Experiment with Interactive Sentence Prediction

Renae Fisher, Zachary Rowton, Christian Westbrook
Computer and Information Sciences Department
University of Arkansas – Fort Smith

October 4, 2018

Abstract

The purpose of this experiment is to test the capability of an interactive natural language processing system to complete partial sentences based on a probabilistic analysis of natural language data and with the use of n-gram language models. The system design consists of a preprocessor that tokenizes and tabulates training data into n-grams and stores them to disk, and a runtime application that takes in the n-grams and user audio input and predicts the next ten words of the given sentence. We successfully implemented an interactive sentence predictor based on probabilistic analysis and n-gram language models. We found that the size, style, and quality of the training data heavily impacted the generated sentences, and that trade-offs exists between the correctness of the generated sentences, the speed of generating predictions, and the domain of natural language represented by the system. Further experimentation with our system design would involve representing the same language models with different data structures in an effort to improve the time complexity of generating predictions.

1 Introduction

The purpose of this experiment is to test the capability of an interactive natural language processing system to complete partial sentences based on a probabilistic analysis of natural language training data and with the use of n-gram language models.

2 Review of Literature

An introduction to text normalization and a survey of modern natural language processing algorithms and techniques are discussed in Speech and Language Processing, by Daniel Jurafsky of Stanford University and James H. Martin of the University of Colorado at Boulder. This text is the source from which we derive our representation of language through N-grams and of their probabilistic analysis.

3 Approach

The system design consists of two subsystems: a preprocessor and a runtime application. The purpose of the preprocessor is to tokenize and tabulate natural language training data and to facilitate the storage of this information to disk. The runtime application consists of both a sentence predictor and a user interface, which itself can be divided into a graphical user interface and an audio input device.

The language models used for probabilistic analysis are generated by the preprocessor, and are based on input natural language training data. The models are tables of unigrams, bigrams, and

trigrams generated from tokenized words derived from the training data. These tables, along with a series of computed metrics, are generated and stored to disk to be used by the runtime application.

The runtime application presents to the user a graphical user interface and an audio input device. The user can speak an incomplete sentence into the audio input device to be used as input to the sentence predictor. A speech to text library is used to convert the input audio to text that can be used by the sentence predictor. The sentence predictor generates the next ten words of the sentence based on both the input text and a probabilistic analysis of the likelihood that any particular word would follow the given phrase. In fact two sentences are generated: one using bigram probabilities and the bigram language model, and another using trigram probabilities and the trigram language model. The total probability of each sentence occurring is also computed. The input audio is displayed as text on the graphical user interface along with the generated sentences and their computed probabilities.

4 Implementation

The tokenizer for the preprocessor was implemented with the JavaCC Java Parser Generator. The tokens are processed by an application written in Java that generates the N-gram tables and computes relevant metrics and stores these to disk. A shell script is included with the preprocessor that facilitates the sequential execution of both the parser and builder application.

The runtime application was implemented as a Java application containing both a sentence predictor and a GUI. The input audio is read with a connected microphone and processed by Voce, an open source speech synthesis and recognition library. The language models are represented as hash tables mapping unigrams, bigrams, and trigrams to their respective frequencies. The sentence predictor uses these frequencies, along with other metrics (such as the total number of unique unigrams stored in the unigram table, or the total count of tokenized words) to calculate bigram and trigram probabilities. These probabilities are used to generate two different sentences, each displaying the most likely next ten words of the input sentence. The recognized speech, the the two generated sentences, and the computed probability that either of these sentences would occur are displayed in the GUI. Two shell scripts are included with the runtime application: one for compiling and one for execution.

5 Results

We successfully implemented an interactive sentence predictor through the use of probabilistic analysis and language modeling. Our sentence predictor successfully generates two sentences based on both the recognized speech input and the given training data. The chance that our system generated an understandable sentence largely depended on the training data. With smaller data sets the number of unique sentences that were able to be generated was much lower than with larger data sets, but the system was able to generate sentences much quicker. With larger data sets sentence generation took longer, but the number of unique sentences able to be generated increased dramatically.

A strange result we didn't apprehend was that as the size of the training data increased the most probable sentences became slightly repetitive. Choosing a word with a lower probability of being next resulted in a very high number of unique and understandable sentences. We also found that if a singular tone or writing style dominated the training data, that style would be maintained within the generated sentences. For example, when operating on a data set consisting only of the complete works of William Shakespeare, the generated sentences maintained the input Old English style.

6 Conclusions

The performance of any natural language processing system will largely depend on the input natural language data and the language models that support natural language processes. Trade-offs exist within our system between the correctness of the generated sentences, the speed of prediction, and the domain of natural language represented by the system.

7 Future Work

Further experimentation with system design could improve the speed or accuracy of this system. In particular we envisioned using a different data structure to represent the language model. In this system we used hash tables to store unigrams, bigrams, and trigrams mapped to their frequencies. Assuming a hash table isn't bound by problematic collision handling, the time complexity of looking up a key-value pair is $O(n)$, and of inserting an entry into the table is $O(1)$. The problem is that insertions are performed in a batch preprocessing step that is able to use extra time if necessary while the lookup operation is used repeatedly in probability calculations by the sentence predictor as it is executing. A faster lookup operation would improve the speed at which the system could generate sentences.

A suggested future experiment would be to represent these language models as classes containing fields for the N-gram and frequency stored within a red-black tree, sorted by the numeric values of the characters in each n-gram. While the insertion operation's time complexity would increase from $O(1)$ to $O(lgn)$, the lookup operation would lower from $O(n)$ to $O(lgn)$. The insertions being performed during preprocessing would be able to execute in as much time as needed given a tractable data set, and the lookup operations performed during runtime would execute faster and potentially reduce the noticeable latency between speaking a sentence and receiving the generated sentences.