CS 187 - Dependency Parsing
Phase 4: Final Paper Draft

Lauren Urke, Nathaniel Herman, Henrik Sigstad, Ariel Camperi

Abstract

The abstract ...

1 Introduction

- Motivation
- What we did and why we think it is important
- Statement of key result

2 Statement of the problem

- Further discussion about why we care...

3 Description of the method

3.1 Description of SVMs

3.2 Description of dependency parsing

3.3 Description of code

Our actual implementation uses the same basic parsing algorithm as that in [1] (i.e. Figure 6 of the paper). Essentially, we look at 2 elements of our array (which is initially the list of words in a sentence) at a time, using the contextual features (like POS tags) to estimate the relation (Left, Right, or Shift) to use between the 2 elements.

Of course, the hard part of the implementation is not in writing the basic parsing algorithm, but rather in writing the get_contextual_features(), estimate_action(), and construction() functions to work correctly for both the training phase and prediction phase (as well as generating the SVM(s) themselves and saving them to disk).

Our array (the "T" variable in [1]) starts as an array of words, which are really dictionaries describing a given word's features like its POS. As we construct child-parent relations, the array elements become trees themselves representing these child-parent relations. To make an element the child of a tree, we append the element to the tree's list of children.

get_contextual_features() works the same whether we're training or predicting, and simply creates a dictionary which maps various information tuples (including information about a tree's children, and contextual information) to 1's. For instance we might have a dictionary (in Python) that looks something like:

```
{(0, 'pos', 'NP'): 1, (0, 'lex', 'resort'): 1,
  (0, 'ch-R-pos': 'JJ'): 1, (0, 'ch-R-lex': 'last'): 1}
```

This specific format is to make it easily convertible to a matrix of 1's and 0's for our SVM.

For the training phase, estimate_action() works as follows: we look to see if the two nodes in consideration have an actual parent-child relation to each other (in the training dataset). If so, we choose Left or Right as our action to construct this parent-child relation. However, once a node is made the child of another node, we can no longer add any children to it. Thus, we only pick Left or Right if the node which will become child has already had all of its own children added to it. In all other cases we pick Shift as our action.

The training model also keeps track of a list of its decisions. We store both what action it picks, as well as what the contextual features were for that particular action choice. Thus, once the training model has been run on the data, we can construct an SVM which associates sets of contextual features with actions, and store it to disk (so we don't have to retrain each run through).

Then we can run our predictor model on test data, where estimate_action() now predicts what action to take using our SVM with the contextual features at the current location in the sentence.

In our full implementation, we actually use multiple SVMs, one per POS tag. Thus to predict the action for the ith element in our array, we use the SVM associated with whatever part of speech the ith word in the array is. This makes it more tractable to run our data with very large datasets (the Penn treebank dataset is fairly large) so we don't end up with one gigantic SVM.

4 Description of data

To implement Dependency Parsing, we used the Penn Treebank Project Data. The treebank data contains sentences that have been parsed into a linguistic trees that also contain part-of-speech tagging.

Parsed in this way, "Pieere Vinken, 61 years old, will join the board as a non executive director Nov. 29.", will look like this:

NEED TO FIGURE OUT HOW TO TAB THIS

((S
(NP-SBJ
(NP (NNP Pierre) (NNP Vinken))
(, ,)

```
(ADJP
(NP (CD 61) (NNS years) )
(JJ old) )
(, ,) )
(VP (MD will)
(VP (VB join)
(NP (DT the) (NN board) )
(PP-CLR (IN as)
(NP (DT a) (JJ nonexecutive) (NN director) ))
(NP-TMP (NNP Nov.) (CD 29) )))
(. .) ))
```

This information is helpful, but could be improved. To conduct analysis on the dependencies, we wanted to be able to easily infer the relationship between two words. Therefore, we used a program from the Lund University Computer Science Department that converted the Penn Treebank parses into dependency parses. For each word in a sentence, a dependency parse indicates the parent word and the part of speech. For example, the same sentence now comes out as this:

ID	Token	Part of Speech	Parent ID	Class
1	Pierre	NNP	2	NAME
2	Vinken	NNP	8	SBJ
3	,	,	2	P
4	61	CD	5	NMOD
5	years	NNS	6	AMOD
6	old	JJ	2	APPO
7	,	,	2	P
8	will	MD	0	ROOT
9	join	VB	8	VC
10	the	DT	11	NMOD
11	board	NN	9	OBJ
12	as	IN	9	ADV
13	a	DT	15	NMOD
14	nonexecutive	JJ	15	NMOD
15	director	NN	12	PMOD
16	Nov.	NNP	9	TMP
17	29	CD	16	NMOD
18	.	.	8	P

This data is now much more helpful for determining whether there is a direct parental relationship between two words in a sentence. The Penn Treebank contains 24 sections. We used sections 02-22 for training and section 23 for testing. Section 23 contains 2,416 sentences, which includes 56,684 words (49,892 without punctuation).

5 Results

5.1 Results

We used three measurements to evaluate the effectiveness of this method.

Dependency Accuracy $= \frac{\text{number of correct parents}}{\text{total number of parents}}$
Root Accuracy $= \frac{\text{number of correct root nodes}}{\text{total number of sentences}}$
Complete Rate $= \frac{\text{number of complete parsed sentences}}{\text{total number of sentences}}$

Using a default context of (2,2), we evaluated two different SVC methods. With a One vs One Linear SVC method, we obtained the following results:

Dependency Accuracy	Root Accuracy	Complete Rate
0.89135	0.93675	0.32711

Using a One vs Many Linear SVC, we obtained the following, slightly worse results:

Dependency Accuracy	Root Accuracy	Complete Rate
0.88680	0.93894	0.32285

The Sci-kit Learn offered several other SVM options, but after testing each of them with a set context length, we determined that none of them improved our results. We then empirically determined the optimal context length for the One vs One classifier. The Context determines how many words to the left and right of the target nodes are considered.

	(2,2)	(2,3)	(2,4)	(2,5)	(3,2)	(3,3)	(3,4)	(3,5)
Dep. Acc.	0.891	0.890	0.890	0.889	0.887	0.887	0.887	0.887
Root Acc.	0.937	0.928	0.934	0.930	0.932	0.929	0.928	0.927
Comp. Rate	0.327	0.330	0.325	0.326	0.312	0.318	0.316	0.317

We determined that the One vs One Linear SVC gave the best results when used with a context of 2 on either side of the target nodes.

6 Conclusion

Yamada and Matsumoto's implementation of dependency parsing used a One vs One classifier to parse the sentences. They also determined that a context of 2 on either side of the target nodes was best. Their implementation gave only slightly better results:

	Dependency Accuracy	Root Accuracy	Complete Rate
Yamada and Matsumoto	0.900	0.896	0.379
Our results	0.89135	0.93675	0.32711

Our implementation successfully modeled Yamada and Matsumoto's dependency parsing algorithm with fairly similar results. In theory, a One vs Many classifier should work better than a three-part One vs One classifier. Possible next steps to improve results include investigating other One vs Many classifiers and attempting to pinpoint why this classifier did not improve parsing accuracy.

References

[1] Yamada, H., and Matsumoto, Y. Statistical Dependency Analysis With Support Vector Machines. In *Proceedings of IWPT*, 2003.