Artificial Intelligence – Stochastic Language Generator

Jacob Ochs
Westmont College
jochs@westmont.edu

INTRODUCTION

The LanguageGenerator java class attempts to create strings of words which are pseudo-randomly generated to become syntactically readable. Some of the concepts behind the code was worked on in joint collaboration with Jonathan Skidanov, Chris Betsill, and Sam Bentz. For this project deliverable, the LanguageGenerator program will be able to process any number of text files of any length, so long as they are all .txt files and are grouped together in the same folder file path. For future reference, we will call this initial input text the corpus. In addition, the program will be able to work with any number of n-gram language modeling (more on this later), and will be able to generate any number of concatenated word Strings as a result. However, runtime may become a detrimental factor as the size of the text files increases. The primary focus and method we will use for generating a readable resulting text is the generation of n-gram models, which we will then use to represent the sequences of word appearances within the text corpus. The end goal of this project is to generate a String of words which are readable and syntactically correct, although they will most likely be semantically incorrect. In order to achieve semantic correctness, we will need to implement more advances methods to filter the n-grams, which was not a required part of this deliverable.

APPROACH

Setup and Corpus Analysis

In contemplating and approaching this project, I first looked at recent examples of randomly generated papers which have been accepted into peer-reviewed articles, just to get a sense of an overall goal and the potential results which could be produced from such a project. Next, I researched the concept of n-gram language modeling and how this can translate to a result of syntactically correct language generation. I then briefly examining the corpus texts given in the project specifications, including War and Peace and Crime and Punishment by Dostoevsky, The Complete Works of Jane Austin, and the Bible. From observing the corpus, I was able to get a feel for what kinds of patterns to look for when generating the n-grams, and also noticed the uses of punctuation and capitalization throughout the corpus. After having done all this prior research, I processed the corpus text in the program constructor.

N-Gram Modeling and Text Generation

For our main method of language generation, the n-gram language model was implemented. The method iterated through the entire corpus, combining all words together in tuples of n, and adding these tuples to a TreeMap<String, Integer>. Then, as the iteration continues, the n-gram method will subsequently increment an integer counter in each Map entry when a duplicate tuple was to be found in the corpus text. Typically, values for n between 2-5 produce the best results.

Once our n-gram models had been generated into the populated TreeMap, each entry corresponding to the String phrase and the number of occurrences in the corpus, we generated text by iterating through the n-grams and randomly choosing one which begins with a capital letter (as this is most likely either a name or the beginning of a sentence). Once we have our first word, we search for all of the n-grams with this word in the beginning, then subsequently choosing a random one of these based on a weighted scale of the number of occurrences. Once the adjoining phrase is chosen, it's last word becomes the new first word and the process repeats itself. On each run through the generation of words, a count of the resulting number of words is kept so as not to surpass the number of desired words.

EXPERIMENTS

Throughout the early phases of development, I only processed a small portion of Crime and Punishment as my corpus. This proved to be a useful strategy for testing and debugging at first, but led to later problems with runtime and punctuation and capitalization use as the development phase progressed. I then began testing the program on this small corpus, plus a larger corpus and finally the entire corpus in order to achieve all possible ranges of results for the various corpuses. In addition, I experimented by implement a completely random text generator in order to establish a benchmark standard to compare to my n-gram generated text. Although this could be greatly improved upon with the implementation of a method to correct specific errors such as punctuation and capitalization, the n-gram modeling does work for any and all values of n.

RESULTS

My results at the present time are mixed. The n-gram language modeling works to some degree, but it is obvious that further methods of improvement are needed in order to generate a text that is truly syntactically correct. The proper recursive chaining result, from the method to choose words based on the n-gram models, is not completely correct at present and needs revision/debugging. In addition, after reading about some methods of improving the linguistic quality given in the text (Russell & Norvig, see References), I have concluded that a more comprehensive method of dealing with punctuation and capitalization in processing the corpus would yield more improved results.

CONCLUSION

This project deliverable goes to show the extent with which he can generate syntactically correct strings of words which are most likely semantically incorrect. Using only these "naïve" methods such as the n-gram language modeling, we can get a result of relatively foolproof syntax. However, this is not a finished product, as there is still work to be done in completing and improving upon my current project status, as well as implementing new methods to filter the text and improve overall quality of result.

REFERENCES

[1] Russell, Stuart and Norvig, Peter. 2002. *Artificial Intelligence – A Modern Approach.* Ch. 22-23, Linguistics. Pp. 790-862