Author Emulator Final Project

Phuong Dinh and Julia Kroll
CS 322: NLP, Fall 2015

Our goal was to generate original sentences in the style of an author. We were inspired by the textbook's demonstration of generating sentences in the style of Shakespeare (p. 200). Jurafsky illustrates the benefits and challenges in using different n-gram models to solve this problem. Using a bigram model allows for a wide array of new sentence constructions; however, it offers little grammatical structure. Using a trigram or quadrigram model produces sentences that are much more syntactically meaningful, but since the training set of Shakespeare's collected works only includes 31,534 unique words, the trigram and quadrigram models are sparse, and end up generating sentences similar or even identical to Shakespeare's own writing. We hoped to preserve a high level of originality by using a bigram model to generate new sentences, but incorporate a parser into the algorithm in order to produce sentences that were grammatical as well as novel.

A bigram language model generates sentences like the following:

We 're , alarm clock , ' ' the

Five year old man said . ' '

Go straight up and the old man

While Jurafsky illustrated the problem using Shakespeare, we chose to model Ernest Hemingway instead. We were concerned about a contemporary parser's success in parsing Shakespeare's older form of Early Modern English. We considered using Jane Austen, but she tended to write long sentences of about 50-60 words, and linear increases in sentence length correspond to exponential increases in parsing complexity. We chose Hemingway because he wrote in contemporary 20th century English, and his sentences tended to be less than 20 words long.

Once we had decided on an author, we needed to prepare a corpus for training. We chose *The Old Man and the Sea*, one of Hemingway's most famous novels. This work, published in 1952, contains about 1900 sentences and 2600 unique words. We

cleaned the corpus by removing characters that would complicate tokenization and reduce bigram model accuracy, including quotation marks (“ ‘), some punctuation (: ; ,), and underscores (_). We also planned to remove chapter headings (“Chapter 1”), but they were not present in this particular book. We made all text lowercase, again to improve bigram accuracy. We used Natural Language Toolkit’s (NLTK) tokenize_sentences method to tokenize the entire text into sentences, and its word_tokenize method to tokenize sentences into words. We used the entire corpus for training, as there was no need to reserve part of the corpus for testing.

Our algorithm consists of several steps to train a language model and then generate sentences using that model. To train our model, we iterated through each sentence in the corpus. We used the Stanford Parser module in NLTK to parse each sentence. This provided us with the tree structure of the most likely parse, and then we traversed the tree to count how many times Hemingway used each grammar rule (e.g. S -> NP VP). In order to exclude rules that would lead to producing ungrammatical sentences, we did not count rules that included the FRAG (sentence fragment) tag. We then used the tree structure to build a bigram model based on sentence headwords, rather than a simple adjacent-word bigram model. To find the headword of each word, we found each terminal in the tree, traversed upward to find the first ancestor with a left child, and then traversed down that left branch to find the headword. We used heuristics developed by inspecting sentences in order to decide which terminal in the branch should be the headword. For instance, if the ancestor was an NP, we searched for a noun terminal (tagged as NN, NNS, NNP, NNPS, or PRP), and if the ancestor was a VP, we searched for a verb terminal (tagged VB, VBD, VBG, VBN, VBP, VBZ). By counting grammar rules and headword-word pairs, we constructed a probabilistic context-free grammar (PCFG) and a headword bigram model. Finally, we gathered emission probabilities by counting how many times each nonterminal tag led to a terminal word (e.g. NN -> “fish”).

With our model constructed, we were ready to generate new sentences. Our program takes in a ‘seed’ phrase consisting of at least one word to begin generating a

sentence. It expands upon that seed by iteratively generating subsequent words. We use the Earley parser with our PCFG to see which rules may be satisfied by the next word. We get all possible terminals that would appropriately satisfy one of the rules in the current column. Then we calculate the probability of each possible word occurring, by adding the log probabilities of the grammar rule occurring, the word being emitted by the tag, and the bigram occurring. We identify the five most probable words, and select a word randomly from that pool of choices. By choosing randomly from a small number of the most probable words, we aimed to choose probable words while maintaining an element of randomness and originality. One additional condition we add is that the same word cannot appear twice in a row, to increase sentence grammaticality and semantic coherency. We continued generating words until the Earley parser recognized the sentence as grammatical, a period was generated, or there were no possible next words. We also chose a minimum length for each generated sentence, which we randomly selected between 5 and 7 words long. We hoped to prevent very short, ungrammatical sentences from being generated. Once we had stopped generating new words, we then checked whether the Earley parser found the sentence grammatical. If it was found to be grammatical, we made that sentence our final output. However, if it was found to be ungrammatical, we generated a new sentence and tried again. Our program makes up to five attempts to produce a grammatical sentence, since we select the five most probable words to generate each next word.

We encountered several problems in developing our algorithm. The first was producing a rule set for our PCFG. We created our own descriptive rule set by using Hemingway's own sentence constructions. However, some of his sentences were not grammatical, which led to our grammar containing ungrammatical rules. Hemingway's frequent use of dialogue was a main contributor to this problem. For example, eight times he writes the sentence "No." As a result of these ungrammatical sentences, our Earley parser allows constructions that are not normatively grammatical. One solution would be to use a different parser to verify whether an Earley-generated sentence was grammatical. However, the purpose of statistical parsers is not to verify whether or not a

sentence is grammatical, but rather to produce the most grammatical parses. Therefore, having difficulty in discerning grammatical versus ungrammatical constructions is a weakness of all statistical parsers, not only ours. One potential solution to better recognize grammatical sentences would be to test our generated sentences against a normative English language grammar, rather than a descriptive grammar of Hemingway's writing. Nevertheless, this would still be problematic, as natural languages are not context-free, so we could not accurately identify whether a sentence was grammatical or not, even using a grammar that claimed to represent the English language.

Another difficulty we faced was finding headwords in generated partial sentences. The Earley parser was advantageous in that it did not need to construct a tree to suggest subsequent words, but this was a weakness because we needed a tree to identify headwords. Our acknowledgement of this problem was to use headwords to construct the bigram model in the training set, but to use adjacent-word bigrams in generating sentences. While our program would have been more accurate if we had used headword bigrams in sentence generation, we still benefitted from using headwords to calculate probabilities in our bigram model. Using headwords in training helped us to create a bigram model that contained more syntactic information, so when we generated our relatively short sentences, we were more likely to produce grammatical constructions.

The nature of our project does not provide an obvious way to quantify success. However, we can compare sentences generated using our earlier, unsophisticated model with our later, more sophisticated one in order to gauge our improvement. We also made improvements to increase the speed of our program. One basic step we took was to ensure we were not duplicating rules in our grammar, as initial oversights that allowed duplicated rules greatly increased speed, on the order of seconds per word. Another way we achieved faster word generation was by using a smaller corpus, which produced a smaller rule set for our PCFG. For instance, training on 100 sentences

allowed us to generate each word in tenths of a second, but training on 1000 sentences slowed our program to several seconds per word.

Seed word	Language model trained on a 250-word corpus
we	we would like to take back
the boy	the boy I'll get him
four	four and he had and when i 'll take
they	they have the old and of the boy
five	five which is what about baseball when the man said
you	you are you to the old man
how	how you were the old and he the man
go	go for him .
i	i must get the old man
but	but the man the older when he was when i have not ?

A sample of sentences generated by our program.

We see two primary improvements that would enhance our program. The first is incorporating headwords into word prediction, so that our sentence generator uses headword bigrams to match the model we develop by training on headword bigrams. The second is to implement a way to more accurately check whether our generated sentences are grammatical, so we can continue to develop new sentences until we finally generate a grammatical one that can be presented as a final output. Although we recognize possibilities for improvement, we believe that we have achieved our goal of incorporating a parser into a bigram language model in order to generate sentences that are more grammatical than a simple bigram model, while still allowing for creative, original phrases that are of primary interest in attempting to produce new sentences in an author's personal style.

References

Publications

Michael John Collins, “A New Statistical Parser Based on Bigram Lexical Dependencies” (1996), http://dl.acm.org/citation.cfm?id=981888.

Shinsuke Mori, et al., “A Stochastic Parser Based on a Structural Word Prediction Model” (2000), http://dl.acm.org/citation.cfm?id=990820.990901.

Ciprian Chelba and Frederick Jelinek, “Exploiting Syntactic Structure for Language Modeling” (1998), http://dl.acm.org/citation.cfm?id=980882.

Libraries

Natural Language Toolkit (NLTK), http://www.nltk.org/.

Stanford Parser, http://nlp.stanford.edu/software/lex-parser.shtml.

Indiana University Earley Parser [we referenced and built upon this resource], http://www.cavar.me/damir/charty/python/.

Corpus

Ernest Hemingway, *The Old Man and the Sea* (1952), http://www.gutenberg.ca/ebooks/hemingwaye-oldmanandthesea/hemingwaye-oldmanandthesea-00-t.txt.