Final Project: **Trump Twitter Generator**

Ahmad Elhamad, Ashwin Chirag, David Lane, Henry Curtis, Tiger-Anne Wing Lawrence

CS 582: Intro to Speech Processing

Professor Chuck Konopka

1 May 2017

**Introduction**

**The Idea:**

We decided to follow in the footsteps of the Shakespeare Synthesizer Project and replicate the same idea, but with a modern twist. Our project instead uses Donald Trump as the “input” for the program and outputs the recreated text into a different format. The format we wanted to emulate was Twitter, to simulate as though these sentences were straight from Donald Trump’s Twitter Account. The main premise of the project revolves around how we can find commonality in someone’s speech, and present this in a social media setting. The concept of the project follows N-Grams, and primarily using a bigram to fit within Twitter’s limitations, and using the pair of words to generate the next most likely pair to follow, based on the last word in the bigram. With a person like Donald Trump, we figured this would be a good example considering the massive amount of data we have from his previous Tweets since 2009, and the fact that he is quite particular in his speech.

**Background**

**Context:**

We wanted to utilize the ideas from the Shakespeare Synthesizer with someone who we thought would be a good fit for the project. Additionally, we wanted this to be an experiment in how we could recreate someone’s writing style, whether that be in literature, social media, or everyday conversation. With someone like Donald Trump, we wanted to see what his speech commonality would be if we ran it through this modified program.

**Current State:**

Currently, n-gram models are used across many fields of linguistic research and business. A major contributor to the use of n-gram models for speech analysis is Google, who have used their large data centers to create n-gram models for a few billion words. They currently use their n-gram models to provide accurate spelling correction, language translation and speech recognition. N-grams have been used for data compression purposes, as well as used to model genetic sequence data to find relational patterns between DNA. As n-gram models can be quite versatile, there is a lot of room for improvement, and further uses left untapped.

**Basis:**

We are basing our idea for a Tweet Generator on a previous project which uses the same concepts, but with the Completed Works of Shakespeare. The previous project had some minor revisions that needed to be implemented, such as annotations, punctuation changes, and removal of redundant code, but overall the older project proved functional for generating the Tweets. We wanted to test how well this code would work for someone as influential and distinct in their speech as Donald Trump, but the same method could be applied to anyone else who has a specific style in the way they project themselves on an obtainable platform, such as social media, literature, or any other documented speech. Our project was also very similar conceptually to what the previous project accomplished by incorporating the idea of N-Grams and using bigrams; trigrams were also implemented in the code, but they are never actually referenced in the generation of sentences. We chose to continue with this method as well primarily because the larger N-Gram would cause a longer sentence, and cause possible character limits to be reached faster, making shorter sentences, or no sentences at all.

**Body**

**Implementation:**

The Synthesizer is broken into two different parts; the Python Files are the retrieval and formatting of the data section of Synthesizing, and the Java Files are the Parsing and Hashing section of the project where. The main premise to the Java Files, and conceptually what the project is doing, is creating N-Gram of words in order to access most frequently used words in a dataset. The idea behind this is that a sentence would have a certain word in it based on what follows and what comes before it, similar to a Hidden Markov Model and basing predictions off of previous experiences. The Python Files first start by accessing a large data set of Trump Tweets since 2009, which it then formats into the desired style the Java Files can use to parse through. Tweetsjson.py is the first of the files, starting by formating the archive we acquired of Trump Tweets since 2009 with the starting pivots in a text document. Then there is the twitter\_streaming.py that formats the text into a workable data set with the appropriate reductions made to compensate for ReTweets, Quotes, Links, and self references. The formatting of the data is done by the Python files in order to make the Java files generate cohesive sentences that are not redundant or confusing.

The Java Files are the main bulk of the program, producing the many Hash Tables that will be accessed later to reproduce random sentences. The first Java File (Word\_Synth.java) is necessary for creating the frequency bins for the other Java File to access, which will pick the most likely word to add to the current sentence/Tweet. Word\_Synth.java takes in the Python File output and begins to divide the words into five different categories: most frequent words overall, pair of two words moving forward, pair of words going backwards, tuple moving forward, tuple moving backwards. Each category is split accordingly and then loaded into the correct Hash Table along with its corresponding Frequency Number, which will be used when deciding the most likely word to choose. Once all the words have been trimmed and split into their bins, the second part of the Java Files (Main.java) can be used. Main.java will use the Hash Tables from Word\_Synth.java to select a singular word at random, and then follow that word with the double word going forward until it hits an end point. The sentence structure is only loosely modified in order to preserve Trump’s overall syntax; minor changes in punctuation are added in order to make the sentences appear to be actual sentences.

**Experiment:**

The overall experiment is split amongst different types of files, starting with the Python Files that access the Trump Tweet Database, and then proceeding to the Java Files that then parse and reconstruct the parsed words. We had to run through a couple tests with the Tweets and the Java Files in order to create sentences that would be recognizable and somewhat coherent. Moreover, we needed to account for a different dataset than the Shakespeare Texts that were used originally; we needed to remove special characters and syntactical elements that are within Twitter's messages, or within the underlying code. However, we were able to create several Tweets that were similar to something Trump would Tweet from his official account.

**Data:**

The data we collected was from an archive found on a GitHub account, dating back from 2009. We collected this data from the archive and then processed it through the Python Files so that it could be read in by the Java Files for parsing and loading. The archives were also stripped of certain characteristics so that the data was more coherent and reasonable based on Twitter’s standards. For example, we eliminated Tweets with “@realDonaldTrump” in them, ReTweets, Links in the Tweets, Quoted Tweets, and other minor changes to keep the text readable and less cluttered with Twitter’s fluff. The data needed to be trimmed down in order to reduce the number of “@name”s that were found when we initially ran the programs; it seems that Trump’s Twitter made many mentions to himself, or tweets were being called from other people referencing Trump’s Twitter account.

**Results:**

The results of the large data set and the programs working together was a moderate success. We have sentences being generated based on a pair of words, with the second word being the most likely to follow the first. Additionally, the sentences generated include some of the iconic statements and structure found in Trump’s speech and writing, with some overhead of Twitter implementation and writing. An interesting note about the generator and the speech construction is the addition of social media influence and how different people will communicate based on the platform they are using. For instance, our first run of the generator picked up many “@theRealDonaldTrump” within a single sentence, based solely on the fact that these are the most common elements of Trump’s Tweets, whether it be a full conversation or a simple “Thanks” to the directed person. The generator gave us many insights into how recreating speech in a social media setting can prove challenging because of the added layer of the platform’s social norms. Overall, the results were desirable, and we had success in creating some genuine Tweets that may be passable if the observer did not know the Tweets were synthetic.

**Conclusion**

**Summary:**

The generator was able to recreate sentences based on the data we found of Trump’s Tweets since 2009, a very large data set that helped bring some more commonality into Trump’s speech. The process of creating N-Grams of words, mainly double-pair of words, and then generating sentences based on Twitter’s character limit proved to have some success, but still resulted in some sentences being nonsense, or cluttered with too many “@name”s. We needed to modify the Tweets so that they were more manageable and coherent, resulting in a loss of dimensionality as we extracted features of Trump’s Tweets. Overall, the experiment was an insightful, and humorous, look into recreating a person’s speech, based solely on their messages from a social media platform, and also with consideration of what the most frequently words they use are. It would be interesting to see this be improved or simulated with different people or platforms, to assess how a person’s conversation changes based on what platform they are projecting their voice with.

**Improvements:**

A major improvement to this project would be to account for the wide variety of retweets, hashtags, and directed messages that make up a major component of Twitter messaging. One problem we encountered, and still encounter in the generator, is the amount of “@theRealDonald” or other names included in a single sentence, which is not entirely far-fetched, but still an oddity for every message. We have had thoughts to reduce these by limiting the amount to only one or two, but then we found ourselves in a situation of constricting a person’s possible speech pattern by formatting it to be “nicer” for us. We decided against implementing this feature based on the fact that we did not want to manipulate the data too much so that it would no longer be Trump’s style, but rather a bastardized version of it.

We would also like to see improvements made to the Java code in regards to how the words are being accessed. We spent too much of our time just trying to get the files better formatted, both for our accessibility and for the Twitter accommodations, that we could not further delve into the conceptual development. It seems the Main.java file only utilizes the double forward words (BiFwords.ser) rather than the full spectrum of words and tables. However, this creates a problem of which N-Gram is better to use, and when to use them, whether at random or in a specific order. We have stuck to the BiFwords.ser file only because it is the best middle ground from the other types of files, but it would be an interesting study of utilizing each version and seeing which worked best. It seems that with a higher amount of N-Grams, say a triple or quadruple pair, that the context of Trump’s speech would be better preserved and not scattered in thought, but with Twitter’s limitations, we cannot consider this because Tweets could be possible cut or too long for Twitter to handle.