3, 4] and an empty stack such as [] then the parser cannot recommend LEFT-ARC($n$) or RIGHT-ARC($n$) because these two operations require an item on the stack to be made the parent or the daughter of the head of the queue respectively.
- If the queue consists of one or more items such as [2, 3, 4] and the stack consists of one item only such as [1], then there are three possible moves: SHIFT, LEFT-ARC(1), and RIGHT-ARC(1). However, the parse model, which is based on a classification algorithm, will recommend only one operation (SHIFT, LEFT-ARC(1), or RIGHT-ARC(1)). Hence, in this kind of state our parser generates three states but only one state will be given a positive score, which is based on recommendation of the parse model.
- If the queue consists of one or more items such as [3, 4] and the stack consists of more than one item such as [2, 1], then our parser may generate more than three states because it checks for relations between the head of the queue and any items on the stack; i.e., states
that are generated by LEFT-ARC\((n+1)\) and RIGHT-ARC\((n+1)\). This approach is a generalisation of proposals by Kuhlmann and Nivre (2010) and Attardi (2006).

We store the states with various scores in an agenda sorted based on their scores, and the state with the highest score is explored by the parser.

### 6 Classification-driven Deterministic Parsing

During some parse transitions, DNDParser may be forced to make deterministic decisions. As explained in the previous section, if the parser is presented with a state that has one or more items on the queue but an empty stack then it will produce one state by performing SHIFT. For example, having a queue with \([1, 2, 3, 4]\) and an empty stack \([\ ]\) then the parser cannot recommend LEFT-ARC or RIGHT-ARC because both of these two operations requires an item from the stack to be made the parent or the daughter of the head of the queue.

Having one or more items on the queue and one item on the stack the parser produces three states, namely: SHIFT, LEFT-ARC, and RIGHT-ARC. In this kind of situation, the parsing model recommends only one operation where we give it a positive score so that the parser can then explore the recommended operation. However, it is possible that the parse model may not recommend any operations if it is presented with a situation that has never seen it during training. This is possible because the classifier may not learn what action to take in every situation the parser encounters during the testing phase. For example, in Fig. 1 we assume that the parse model did not recommend any operation, where all three operations receive a score of 0, and thus they will all have equal scores (which is the score inherited from the original state).

<table>
<thead>
<tr>
<th>States</th>
<th>Operations</th>
<th>Queue</th>
<th>Stack</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current state</td>
<td>-</td>
<td>[2, 3, 4]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>New states</td>
<td>SHIFT</td>
<td>[3, 4]</td>
<td>[2, 1]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RIGHT-ARC((1))</td>
<td>[1, 3, 4]</td>
<td>[]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>LEFT-ARC((1))</td>
<td>[2, 3, 4]</td>
<td>[]</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 1. Generating three parse states from one state

In this kind of situation, it is not clear which operation the parser should explore first, LEFT-ARC\((1)\), RIGHT-ARC\((1)\) or SHIFT. There are six different deterministic strategies (order-of-preference) we can give to the parser as to which operation it should explore first, those are:

1. SHIFT-LEFT-ARC-RIGHT-ARC
2. SHIFT-RIGHT-ARC-LEFT-ARC
3. LEFT-ARC-SHIFT-RIGHT-ARC
4. LEFT-ARC-RIGHT-ARC-SHIFT
5. RIGHT-ARC-SHIFT-LEFT-ARC
6. RIGHT-ARC-LEFT-ARC-SHIFT

Furthermore, in situations where the parser is presented with a state that has one or more items on the queue and more than one items on the stack, the parser can then generate more than three states because it checks for relations between the head of the queue and any items on the stack; i.e., states that are generated by LEFT-ARC(n+1) and RIGHT-ARC(n+1). In this kind of situation, it is possible that two or more operations may be recommended by the parse model, where two or more states receive positive scores. For example, in Fig. 2 where the parsing rules suggested LEFT-ARC(1)(making 3 from the queue the parent of 2 on the stack) and also LEFT-ARC(2)(making 3 the head of the queue the parent of 1 from the stack) they are both given a score of 1.

Fig. 2. Generating more than three parse states from one state

In this kind of exemplified situation we may deterministically choose to perform LEFT-ARC(1) instead of LEFT-ARC(2), by giving more priority to reduce operations that involve two items that are closer to each other. Alternatively, we may deterministically choose LEFT-ARC(2), by giving priority to reduce operations that involve two items that are further away from each other. This leads to another two different deterministic choices, which are:

1. furthest-item-first: this operation involves making relations between the head of the queue and an item that is furthest away from it on the stack.
2. closest-item-first: this operation involves making relations between the head of the queue and an item on the stack that is closest to it on the stack.

We can run the parser deterministically by allowing it to accept the first terminal state that it produces, which is a state where there are no possible actions for the parser to take (i.e. if the queue is empty).
7 Classification-driven Non-deterministic Parsing

Running our parser completely deterministic, then we allow it to accept the first terminal state it produces (whether a well-formed tree is produced); i.e., when the queue becomes empty because processing of all the words in it is performed by removing queue items on to the stack. If we run the parser non-deterministically, we allow it to explore the alternative states that remain on the agenda if the first terminal state is not well-formed; i.e., where the stack has more than one item on it, which means that some words did not receive a parent and hence a complete parse tree is not produced. This means that the parser rolls back to the previous highest scored state on the agenda and explores it until a state is generated whereby the stack contains one item and a complete parse tree is generated.

8 Evaluation

In this section, we will present our evaluation of the deterministic and non-deterministic versions of DNDParser. We show three different parsing accuracy measures, those are: (i) Labelled Attachment Scores (LAS), which is the percentage of the correct dependency relations with the correct labels of the dependency relations (DEPREL) between tokens; (ii) Unlabelled Attachment Score (UAS), which is the percentage of correct dependency relation (i.e., the percentage of tokens with correct heads) regardless of the DEPREL; and (iii) Labelled Accuracy (LA) which is the percentage of tokens with the correct dependency label. The efficiency of the parser is also presented, which is amount of time in seconds the parser consumes for establishing a dependency relation between two words.

8.1 Deterministic Parser Evaluation with Various Deterministic Choices

In this section we will evaluate DNDParser by running it completely deterministic. In deterministic mode, the parser accepts the first terminal state it produces regardless of whether the state contains a complete parse tree for a given sentence. Moreover, we present results for the various deterministic strategies, which we outlined in Section 6. We can observe from Table 1 that from the six deterministic order-of-preferences, the LEFT-ARC-SHIFT-RIGHT-ARC strategy produces the highest parsing accuracy.

We can also observe that the LEFT-ARC-SHIFT-RIGHT-ARC order-of-preference produces higher parsing accuracy when combined with the furthest-item-first reduction strategy than when it is combined with the closest-item-first reduction strategy. However, combining the LEFT-ARC-SHIFT-RIGHT-ARC order-of-preference with the furthest-item-first reduction strategy degrades the parsing efficiency by about 7% compared with when it is combined with the closest-item-first reduction strategy.
Deterministic Choices in a Data-driven Parser

Table 1. Deterministic parsing evaluation

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Furthest-item-first reduction</th>
<th>Closest-item-first reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS (%)</td>
<td>LAS (%)</td>
</tr>
<tr>
<td>LEFT-ARC-SHIFT-RIGHT-ARC</td>
<td>72.48</td>
<td>70.63</td>
</tr>
<tr>
<td>SHIFT-LEFT-ARC-RIGHT-ARC</td>
<td>59.77</td>
<td>58.12</td>
</tr>
<tr>
<td>SHIFT-RIGHT-ARC-LEFT-ARC</td>
<td>59.41</td>
<td>57.76</td>
</tr>
<tr>
<td>RIGHT-ARC-LEFT-ARC-SHIFT</td>
<td>53.67</td>
<td>52.25</td>
</tr>
<tr>
<td>LEFT-ARC-RIGHT-ARC-SHIFT</td>
<td>53.67</td>
<td>52.25</td>
</tr>
<tr>
<td>RIGHT-ARC-SHIFT-LEFT-ARC</td>
<td>53.27</td>
<td>52.15</td>
</tr>
</tbody>
</table>

8.2 Non-deterministic Parser Evaluation with Various Deterministic Choices

In this section we will evaluate our parser by running it non-deterministically. In this mode, the parser explores other states until it finds a well-formed terminal state, which is a state where the stack contains one item and a complete parse tree is generated. We run the parser in this mode by integrating various deterministic strategies that we outlined in Section 6. We can note from Table 2 that from the six deterministic order-of-preferences (see Section 6 for more detail), the SHIFT-LEFT-ARC-RIGHT-ARC order-of-preference produces the highest parsing accuracy. We can also observe that the SHIFT-LEFT-ARC-RIGHT-ARC order-of-preference produces higher parsing accuracy when combined with the furthest-item-first reduction strategy than when it is combined with the closest-item-first reduction strategy. However, combining the SHIFT-LEFT-ARC-RIGHT-ARC strategy with any of the two strategies (furthest-item-first reduction or closest-item-first reduction) the speed of the parse is not largely affected (about 2.4%).

It appears that using different settings affects the performance of the parser greatly. From the experiments conducted in this section, and the previous section, it is apparent that running the parser non-deterministically with SHIFT-LEFT-ARC-RIGHT-ARC order-of-preference and using the furthest-item-first reduction strategy produces the best parsing performance.
Table 2. Non-deterministic parsing evaluation with different deterministic choices

<table>
<thead>
<tr>
<th>Strategy</th>
<th>UAS (%)</th>
<th>LAS (%)</th>
<th>LA (%)</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furthest-item-first reduction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHIFT-LEFT-ARC-RIGHT-ARC</td>
<td>74.5</td>
<td>71.0</td>
<td>93.6</td>
<td>0.081</td>
</tr>
<tr>
<td>LEFT-ARC-SHIFT-RIGHT-ARC</td>
<td>72.6</td>
<td>70.7</td>
<td>92.0</td>
<td>0.072</td>
</tr>
<tr>
<td>SHIFT-RIGHT-ARC-LEFT-ARC</td>
<td>57.5</td>
<td>55.8</td>
<td>88.1</td>
<td>0.074</td>
</tr>
<tr>
<td>RIGHT-ARC-LEFT-ARC-SHIFT</td>
<td>53.6</td>
<td>52.2</td>
<td>87.9</td>
<td>0.060</td>
</tr>
<tr>
<td>LEFT-ARC-RIGHT-ARC-SHIFT</td>
<td>53.6</td>
<td>52.2</td>
<td>87.9</td>
<td>0.059</td>
</tr>
<tr>
<td>RIGHT-ARC-SHIFT-LEFT-ARC</td>
<td>53.6</td>
<td>52.2</td>
<td>87.9</td>
<td>0.060</td>
</tr>
</tbody>
</table>

| Closest-item-first reduction     |         |         |        |            |
| SHIFT-LEFT-ARC-RIGHT-ARC        | 70.75   | 68.95   | 91.0   | 0.079      |
| LEFT-ARC-SHIFT-RIGHT-ARC        | 66.48   | 64.74   | 90.7   | 0.058      |
| SHIFT-RIGHT-ARC-LEFT-ARC        | 57.01   | 55.27   | 88.1   | 0.077      |
| RIGHT-ARC-LEFT-ARC-SHIFT        | 52.55   | 51.11   | 87.8   | 0.056      |
| LEFT-ARC-RIGHT-ARC-SHIFT        | 51.34   | 49.96   | 87.8   | 0.052      |
| RIGHT-ARC-SHIFT-LEFT-ARC        | 51.34   | 49.96   | 87.8   | 0.051      |

9 Summary

Parse models are one of the main elements of data-driven parsers. They are used for guiding parsers during the processing of natural languages. However, it is possible that parse models may recommend no/several parse operations to a parser in a given situation. When parse models recommend no/several parse operations it is difficult for a parser to determine what operation to perform. Therefore, they are allowed to make deterministic choices. In this paper, we have identified several deterministic choices that a parser may take when it is presented with no/several parse operations, which are recommended by a parse model. We have observed and examined the effect of each deterministic choice on the performance of a data-driven parser, which is based on the shift-reduce algorithm. We have identified that each deterministic choice affects the parsing performance in different ways. Some choices affect accuracy while other choices affect efficiency.

Acknowledgment

Sardar Jaf's contribution to this work was supported by the Qatar National Research Fund (grant NPRP 09-046-6-001). Allan Ramsay's contribution was partially supported from the same grant.
References


