Evaluation of Language Models Trained on Tweet Data

Zachary Bebbington zbebbington@umass.edu
Quan Bui qbui@umass.edu
Frick Shao fshao@umass.edu

1 Problem statement

The Goal of our project is to evaluate a number of generative models as well as improve one of those generative models. To elaborate on the first section, we have decided to train 7 generative models on the data that we obtained from four twitter accounts. Some of the generate models that we utilized were fully implemented by users on github, such as textgenrnn, and all that was required of us was to provide it our data sets. However, other models that we discovered we're presented in a demonstrative manner, like on of the LSTMs that we used, at which point we first had to reproduce the code then feed in our data sets. In both cases, every model was supplied all of the four data sets one at a time, so each model was trained 4 times total. Once a model had been trained on a data set we had it produce 100 outputs, the idea is that we would consider each of these text outputs to be tweet that the model attempted to make based on what it learned from the tweet data set. Because we had 7 models and each model would be tasked with producing 400 outputs we ultimately had 2800 output texts. From here we wanted to humanely evaluate all of these outputs, to rank the performance of our models. Initially, our plan was to individually look at the 400 tweets produced by each model and rate them in a binary fashion (1 = it was a believable tweet based on the data set it was trained on, 0 = it was not a good tweet) from there we would average our counts together and rank the models accordingly. But we ultimately realized that this would've been an oversimplification of the task. Primarily because few of the outputs well resembled a tweet, so it would be more of a competition for last than first. It was at this point that we elected to diversify the way that we were going to evaluate the output of our models. We were now going to consider more unique attributes of the data set trained upon when looking at each output. For example, two of the data sets that we used were recent tweets made by the account @BarackObama and @realDonaldTrump we'd want to note the difference is use of hashtags and link, as well as where they appear in these users tweets and see how well each model can mimic these characteristics. We ultimately felt that the most important characteristics to look for were: length of tweets(does it do a good job at covering the different tweet lengths the user ranges from), hashtags(are they present as often as the data sets uses them, and in the same places, are they the same hashtags or new ones), links(are they present as often as the data sets uses them, and in the same places), language(how long does the coherency of the tweet last for, does it actually use language similar to what's used in the data set it was trained on). And with this we had a much better direction to move forward in for this first section of our project.

The second section of our project is going to remain true to our initial design. Once we've completed the evaluation process and ranked our models the plan is to take the highest ranking model and tinker with it, with the ultimate goal of bettering its performance on this task. With that being said our personal goal for this section of the project is to have a more personal interaction with a generative model. In the first section we're simply utilizing models that have already been created. But in this section we'll have the time to dissect the top performing model and better understand the design and algorithms that allowed it to perform so well. From there we can attempt to tweak those aspects on the model and see how it impacts the models output. For example, one general change that could be made to the models

is the loss function that they utilize perhaps changing it to or from cross entropy could lead to an improvement. A more specific example could be changing the structure of a neural network, the Pytorch character rnn that we utilized provided the ability to easily change the number of hidden layers, epochs, and neurons so changes here could also prove to be beneficial. If time permits we may also attempt to work with the model that performed the worst on our task as this could provided us with yet another opportunity to closely interact with a generative model and see what attributes of it can be altered to improve its performance on a specific task. With that we have our plan for the second half of the project, getting our hands dirty and doing what we can to tweak the best, and possibly worst, model to learn how it impacts the text it outputs.

2 Your dataset

Our data sets are the tweets of four different people namely Barack Obama, Donald Trump, Adam Best, and Mindy Robinson where each individual data set has thousands of individual tweets. These are divided into two categories. Barack Obama and Donald Trump have larger data sets with around 2800 tweets, while Adam Best and Mindy Robinson only have around 1500 tweets. Tweets are different from regular text since it is a platform where people share information with other a large amount of other people. Our biggest worries stems from the character limit of tweets. Since there can only be a certain number of characters per tweet, the online culture has evolved in such a way as to adapt to this constriction.

People use abbreviations like b/c for because and b4 for before. Similarly, acronyms such as btw represents by the way and ftw represents for the win. However, if someone used the phrase by the way, the meaning should be the exact same as btw. Therefore, the model should not give significantly different probabilities for the two in a certain context. We expect this to be okay with abbreviations, but potential problems arise with acronyms. Since acronyms will be considered by the models as a single word, it is more likely to occur than a string of words. Therefore, even though btw and by the way have the same connotation, btw will likely be assigned more probability. We cannot directly map these phrases to their acronyms, because not only is it already a large set, but more will be added over time. Therefore, our ideal model will be able to deal with this problem on its own.

The second type of problem that tweets presents are strings that are not words. These strings will be present in our vocabulary as they are still part of the training data. However, if included in the outputs,they will make very little sense.

The most obvious example of this problem are links. The string https://t.co/kovcij4fwu is contained in one of Donald Trump's tweets. In the context of the tweet where it was found, the link makes perfect sense. If someone want to support a certain cause or learn more about that specific topic, then they should click the link. The model though will likely not recognize when is an appropriate time to put a link. It may decide to put a link in the middle of a sentence and as a result disrupt the flow of said sentence. This is obviously not desired unless the person the model is being trained on has a tendency to do so.

Another type of unwanted strings are misspellings. Since our training data is over real tweets, undoubtedly there will be some human error in them as well. However, our output should not contain any misspelled words since it may lead to confusion as to what the tweet actually means. Therefore, our ideal model should be able to identify which words are not actual words during training and not include it into the vocabulary. Currently though, there is no such mechanism and each misspelled word will have a chance to appear in the output tweets.

On a similar note, words in other languages will sometimes appear as well. For example, Barack Obama's tweets are predominantly in English. However, they also contain some Spanish such as in this tweet, usted y su familia merecen la tranquilidad de saber que estn cubiertos. el mercado de seguros abre maana. https://t.co/9ircklrgzd. In cases such as these, we should not discard them as with misspelled words, because the original tweeter intended to use these words. On the other hand, these tweets will end up in the same vocabulary as English words. This means a Spanish word might end up in a sentence

that is otherwise English and vice versa.

3 Baselines

One of the models that we've worked with for this project is the tweet generator package on GitHub created by Olivier Morissette (Morissette, 2018).

Upon running the below code:

```
from tweet_generator import
    tweet_generator
    TPCK = '<public_consumer_key>'
    TSCK = '<secret_consumer_key>'
    TPAK = '<public_access_key>'
    TSAK = '<secret_access_key>'
    twitter_bot = tweet_generator.
        PersonTweeter('user_name',TPCK,
        TSCK,TPAK,TSAK)
```

At this point the model scrubs through all of the tweets that it can access under the restrictions of the Twitter API. With each tweet that it runs over it updates a JSON file that contains a markov chain state model from phrases to phrases. For example, after looking at tweets from the twitter user iheartmindy the JSON file contained some of the following data:

```
"\"But there": ["we"],
"\"Democrats in": ["Georgia"],
"\"For many,": ["this"],
"\"People you": ["may"],
"\"violated state": ["and"]
"#CampFire #CampFireJamesWoods": ["#
    WoolseyFire"],
"#CampFireJamesWoods #WoolseyFire": ["#
    ParadiseFire"],
"#DaysOfMindy Day":
    ["45:","44:","42:","41:"]
```

In this format we can see that the phrase violated state' transitions to only one word and' where as the phrase DaysOfMindy Day' has four transition possibilities 45:', '44:', 42:', and 41:'

From here we can tell the model to perform a walk over these markovian states and from this walk an output will be produced.

```
random_tweet = twitter_bot.
    generate_random_tweet()
    print(random_tweet)
```

Keeping with the iheartmindy account one such tweet that it generated from a walk was:

```
I delete my account the communists
    running Facebook win. I WILL NOT
    BACK DOWN when it comes  t o
```

Another model that we used was a character level RNN created by Sean Robertson that utilized the pytorch library (Robertson, 2017)

First first step here is just like above to train the model on a new data set. To do so we execute the line:

```
python train.py adamcbest_tweets.txt
```

from the terminal. Here the training process occurs over 1000 epochs. In each epoch 100 strings of 200 character are sampled from our training set and with that data the weights within the RNN are updated via a training method that follows the update procedures that we learned about in class regarding gradient ascent. Once the 1000 epochs have run we will be presented with a new .pt file. This file contains all of the weights that were learned from our training data. Because we wanted to see how well the model could perform out of the box all of hyperparameters were kept at their default values.

```
Number of epochs: 1000
Hidden size of GRU: 50
Number of GRU layers: 2
Learning rate: 0.01
Chunk length: 200
Batch Size: 100
```

From here we can run the line python generate.py adamcbest.pt and we will be presented with text generated from our model. In the case of the line provided we obtained output such as this: that he was in the gop has to presidential seriously there thing. else for white fightents, lindsey'

Just as was stated in the above paragraph all hyperparameters for text generation were kept to their default values.

```
Prime string = ' '
Output length = 100
Temperature = 0.8
```

Another model used was based off the IBM Developer article published by Uche Ogbuji: *"Using Markov Chains to generate language from letter correlation matrices and N-grams"* (Ogbuji, 2018). It went through the steps of generating n-gram frequencies at a letter level using the Natural Language Tool Kit library (nltk). From there, output text based on the n-gram frequencies would be reproduced. We followed the article, modifying the code to work with a set of tweets. We collected outputs for several different n-grams shown below for 3-gram and 10-gram.

The first part of this model counted the frequency of each n-gram which simply took in the data and counted the frequency of every n-gram

that appeared. For each user, we fed it the csv file containing the respective user's tweets. It output the n-gram frequencies into nGramFreq.json for each respective n-gram.

The second part of the model took the nGramFreq.json file and followed a similar markov process as show above in Morisette's tweet generator.

Sample output put from this model:
3-gram model

```
"itur ms bill of they amiliong so la
    paturs don a grocranitice of han the
     thoplegethermorepddrok good of the
    thans the co mocratimers they hug"

" spar cand crageried las oudurly we
    brnothe demills lehour who amp
    rondfinuclawleattpstrulecrincre i
    knot know counch to inny policke
    safeten"

"zokin to mostrult istch nomen
    tolareateresparyoutione the elicamp
    riblif of johe omontaxpled of
    themsyrestrousto ma stodwily wit se
    inthey do"
```

10-gram model

```
" and corrupt cities in honor of
    awarding the inspector general kelly
     book is a scam investigation thats
    because of the war on coal and will
    be f"

"ary and trade many other countries who
    treat the press also comes with a
    really get something i have no fear
    american people hurt so bad for our"

" on the fact that no government funding
     to pass through and it will all
    just have stated mary matalin a
    great sovereign nation we wont
    forget th"
```

When we use 3-gram, the output is gibberish. Increasing n gives output that make sense at first glance; however, with some obvious errors.

We also modified this to work at a word level. One output it gave was (7-gram):

```
"you want to protect law-abiding
    americans vote than claudia and she
    is a producer have a president that
    is loyal and just concluded a
    briefing with the fbi pleased to
    announce that matthew g whitaker
    times wrote a long and boring
    article for buddforcongress and
    markharrisnc9 https t c"
```

Which does not make up words, but does include parts of links and gets cut off by the length limit.

At the bottom of the document will be the graphs for the bag of words for each user (top 50).

4 Error analysis

Parts of our worries proved to be true. For example, given the training set on Barack Obama, the output of our tex gen rnn contained tweets such as I'm really WIFE and now I'm going over some files or tomorrow and then relaxing beore going to bed. Obviously the beore, is intended to be before. The problem with the context of links is also represented in the output of this model through a similar mechanism, the hashtags. Another output tweet is upgrade: this equal. join this movemet in people to unitedonclimate. https://t.co/sutaukoazn. In this tweet, the content of the tweet does not match the hashtag. Nowhere in the tweet is there a mention of something related to climate. The model does not understand the difference and intent between the content and the hashtag. As a result, the reader of the tweet will easily be able to see the inconsistency of between the two and the model will have created an undesirable tweet.

However as stated in the first section and shown in the example output tweets, our evaluation metrics of these systems were off. Similarly, our worries were too far advanced. The biggest glaring problem in our models is the incoherency in all the sentences. Not only do our model outputs not resemble the designated target, they do not resemble human speech. Currently, our models will fail on any sort of input. The biggest idea we have on improving our models is providing the models with more data. Given more data, the models ideally will be able to recognize when to use certain words and hashtags. Already, our tex gen rnn recognizes that links and hashtags are usually at the end of sentences.

However, we also intend to improve our models by pruning links, because they do not provide any more information from the tweeter. We also intend to make hashtags more sensible by basing it on the topic of the tweets. If the word climate is mentioned multiple times inside a tweet, then the tweet might want a hashtag climate at the end.

Also providing our models with an initial string is something we might explore as our n-gram markov model currently has sentences starting with a random letter or half a word.

5 Your approach

We have successfully found pre-existing models that can do the desired task. We found models that were already built for the specified task of generating tweets based on a user's past tweets along with general text generation models that existed that we modified to work with tweets. We also found articles that walked through implementing generation models for text. We have generated a large set of mock tweets for each respective user and we now need to evaluate them.

We collected the Twitter data using Tweepy and the twitter API and parsed the data into json files and csv files to work with each of the models we are looking at. After giving the models the input on each of our personal computers, we are getting an output of 100 tweets per model.

The models we are looking at are:

char-rnn.pytorch

"A PyTorch implementation of char-rnn for character-level text generation."

(Robertson, 2017)

textgenrnn

"textgenrnn is a Python 3 module on top of Keras/TensorFlow for creating char-rnns"

(Woolf, 2018a)

oliviersm199 Tweet Generator

"The tweet generator uses a Markov chain finite state model to take a particular user id from twitter and generate tweets based off of that person's previous tweets."

(Morissette, 2018)

minimaxir Tweet-Generator

"Tweet Generator is based off of textgenrnn, and trains the network using context labels for better tweet synthesis."

(Woolf, 2018b)

Using Markov Chains to generate language from letter correlation matrices and N-grams

"Modeling natural language characteristics at the level of the word and generating frequency plots"
***We modified this to also work at the word level as another model.

(Ogbuji, 2018)

Text-Generation-using-Bidirectional-LSTM-and-Doc2Vec-models

"Text Generation using Bidirectional LSTM and Doc2Vec models"

(campdav, 2018)

Notable libraries involved:
Natural Language Toolkit, PyTorch, Tensorflow, tweepy

An issue we ran into with the textgenrnn is that it would produce an empty output most of the time. It occasionally gave us something that we could work with. We were not sure if this was a code issue or a machine issue.

Another issue, which is more of a limitation is the 3200 max tweets we can download from a user's timeline. We're also not including any retweets so that reduces the amount of data we can work with as well. For example, Trump and Obama both only have about 2800 tweets we can access.

Now we plan to compare the outputs and find the best one, as each model exhibits some sort of errors. From there, we will optimize the best model to the task of generating tweets.

6 Timeline for the rest of the project

1. Analyze and compare the results to find the best model. (1/2 Week)
2. Then we will optimize the best model for the task. Changing aspects of the model, such as the loss function used. (1/2 Week)
3. Optimize the worst model for the task (Optional)
4. Examine how well the final model mimics a twitter user's writing style. Do error analysis. (1 Weeks)
5. Write final report and prepare presentation. (1 Weeks)

References

campdav (2018). Text generation using bidirectional lstm and doc2vec models. https://github.com/campdav/Text-Generation-using-Bidirectional-LSTM-and-Doc2Ve

Morissette, O. (2018). Tweet generator. github.com/oliviersm199/Tweet-Generator.

Ogbuji, U. (2018). Using markov chains to generate language from letter correlation matrices and n-grams.

Robertson, S. (2017). char-rnn.pytorch. https://github.com/spro/char-rnn.pytorch.

Woolf, M. (2018a). textgenrnn. https://github.com/minimaxir/textgenrnn.

Woolf, M. (2018b). Tweet generator. https://github.com/minimaxir/tweet-generator.

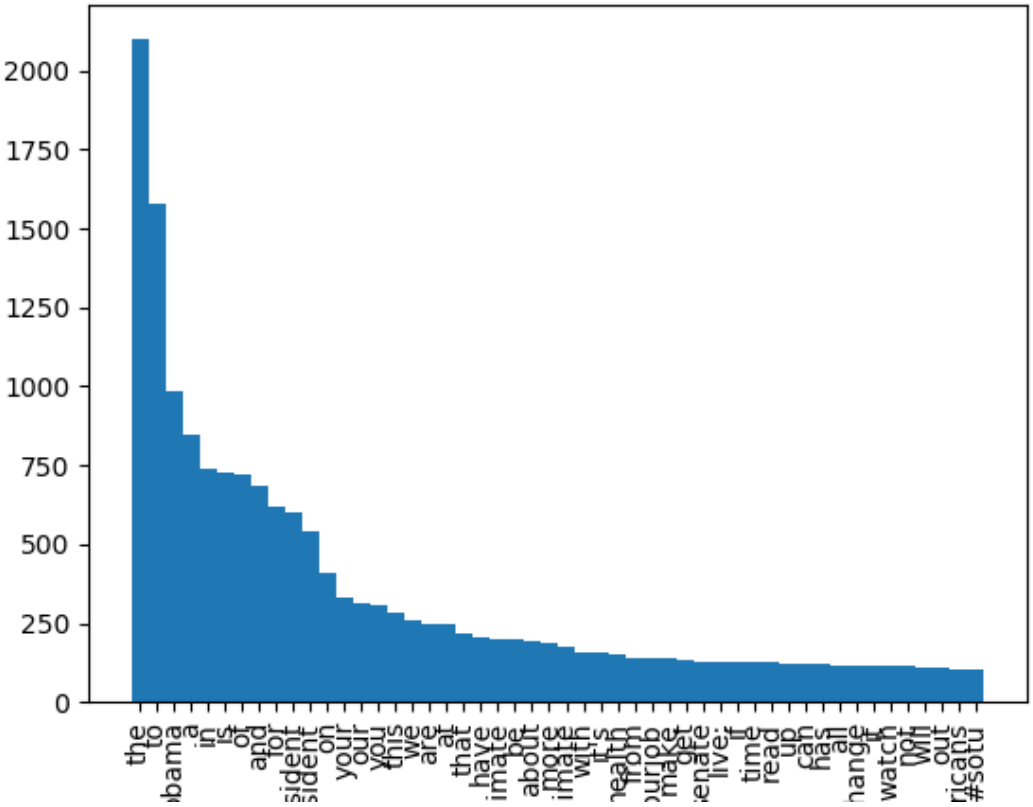

Figure 1: Obama's top 50 bag of words

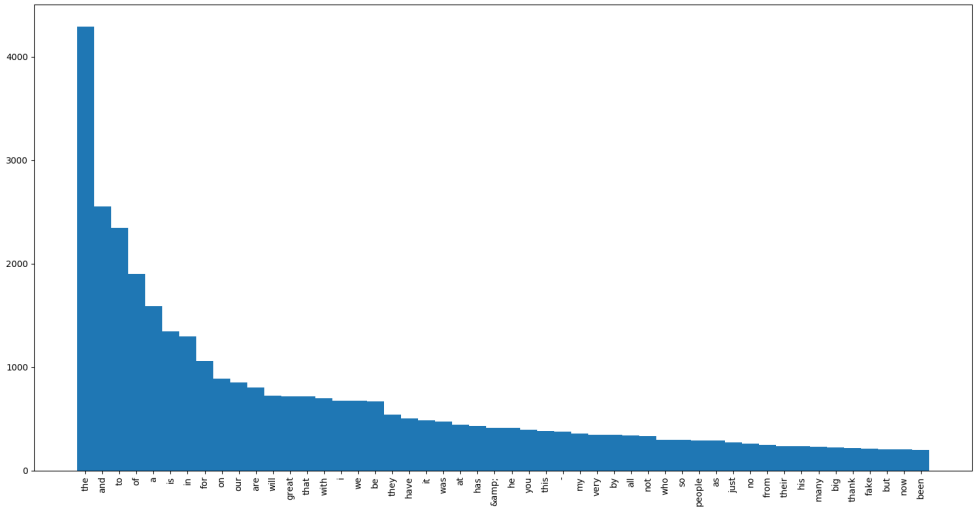

Figure 2: Trumps's top 50 bag of words

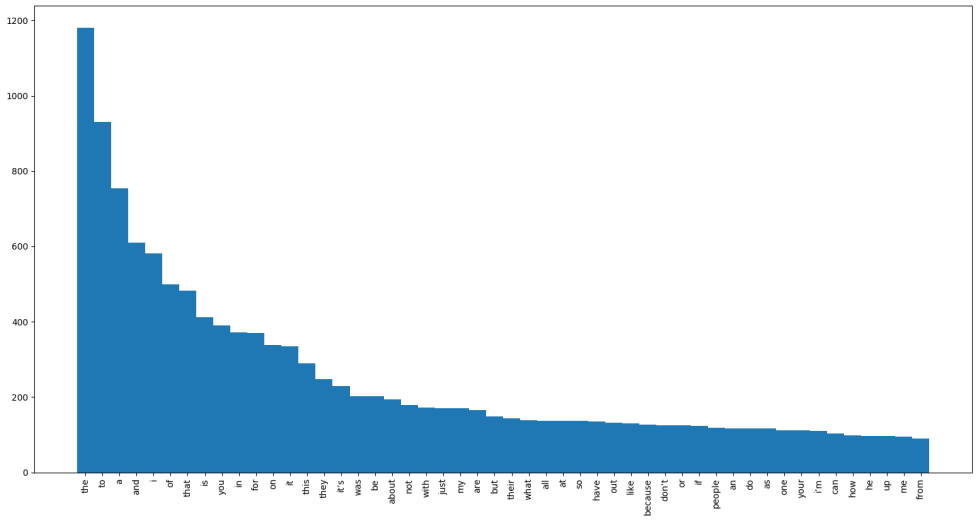

Figure 3: Mindys's top 50 bag of words

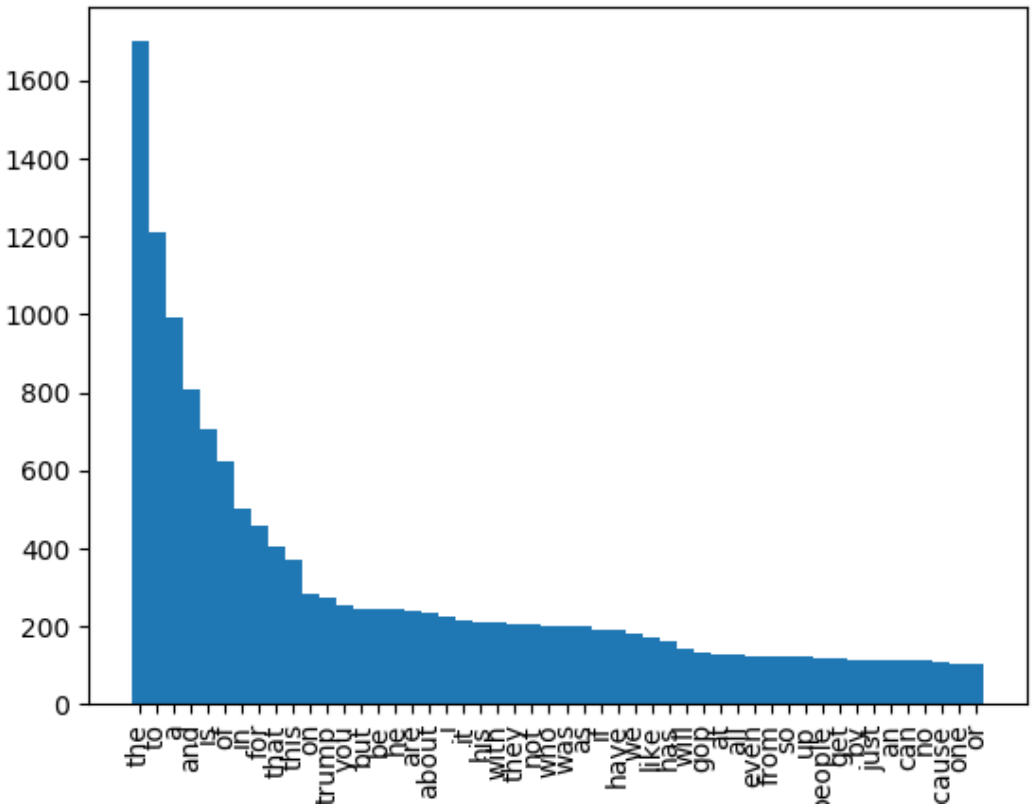

Figure 4: Adam's top 50 bag of words