MCMC Random Walk Trump Tweet Generation

Dominic Carrano, Mia Mirkovic, Keyan Nasseri, Ashvin Nihalani

EECS 126 Project - Fall 2018

1 Introduction

The purpose of this project was to investigate the applications of Markov Chains in language; specifically, to examine the extent to which they can be used to generate sentences, in particular Tweets. To give a humorous angle to the project, we decided to use President Donald Trump's tweets, known for their brashness and lack of tact, as the data set. The idea was to use these tweets to generate a Markov chain characterizing the way President Trump speaks in general. The states of this chain are words that have appeared in President Trump's tweets, while the transition probabilities represent the empirical probability of a particular word following another word in one of the tweets. We then randomly walk on this chain to get randomly generated Trump tweets which can mimic the way that President Trump speaks/tweets.

2 Methods

2.1 Theory

The theory behind the generation of our tweet Markov Chain was maximum likelihood estimation, specifically to use the observations from Twitter to construct a Markov chain of maximum likelihood representing the probabilities of transitions between words. We decided on MLE estimation due to its relative simplicity as a form of chain construction, and also because we found it difficult to conceptualize what a prior distribution for a MAP would look like. To do this we needed to derive the MLE for Markov chains.

We do this by maximizing the probability of a particular sequence X_1^n of a Markov chain, which is given by

$$L(p) = Pr(X_1^n = x_1^n) = Pr(X_1 = x_1)\prod_{i=2}^{n} Pr(X_i = x_i | X_{i-1} = x_{i-1})$$

This can be simplified by introducing the variables n_{ij}, where n_{ij} represents an observed transition from state i to state j; this gives the likelihood

$$L(p) = Pr(X_1^n = x_1^n)\prod_{i=1}^{k}\prod_{j=1}^{k} p_{ij}^{n_{ij}}$$

Taking the log-likelihood, we get to the expression

$$L(p) = \log(Pr(X_1 = x_1)) + \sum_{i,j} n_{ij} \log(p_{ij})$$

By applying the constraint that

$$\sum_j p_{ij} = 1$$

and eliminating parameters we arrive at the MLE for Markov chain transition probabilities p_{ij}, given by

$$\hat{p_{ij}} = \frac{n_{ij}}{\sum_{s\in\Sigma} n_{is}}$$

where Σ represents our overall state space.

As a result, to generate the Markov Chain of maximum likelihood given the observed tweets, we needed to compute this quantity (essentially requiring us to compute the n_{ij}) for each observed transition, where our states are words and we define a state transition as a word which follows another word.

2.2 Pseudocode

In terms of code, there were several distinct steps to getting the tweet generator working. The first step was to use the Twitter API to pull Trump's tweets from the internet and construct a dataframe of them, which we did with the help of the tweepy python package. The functions for this include 'load-keys' and 'download-recent-tweets-by-user'. Some of this code is borrowed from a Data 100 project which also involved doing other forms of analysis on Trump tweets, although the only code we used from them was to actually grab the Trump tweets for the dataset; all the processing was added by us. Next we had to do some additional string processing on the tweets, which included getting rid of all links in the tweets, getting rid of newline characters, and getting rid of ampersands.

Then, we had to collect all the unique words to form the states of the Markov Chain. The chain itself is stored in a dictionary of dictionaries, 'Markov-dict' this dictionary essentially maps each unique word i to another dictionary; this inner dictionary maps every word that j that followed i at some point in Trump's tweets to the MLE transition probability $\hat{p_{ij}} = \frac{v_{ij}}{\sum_{s\in\Sigma} v_{is}}$ discussed above. We computed these probabilities using the function 'generate-tprobs-for-word', which essentially finds all the occurrences of a word throughout the tweets, where the word is not the last word in the tweet, and records the corresponding transition. This brought us to 4819 distinct words.

We then wrote a function to generate a random tweet given a start word w and a specified length l; in other words, a function which starts at state w and randomly walks on the Markov chain defined above, recording its path and building a string with this states until it has made l transitions. This function is 'gen-tweet-given-start'. This function not only outputs the randomly generated tweet, but also records the cumulative probability of the randomly generated tweet by multiplying the transition probabilities at each transition and outputs this probability in a tuple with the tweet itself, the reason for which we will mention later.

3 Experiments

To actually test out the random tweet generator, we did the following many times, with various start words: generate 100 random tweets with a particular start word, sort them based on their cumulative probabilities in descending order, then examine the top 10 most probable random tweets and look for any that seemed to mimic trump tweets or that were particularly humorous. We tried various tweet lengths, ranging from 4 words to 30 words; we found that we got the best result by keeping the tweet length between 5 and 7 words, as tweets that are too long had a lower probability of being coherent as they have more randomness involved. Some of the start words we tried out included "Cruz", "Rubio", "Obama", "Hillary", "Jeb", "Bush", "I", "ISIS" and "Marco".

4 Results and Analysis

We did manage to get some fairly accurate (in terms of similarity to what we expected based on actual tweets we've seen from President Trump) results, including "I beat him and will #MakeAmericaGreatAgain!" and "Trump speaks. #TrumpTrain #Trump2016 #MakeAmericaGreatAgain #Trump2016". In addition, the general flow/syntax of the tweets generated seemed to be fairly consistent with We compiled a list of our favorite tweets that were generated, although our all time favorite, not shown below, is: "Jeb Bush has a Vatican City is desperately".

```
best = ['Ted Cruz lies and he runs ',
        'Ted Cruz campaign is very sleazy ',
        'Obama is working with Trump. Thank you people ',
        'Jeb Bush has ties to Trump and Cruz ',
        "Bush just on @FoxNews - I wasn't enough ",
        "Marco 'Amnesty' Rubio is thinking of little Mort ",
        'Marco Rubio was as if he is weak ',
        'Marco Rubio is jealous of Cruz & strong ',
        'Trump has a dinner in bed ',
        'Democrats numbers have done the dying ',
        'I did a WALL and ObamaCare, ',
        'I could be just another politician. ',
        'Trump won all the evangelical vote ',
        'ISIS is a person in Salem, ',
        'ISIS is better than a rock! ',
        "I'm going to church I WILL. ",
        'many believe @realDonaldTrump Wow! Thank you, ',
        'so easy to be president - ']
```

5 Discussion

5.1 Limitations

Our biggest regret regarding this project was how late we started on it - we invested over 70 person-hours into the Digital Communication project, trying an incredible number of schemes based on using Huffman coding for the Source coding method, before ultimately declaring failure and moving to the MCMC project instead. We tried both Hamming and Reed-Solomon channel coders; the former was annoying because it required having data padded to the parity block size. There was a very nice online Python library we found for doing Reed-Solomon encoding that used. The main problems we encountered were in the modulation phase: we went to the trouble of implementing several different schemes in Python including AFSK1200, orthogonal signalling, QAM, MSK. We also played around with a huge number of different schemes in MATLAB/Simulink to no avail.

We did manage to get orthogonal signalling to work in noise-free setting, but failed to achieve noise-resilience and decided not to submit that as a result.

Another limitation we had was raw computational power; due to runtime constraints, we were unable to use a data set of more than 1000 tweets to generate the Markov chain, whereas our original data frame pulled from Twitter contained over 9000 tweets. It is very possible that with more data, we could have obtained more accurate transition probabilities as well as more unique words, ans and as a result more accurate and varied tweets.

5.2 Future Work

In terms of ways to improve the random tweet generation, one particular change could be incorporating more NLP techniques to take into effect the fact that word choice language often depends on more than just the previous word and is thus not a pure Markov process.

Further, to address to runtime limitation addressed above, services like AWS or some kind of distributed framework could easily be utilized to build the Markov chain using a data set of 9000 tweets or perhaps even more, and potential increases in accuracy could thus be explored.

There are several related features that could be implemented as add-ons to this project beyond what was mentioned above, including:

- Return a probability score that any tweet (generated or not) was actually something that President Trump tweeted.
- Devise a normalized 0-100 metric for comparing how similar two tweets are, and use this to compare a generated tweet against ALL past real Trump tweets, and return the Tweet that a generated Tweet is most similar to, as well as its similarity score. This would likely involve delving fairly deep into NLP theory and we didn't have the time to explore it as much as we would have liked, but it'd be a very cool future extension to explore.
- Analyze the evolution of Trump's tweets over time, and estimate the approximate time (year, or month and year) that Trump would have been most likely to make the tweet.