

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, HYDERABAD

LANGUAGE TECHNOLOGY AND RESEARCH CENTER

MONSOON 2017

NATURAL LANGUAGE PROCESSING

PROJECT REPORT - TEAM 29

Improving parsing through distributional representation

Samyak Jain

201501074

Devansh Manu

201556073

Raghu Vamshi

20150165

The aim of this project was to improve an already existing PCYK parser using distributional semantics so that it is able to handle words that are not present in its Context Free Grammar (CFG).

The **Cocke–Younger–Kasami algorithm** (alternatively called **CYK**, or **CKY**) is a parsing algorithm for CFGs, that employs bottom-up parsing and dynamic programming.

The standard version of CYK operates only on context-free grammars given in Chomsky normal form (CNF). However any context-free grammar may be transformed to a CNF grammar expressing the same language.

The importance of the CYK algorithm stems from its high efficiency in certain situations. The worst case running time of CYK is $O(n^3 |G|)$ where n is the length of the parsed string and $|G|$ is the size of the CNF grammar G . This makes it one of the most efficient parsing algorithms in terms of worst-case asymptotic complexity, although other algorithms exist with better average running time in many practical scenarios.

The improvement we tried was in two aspects :

1. Handling unseen words
2. Improving span of grammar

We took the brown corpus and created distributional semantics out of it using *gensim library* , so now we had a vector representation for every word in the brown corpus. We would be using this vector representation in the PCYK parser.

Handling unseen words:

When we get a sentence to parse, we check for the words which are unseen. Unseen words are those which are in the vocabulary (brown corpus in this case) but not in the parser grammar and then using the distributional representation we find the closest word to it which is also there in the grammar and replace the unseen word with that word.

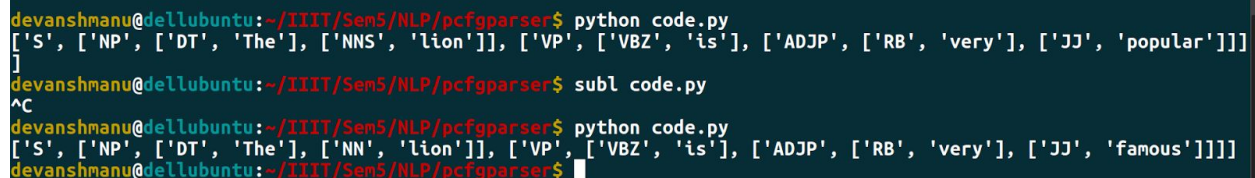
This technique helps improving the case where we have unseen words in the sentence.

For example, when we tried to parse the sentence 'The lion is very strong', 'lion' and 'strong' were unseen words in our parser's grammar. However, after replacing them with the closest words using distributional representation, we ended up with the sentence 'The prayer is very red' which although does not sound suitable, the sentence

semantics are the same as both prayer and lion are nouns, and strong and red are adjectives.

Now in this following example, we took two sentences “The lion is very popular” and “The lion is very famous”.

One word exists in the vocabulary and the other doesn't, but still the improvised parser, parses the sentence and gives the most relatable semantics

A terminal window with a dark blue background and light green text. It shows the execution of a Python script 'code.py' twice. The first execution processes the sentence 'The lion is very popular' and outputs a list of semantic categories: ['S', ['NP', ['DT', 'The'], ['NNS', 'lion']], ['VP', ['VBZ', 'is'], ['ADJP', ['RB', 'very'], ['JJ', 'popular']]]. The second execution processes the sentence 'The lion is very famous' and outputs a similar list, but with 'famous' instead of 'popular' in the final category: ['S', ['NP', ['DT', 'The'], ['NN', 'lion']], ['VP', ['VBZ', 'is'], ['ADJP', ['RB', 'very'], ['JJ', 'famous']]].

```
devanshmanu@dellubuntu:~/IIIT/Sem5/NLP/pcfparser$ python code.py
['S', ['NP', ['DT', 'The'], ['NNS', 'lion']], ['VP', ['VBZ', 'is'], ['ADJP', ['RB', 'very'], ['JJ', 'popular']]]
]
devanshmanu@dellubuntu:~/IIIT/Sem5/NLP/pcfparser$ subl code.py
^C
devanshmanu@dellubuntu:~/IIIT/Sem5/NLP/pcfparser$ python code.py
['S', ['NP', ['DT', 'The'], ['NN', 'lion']], ['VP', ['VBZ', 'is'], ['ADJP', ['RB', 'very'], ['JJ', 'famous']]]]
devanshmanu@dellubuntu:~/IIIT/Sem5/NLP/pcfparser$
```

Improving span of grammar:

Our idea to improve the span of the grammar was to first modify our grammar using Clustering. For example : Any english grammar is very likely to have the following two rules

VB -> smiling VB -> laughing and many other rules such that the terminals are close in meaning. So here its a possibility that we can reduce our grammar without compromising much of accuracy if we build the grammar such that the terminals are not words but word clusters. So words like laughing and smiling will lie in the same cluster and therefore we are able to reduce the number of rules in our grammar which improves the efficiency of the parser. Now an unseen word instead of being compared to another

word would now be compared to various cluster centres and then we would find a word from that respective cluster such that it also exists in our CFG.

However, during our implementation, this wasn't as practical as 95% of the words were clustered into one cluster. Theoretically, if we have many clusters though (for example, an average of around 10-20 words per cluster), this model would perform much better than the regular CFG grammar terminals.