L645 Final Project Report

Ritesh Agarwal
School of Informatics and Computing
Indiana University
riteagar@indiana.edu

Misato Hiraga
Department of Linguistics
Indiana University
mhiraga@indiana.edu

Eriya Terada
Department of Computer Science
Indiana University
eterada@indiana.edu

Abstract

This paper will describe the background and various steps we have taken in order to implement a supervised machine learning technique for predicting correct sentence parsing.

1 Introduction

Probabilistic context-free grammars (PCFGs) have been used for syntactic parsing. The models use context free grammar with probability of how likely each CFG rule occur. They are the simplest and most natural probabilistic models for tree structures (MS99a). We are interested in choosing the right parsing structure among various possible parsing options that a parser outputs. We will use a supervised machine learning approach with CFG rules extracted from Penn TreeBank as features to classify optimal and incorrect parsing tree structures.

2 Motivation

The motivation for this paper comes from the fact that when a product like Siri is trying to understand a sentence uttered by a user, it generally fails to find relationships between words. We are trying to find the best syntactic representation for any sentence which will make an AI like Siri answer more to the point. Even though the NLP community is trying to build an AI that talks to a person like another person, it is far from the goal and interpreting any sentence in the right manner is the next step towards achieving the same. In future, we would further move our focus on syntactic dependency representation and speech recognition towards building an efficient AI.

3 Related Work

3.1 Relation to the NLP world and what we learned in class

We looked at a syntactic parsing model based on PCFG in class. The simple parser we can develop will calculate the likelihood of each word or phrase combination given the probability of occurrence of the words and phrases in a corpus. The phrase rules are abstract, such as N, NP, PP, which do not carry semantic meaning of each word, so the PCFG-based models cannot distinguish syntactic and semantically well-formed sentences from non-well-formed sentences.

Although a parser may not be able to distinguish ambiguous sentences, we may be able to classify correct and incorrect parses using a classifier. Such classifiers have been developed in the field of grammatical error detection in the learner languageand and machine translation outputs.

(TFC10) used parse features in their system that detects errors in choice of a preposition from English texts written by non-native speakers. Adding the parse tree features improved the classifier's perfomance significantly compared with a model without the parse features. This suggests that the parse features are informative in detecting syntactic errors. (LG05) used syntactic features to evaluate machine translation output. They used parse tree features and dependency tree features. These studies suggest the importance of parse tree features in evaluating correct syntactic structures. Following these studies, we use parse tree features and classify correct parse trees and incorrect parse trees.

4 Long- and Short-term Goals

Our long term goal is to develop a system that will be able to choose the best syntactic parse tree among many possible trees. For this task, collecting good features that can implement both syntactic and semantic information. Ultimately we will include semantic information (lexical items), but as a start, we only use syntactic parse information. Thus, our short-term goals (what we want to achieve in this paper) are as follows:

1. implement a PCFG-based model trained on the Wall Street Journal (WSJ) dataset,
2. use PCFG rules obtained from the first step to build a feature set to represent each sentence
3. generate possible parsing options for each sentence using the chart parser from NLTK and automatically label correct and incorrect trees by comparing with the Gold Standard from Penn Treebank.
4. write our own classification programs using Naive Bayes, Neural Network, RBF, Logistic Regression algorithms.
5. train the classifiers with the training set that contain correct and incorrect labeled sentences and test the classifier on unseen data to evaluate how well the classifiers can predict correct vs. incorrect syntactic parses.

5 Methods

5.1 PCFG

5.1.1 Background

The very first method we will be implementing is PCFG. The main idea behind incorporating Context-Free Grammars (CFG) with probabilities lies in the fact that in the real world, phrases are not uniformly distributed. If it is, then we would be reading something less meaningful for most of the time. Statistically, some grammatical constructions are more common than others, and some words are more common than others (as mentioned in Zipf's law). These kinds of frequencies can be measured by analyzing corpora and treebanks.

PCFG parsing gives you the most probable parse for a sentence based on the probabilities calculated from a corpus. Where a normal CFG parser might give you a list of all possible parses for the phrase "I saw the man with the telescope" (or maybe just an arbitrary parse of the phrase, which does not give us much information), a PCFG parser can give you the most probable parse or a ranking of possible parses, based on multiplying the probabilities of the rules necessary for the derivation of each parse. This makes for a more accurate natural language processing. Parsing can be done in $\Theta(n^3|G|)$ time (where n is the string length and $|G|$ is the size of the grammar) using the CYK (Cocke–Younger–Kasami) parsing algorithm or its variants.

There are some disadvantages to it. Despite being better suited for parsing and generation than non-stochastic CFGs, PCFGs are still not powerful enough to describe context-sensitive languages. Recall that they describe exactly the same languages as their non-stochastic counterparts. More extensions to PCFGs can be made to improve parsing performance, such as adding lexicalization (annotating categories with their headword), but they still won't be able to formally capture the kinds of dependencies found in natural language.

5.1.2 Implementation

The main structure that we followed comes from the course textbook (MS99b) and the slides shown in the lectures.

We first consult the entire WSJ corpora in the Penn Treebank (MMS93) and use sections 0 to 18, 19 to 21, 22 to 24, for training, development, and testing, respectively, as done by (SMN10). Because we use the CYK algorithm intended for grammars to be written in Chomsky normal form (CNF), we use the `treetransforms.chomsky_normal_form()` method provided by NLTK (BK09) to convert all production rules to CNF so that all rules are either in the form of $X \to YZ$ or $X \to x$, where Y and Z represent nonterminal symbols and x is a terminal word. This process takes about 50 seconds on a MacBook Pro with an Intel i7 CPU running at 2.3GHz.

In order to calculate the probability of a production rule being applicable to a newly seen sentence, we first count the number of occurrences for each rule by using a python `Counter`. The top 20 rules are shown in Table 1.

Now, in order to find the $P(X \to YZ)$, the probability of a rule $X \to YZ$, we need to calculate the sum of the occurrences of all rules that has X on its left hand side, and divide the frequency of that rule by this sum. This can be represented as

$$P(X \to YZ) = \frac{C(X \to YZ)}{\sum_\gamma C(X \to \gamma)} \qquad (1)$$

where $C(\cdot)$ is the number of occurrences for a rule. So for example, we have 37252 occurrences of the

Frequency	Production rule
43862	, → ’,’
37252	DT → ’the’
37006	PP → IN NP
35437	. → ’.’
30735	S → NP-SBJ VP
27887	S|<VP-.> → VP .
20883	IN → ’of’
20448	NP-SBJ → -NONE-
20135	TO → ’to’
19857	NP → NP PP
18827	NP → DT NN
17383	DT → ’a’
14509	CC → ’and’
13642	IN → ’in’
12810	S → NP-SBJ S|<VP-.>
11755	VP → TO VP
11300	NP|<JJ-NN> → JJ NN
10722	PP-LOC → IN NP
10485	-NONE- → ’*-1’
10437	NP → -NONE-

Table 1: The 20 most common rules and their frequency

rule $DT \rightarrow `the'$, and we have 74371 rules where the left hand side is $DT$. Hence, the probability for $DT \rightarrow `the'$ is $37252/74371 = 0.5008$.

Because the task of calculating the probabilities for all possible combinations of parse trees for a given sentence, we incorporate the CYK algorithm to utilize dynamic programming for efficiency. This also allows us to have a backpointer in which we could later use to reconstruct the most probable parse tree for a sentence.

In order to speed up this task in our program as much as possible, we use python dictionaries to store the probabilities for each production rule. We also use another dictionary to look up the probabilities for terminal words, called `terminal_probs` that stores a list of tuples containing a possible POS for a given word and its probability. This saves us from iterating through the first dictionary just to find the probabilities for terminal words. For example, `terminal_probs["‘bank’"]` returns `[(‘NN’, 0.0032080507982408), (‘VB’, 4.1625041625041625e-05)]`.

We assigned unseen words the probability of terminal nodes (such as N and V) that appeared in the training set. For example, if nouns appeared 50% of the time and verbs 50%, the unseen terminal word in the testing set was assigned 50% of being a noun and 50% of being a verb.

Method	Accuracy
Our PCFG in CNF	0.0073
Our PCFG in CNF with smoothing	0.0096

Table 2: Results

5.1.3 Results from PCFG

Following (SMN10), we limit our training and testing data to sentences from the Penn Treebank with at most 15 tokens, not including the final punctuation. The testing set contains 417 sentences extracted from section 22 of the Penn Treebank. The accuracy of our PCFG is shown in Table 2.

We would like to note that the accuracy is low because we only let a parse to be correct if it is exactly identical to the parse tree presented by the gold standard. Consider the sentence “The above represents a triumph of either apathy or civility." Shown below is its parse from the gold standard:

```
(S
  (NP-SBJ (DT The) (JJ above))
  (S|<VP-.>
    (VP
      (VBZ represents)
      (NP
        (NP (DT a) (NN triumph))
        (PP
          (IN of)
          (NP
            (DT either)
            (NP|<NN-CC-NN>
              (NN apathy)
              (NP|<CC-NN> (CC or) (NN civility)))))))
    (. .)))
```

And this is its parse from our PCFG:

```
(S
  (NP-SBJ (DT The) (JJ above))
  (S|<VP-.>
    (VP
      (VBZ represents)
      (NP-PRD
        (NP (DT a) (NN triumph))
        (PP
          (IN of)
          (NP
            (DT either)
            (NP|<NN-CC-NN>
              (NN apathy)
              (NP|<CC-NN> (CC or) (NN civility)))))))
    (. .)))
```

Notice how the two parses are identical except for the one POS tag where ours has a NP-PRD instead of NP below the word *represents*. This is due to the fine-grained nature of the gold standard in Penn Treebank, and we believe that ignoring these details will definitely increase our accuracy.

Method	Accuracy
Our PCFG in CNF	?
Our PCFG in CNF with smoothing	?

Table 3: Results for proper subtree overlap

As a compromise, we present another metric for measuring accuracy by comparing how many proper subtrees of our parse tree overlap with those from the gold standard. The basic idea is built upon the Jaccard similarity coefficient, and we present the algorithm below. Using this algorithm, we can calculate the similarity between two trees having 4 proper subtrees differing in only one subtree to have a value of $3/(4+4-3) = 0.6$. The accuracy measured with this method is shown in Table 4.

Algorithm 1 Algorithm for obtaining proper subtree overlap (PSO)

```
 1: procedure PSO($tree_1, tree_2$)
 2:     $union \leftarrow 0$
 3:     $subtrees_1 \leftarrow tree_1.properSubtrees()$
 4:     $subtrees_2 \leftarrow tree_2.properSubtrees()$
 5:     $size_1 \leftarrow len(subtrees_1)$
 6:     $size_2 \leftarrow len(subtrees_2)$
 7:     for all $subtree \in subtrees_1$ do
 8:         if $subtree \in subtrees_2$ then
 9:             $union \leftarrow union + 1$
10:             $tree_2.remove(subtree)$
11:         end if
12:     end for
13:     return $union/(size_1 + size_2 - union)$
14: end procedure
```

5.1.4 Difficulties

Here are some difficulties that we have encountered upon implementing PCFG:

1. As seen in Table 1, for some reason the grammar is not converted CNF in its entirety. There are symbols such as *-NONE-*, and there are rules that only have one non-terminal symbol on the right hand side. For our purposes we decided to ignore these rules in our calculations (therefore there will inevitably be sentences that cannot be parsed).

2. Due to a restriction in NLTK, we could only convert the grammars into CNF at the sentence level, and not at the corpus level. This may have caused certain rules to be divided into smaller, finer rules, creating a disjunction between the probabilities of those rules when they should have been treated as one whole.

Method	Accuracy
Naive Bayes	77.34
Logistic Regression	82.71
Radial Basis function	80.03
Neural Network	89.63

Table 4: Results for proper subtree overlap

5.2 Classifiers

We made each parse tree rule (except for terminal nodes) extracted from Penn Treebank as features, and represented sentences with a vector that contains 1 or 0. If a parse rule is found in a sentence, the vector for that sentence has 1 for that feature and if not, it would have 0. We initially planned to extract wrong trees by using a chart parser from NLTK, but it took too long to generate wrong trees. Thus, as our PCFG did not do well, we used the output of our PCFG as wrong parses and labeled them as 0 for incorrect, and we labeled 1 (correct) to the sentence representations from Penn Treebank. Our data contains 8856 instances of correct parses and 1000 instances of wrong parses. Algorithms used: Neural Network, Naive Bayes, Logistic Regression with and without regularizers and Radial Basis function. First of, we used Neural Network to get the accuracy where we created a dataset for trees. Each child-parent rlationship is taken as a feature and the dataset is created using Penn Tree bank. Neural Network is a connected set of networks used to evaluated the weights of features in each layer and the final weights gives us the class label. We have used one hidden layer and got the output further calculating the accuracy. the accuracy when tested on a very small dataset gave us 89.63 percent accuracy. We also implemented Naive Bayes on the same dataset the accuracy was 77.34 percent. Logistic Regression gave us 82.71 percent and Radial Basis function gave us 80.03 percent. A table for the same has been provided in Table 4. As we can see that Neural Network performs the best. We need to run script_classify.py to run the algorithm.

6 Future Work

As of now, the feature set includes only abstract syntactic information. We can include semantic information to improve our classification. This may be all the lexical items in the corpus or some of the lexical items that are key to solve ambiguity. For example, if we include verbs as feature along with their syntactic parse features, the verb can tell whether PP should attach to the VP or NP in cases where the verb is *put*. A naive parser may parse the verb phrase in the sentence *I put the book on the table* into VP→NP,PP or VP→NP and NP→ N,PP. Without the lexical information, the both syntactic rules are accepted. However, by adding lexical information the classifier can be trained that *put* most always requires two arguments and the first parse tree is correct. Another example is the sentence pair "I ate sushi with chopsticks" and "I ate sushi with tuna", which we discussed earlier. If we have the lexical feature "sushi" and "chopstics" with correct parse features, the classifier can tease apart correct and wrong syntactic trees. However, it may cause a problem when a classifier is faced with unseen nouns. We need some way of implementing semantic categories of words or similarity of nouns to correctly classify trees with unseen words.

7 Conclusion

We have tried a different approach as to what was used earlier and the results were not great but there is a lot of work that can be done and the algorithms can improve. For instance, we can create a much larger network of Neural Network which may increase the accuracy. Also trying different meta parameters will give us different results in case of Logistic Regression and Radial Basis Function. Overall, we can say that we have had a great time implementing this project and we hope to make it better in future

References

[BK09] Edward Loper Bird, Steven and Ewan Klein. Natural language processing with python. 2009.

[LG05] Ding Liu and Daniel Gildea. Syntactic features for evaluation of machine translation. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 25–32, 2005.

[MMS93] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[MS99a] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[MS99b] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[SMN10] Richard Socher, Christopher D Manning, and Andrew Y Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9, 2010.

[TFC10] Joel Tetreault, Jennifer Foster, and Martin Chodorow. Using parse features for preposition selection and error detection. In *Proceedings of the acl 2010 conference short papers*, pages 353–358. Association for Computational Linguistics, 2010.