Generate Random Sentence using PCFG

Group Members:
Huili Chen
Xinyi Wang

Relevance

Our goal of project is to build an automatic English sentence generator using probabilistic context-free grammars (PCFGs), which is a probabilistic version of regular context-free grammars that we have learned in class.

Effort

Huili Chen: 3.5 hrs (Mar. 20th), 3 hrs (Mar. 21st), 45 (Mar. 22nd). Total: 7hrs 15mins
Xinyi Wang: 3hr (Mar. 20), 3hrs (Mar. 21st), 1.5 hrs(Mar. 22rd). Total: 7hrs 30mins

Introduction

Probabilistic context-free grammars (PCFGs)

PCFGs have been used to generate random sentence. Our project did not choose to use regular context-free grammars (CFGs) due to its significant limitations on handling constituency relations in a natural language sentence. For example, CFGs perform poorly at dealing with the "subject-verb agreement" and "sub-categorization" issues [1]. More intuitively speaking, CFGs would fail to fit the verb in a given English sentence to its subject or have control of the verb's tense. Thus, we chose the PCFGs, which are more sophisticated versions of CFGs but can more accurately model English sentences than regular CFGs do.

The PCFGs, which are also called stochastic context-free grammars (SCFGs), have been widely used for building high-performance parsers[2][3][4]. They are often used to solve the ambiguity problem in natural language syntax [5][6]. It extends regular CFGs by adding a probability distribution over all possible derivations [7]. For example, the probability of an English sentence can be displayed in the format below.

$$P(w_{1n}) = \sum_{t} P(w_{1n}, t)$$

where t is a parse tree of w_{1n}

An example of the PCFGs is shown below.

Sentence -> [noun phrase] [verb phrase]. Probability: 1.0
[verb phrase] -> [verb] [noun phrase]. Probability: 0.2
[proposition] -> at. Probability: 0.42
[prepositional phrase] -> [preposition] [noun phrase]. Probability: 0.36

This probability distribution allows us to rank over possible parses for a given sentence based on their probabilities. This "simply" addition of a probability distribution enables PCFGs to handle the issues of ambiguity in natural English sentences by ranking n parses with respect to their probability scores. This advantage of PCFGs is our major motivation of using it to generate random sentences in our project. Another motivation comes from its easy implementation. Since it only requires an additional probability distribution, it is relatively easy for us to implement in Python.

However, we were also conscious of their limitations. For example, they have difficulty modeling lexical dependencies that other methods (e.g., n-grams) may able to capture [8]. Thus, the performance of an PCFG-based automatic random sentence generator may suffer from its drawbacks.

Berkeley Parser
Our project uses Berkeley Parser, of which source code can be directly downloaded online [9]. The Berkeley Parser, which is developed on the basis of PCFGs, can improve the naive grammar of a PCFG that "dummily" incorporates the empirical rules and probabilities of a treebank [10]. More specifically, one major advantage of this Berkeley Parser is its capability to learn grammars of much smaller size and higher accuracy than other grammar parsers [10]. For example, the Berkeley Parser achieves a grammar's accuracy higher than fully lexicalized systems (e.g., Charniak and Johnson's maximum-entropy based parser) [4]. The method starts with the barest possible initial structure and uses a hierarchical split/merge training to learn a PCFG. Such strategy allows the method to learn very dense but accurate grammars [10]. As an example of how the splitting-and-merging training works, the figure below shows how the determiner tag develops, and only top three words from each subcategory and their respective probability are displayed in the figure [10].

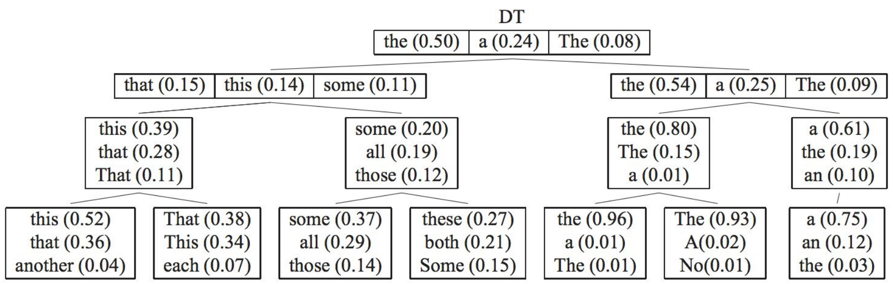
DT
the (0.50) a (0.24) The (0.08)
that (0.15) this (0.14) some (0.11)
the (0.54) a (0.25) The (0.09)
this (0.39) that (0.28) That (0.11)
some (0.20) all (0.19) those (0.12)
the (0.80) The (0.15) a (0.01)
a (0.61) the (0.19) an (0.10)
this (0.52) that (0.36) another (0.04)
That (0.38) This (0.34) each (0.07)
some (0.37) all (0.29) those (0.14)
these (0.27) both (0.21) Some (0.15)
the (0.96) a (0.01) The (0.01)
The (0.93) A(0.02) No(0.01)
a (0.75) an (0.12) the (0.03)

Figure 1: Evolution of the determiner tag during the splitting/merging training.

Due to its superiority in terms of learned grammar accuracy and length, we chose to incorporate this parser in our project for generating random sentences.

Methods

Here are the steps we took to generate random sentences:

1. Extract grammar files from Berkeley Parser
   By using the source code of Berkeley Parser hosted on GitHub(https://github.com/slavpetrov/berkeleyparser), we extracted two grammar files: out.txt.grammar and out.txt.lexicon.
2. Clean up grammar files and read into rule dictionary
   The out.txt.grammar file is inspected and cleaned up. Some rules, like the nonterminal goes to itself with probability 1, is removed.
3. Start from nonterminal "ROOT_0" and randomly choose generation rule

A recursive function randomly choose production based on probabilities of the rules is called on the root nonterminal “ROOT_0”. The recursive function terminates when a terminal symbol is generated. A sentence is formed by combining all the generated terminal symbols.

Results

Example output:

Some of the sentences are really correct:

The company was priced at $ 74.8 billion .

the company `` did n't continue to be covered by its demonstrations .

Some are not so good, but resembles human languages:

the stock show -- originally , had likened his farm , against a handful that falls a closely passed infringement .

May 18 , 1990 , which asked the U.S. in Switzerland ahead , rose the New Street francs 86 or says American Fitzwilliam , the Bank under Violetta , S&P , , , rather from Next to 40 .

Any alternative % in New Hampshire started on property thrift RATE .

Mr. Prohibition said Mr. Berthold may stop about 13 % before the meeting from financial services about its DEC Inc. show for the past 30 years .

It also founded the proposal last month sticking performance , when it considered for efficient instance for a market

Improvements

The code needs to be redesigned so that users could interact with the program. Now it is only able to generate 5 sentences and terminate.

The code could also be restructured into a class so that it is cleaner and more reusable.

Other methods of random sentence generation could be inspected and compared against this method.

The generation takes a long time and we could think about ways to speed things up.

[1]: http://www.ling.helsinki.fi/kit/2004k/ctl272/Bangor/clbook_43.html

[2] M. Collins. 1999. Head-Driven Statistical Models for Natural Language Parsing. Ph.D. thesis, U. of Pennsylvania

[3] E. Charniak. 2000. A maximum–entropy–inspired parser. In NAACL ’00, p. 132–139.

[4] E. Charniak and M. Johnson. 2005. Coarse-to-fine n-bestparsing and maxent discriminative reranking. In ACL’05, p. 173–180

[5]: Taylor L. Booth and Richard A. Thomson. 1973. Applying probability measures to abstract languages. IEEE Transactions on Computers, C-22:442–450.

[6]: James K. Baker. 1979. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, Speech Communication Papers for the 97th Meeting of the Acoustical Society of America, pages 547–550.
[7]: Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In Proceedings of the 41st Meeting of the Association for Computational Linguistics.
[8] https://courses.cs.washington.edu/courses/cse590a/09wi/pcfg.pdf
[9] https://github.com/slavpetrov/berkeleyparser
[10] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In COLING-ACL ’06, pages 443–440.