Trumpy Tweets: Tweet Classifier and Generator

Wenjun Sun
UNI: ws2578
ws2578@columbia.edu

Abstract
In this project, I built two tweet generators and two tweet classifiers, and used the better-performed classifier to evaluate the performance of the generators. For the generators, I used about 24k tweets of Donald Trump to train two text generators. Second, I added another 24k tweets from Obama, Hillary, and Kim Kardashian and trained two tweet classifiers. Third, I compared the performance of classifiers and used the better-performed one to evaluate the performance of the tweet generators.

Introduction
As a particular application of Natural Language Generation (NLG), automatically generating text that mimics what a real person says is an interesting yet challenging problem. Thanks to various modeling techniques scientists have developed during the recent years as well as abundant data available on the Internet, letting computers generate almost-real sentences that can easily fool people is no longer a wild dream.

I choose to build a tweet generator that mimics Trump for two reasons. First, Trump frequently uses social media, especially Twitter. His twitter obsession provides a large number of tweets. Second, studies found that Trump has a very unique writing style, featured with “a casual tone, a simple vocabulary and grammar, repetitions, hyperbole and sudden switches of topics” (Inzaurralde, 2019). Both factors make Trump's tweets an ideal data source for training a text generator.

In this project, I implemented two style-mimicking text generators using Multinomial Naive Bayes Model and Long short-term memory (LSTM) model; they generate tweets in the style of Trump. Meanwhile, I separately trained two text classifiers using Markov Chain Model and LSTM model that aim to distinguish Trump's tweets and other people's tweets. Then I used the text classifier with better accuracy to evaluate the performance of the text generators by seeing what the proportion of generated tweets that successfully fooled the classifier is.

Related Work
Text Generation
There are primarily two types of language models for building text generators. One is statistical language models, which use traditional statistical techniques like N-grams, Hidden Markov Models (HMM), and certain linguistic rules to learn the probability distribution of words. Another is neural language models, which emerged recently and have surpassed the statistical language models in their effectiveness, using different kinds of neural networks to model language, including feedforward neural networks, recurrent neural networks and convolutional neural networks.

The underlying mathematics of the n-gram was first proposed by Markov. Shannon (1948) applied n-grams to compute approximations to English word sequences. Based on Shannon's work, Markov models were commonly used in engineering, linguistic, and psychological work on modeling word sequences by the 1950s. Later, Jelinek (1976) at the IBM and Baker (1975) at CMU independently used n-grams in their speech recognition systems with some success. The highest accuracy language models by now are neural network language models, including feedforward language models (Bengio et al. 2003), recurrent language models (Mikolov 2012), and gated convolutional networks language model (Dauphin 2016). They solve a major problem with n-gram language models: the number of parameters increases exponentially as the n-gram order increases, and n-grams have no way to generalize from training to test set. Neural language models instead project words into a continuous space in which words with similar contexts have similar representations.

Text Classification

There are also two primary approaches for building text classifiers. One is rule-based systems, which classify text into organized groups by using a set of handcrafted linguistic rules. These rules instruct the system to use semantically relevant elements of a text to identify relevant categories based on its content. Each rule consists of an antecedent or pattern and a predicted category. Rule-based systems are human comprehensible and can be improved over time, but these systems require deep knowledge of the domain. Also, generating rules for a complex system can be very time-consuming, and these systems don't scale well given that adding new rules can affect the results of pre-existing rules.

Another is machine learning based systems, which learns to make classifications based on past observations instead of relying on manually crafted rules. They are usually much more accurate than human-crafted rule systems, especially on complex classification tasks, and are also easier to maintain and scale by only feeding new tagged training examples to the model to learn new tasks. Some of the most popular machine learning algorithm for creating text classification models include traditional statistical methods like Naive Bayes (Wang & Manning, 2012), support vector machine (Silva et al., 2011), modified logistic regression (Le & Mikolov, 2014), and neural network methods, including recursive neural networks (Wermter, 2000) and convolutional neural networks (Kalchbrenner et al., 2014, Santos & Gatti, 2014, Shen et al., 2014).

Data

Dataset

This project is based on two datasets – a set of Trump’s tweets and a set of other people’s tweets. The classifiers used both datasets, while the generators only used the former dataset.

Given the Twitter API has many limits– for example,it only gives access to the last 3200 tweets of a user, I got Trump’s tweet from a website called “Trump Twitter Archive”. It is maintained by Brendan Brown and it collects almost all of Trump's tweets. The data is archived by 12/2/2019, it contains 36307 tweets. A considerable proportion of this dataset are retweets or quotations, which makes proper data cleaning extremely important for the success of the models.

The second dataset is tweets from other people. I choose tweets from Barack Obama, Hillary

Clinton, and Kim Kardashian as “other people’s tweets” for training the classifiers. This dataset is a subset of the "Raw Twitter Timelines w/ No Retweets" dataset on Kaggle. The size of this dataset is 24196; however, it also contains quotations and should be cleaned before feeding to models.

Data Preprocessing

The data preprocessing is divided into two parts. The first part of preprocessing is done to all tweets, which the second part of preprocessing is only done to tweets that are used for training classifiers. The reason why it is done in two parts is that: the goal of the classifiers is to correctly categorize an input sentence, while the goal of the generators is to generate tweets that look as similar to Trump's as possible. Therefore, the data I want the classifiers to learn is different from that for the generators. For example, for the generator, I want to keep @mention, #tag, and special characters (like !?,.) in the dataset, because I do want our bot to be able to @mention, #tag, and using exclamation marks. However, things like @, #, and other special characters create many difficulties in tokenizing, so I need to remove them before training the classifiers.

For the first part of preprocessing, i.e., the part that is done to all tweets. I removed all the retweets, removed quotations, URLs, “\n”s from tweets, and replaced “&amp” with “and”. After removing empty tweets (most likely to be tweets that are quotations), the Trump dataset is left with 24595 tweets and the other people dataset is left with 24127.

A second part of preprocessing is done to the data that are to train the classifiers. First, tweets are lowercased. Second, common abbreviations like “’ll”, “won’t”, “n’t” are replaced with their full forms. Third, mentioning and tagging are removed, along with other special characters. Finally, stopwords are also removed, except for "no", "not" and "never". Common tweet symbols such as "rt" and "cc" are also added to the stop word list for removal.

Method

Evaluation Metrics

There are two sets of models to evaluate in this project: (1) a text classifier for distinguishing Trump's tweets, and (2) a text generator for generating made-up tweets in the style of Trump.

The classifiers are optimized towards precision, which equals to $\frac{True\ Positive}{True\ Positive+False\ Negative}$, and in this case, it can be written as $\frac{True\ Trump\ tweet}{All\ tweets\ predicted\ to\ be\ Trump's}$. Intuitively, precision is the ability of the classifier not to label a fake tweet as Trump's. I am aware that metrics like F1 score and accuracy are more commonly used in classification evaluation. However, given the goal of the classifiers is to try not to be fooled by the generators, I believe precision is a more appropriate metric for evaluating the classifiers in this case.

The text generator is essentially a language model. The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called extrinsic evaluation. Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus, for text generation, we can compare the performance of two generators by running a separately trained text classifier. The one which is better at fooling the text classifier, i.e., whose made-up tweets

yielding lower accuracy on the text classifier is considered a better text generator.

Models

Two kinds of models are built: classifiers and generators.

For text classification, a Multinomial Naive Bayes model and an LSTM model are trained and compared. The classifier with the highest prediction accuracy on the test set is considered the best classifier and is used to evaluate the generators.

For text generation, two models are used and compared. First, a traditional word n-gram model is used, including bigram, trigram, and trigram with POS tagger models. Second, a neural language model is used, including a character-level long short-term memory (LSTM) model and a word-level LSTM model.

Tweet Classifiers

Two different text classifier models are trained and compared: One is a Multinomial Naive Bayes text classifier, a traditional statistical approach; the other is an LSTM text classifier, a modern neural network approach.

Multinomial Naive Bayes Text Classifier

Multinomial Naive Bayes implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (the other is Bernoulli Naive Bayes model, which also assumes multiple features but each feature is binary-valued). The features are usually word vector counts, but tf-idf vectors are also known to work well in practice.

After a 10-fold grid-search on the training set, the best set of parameters is selected, and the model is then trained on the training set.

LSTM Text Classifier

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture that has feedback networks. It was developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. As a variation of RNN, LSTM is well-suited to classifying, processing, and making predictions on serial data.

Before fed to the LSTM model, the input tweets are tokenized, then each sentence is "standardized" and each word is transformed into an index in the truncated vocabulary. Any tweet that is longer than 40 words is truncated at the tail, and any tweet that is shorter than 40 words is padded at the tail with value 0.

The model implemented in this project has two Bidirectional LSTM Layers, two Dense Layers, and an output layer. The detailed architecture of the LSTM model is as follows:

- The tokenized sentences are feed into an embedding layer. Weights from GloVe embeddings pretrained on Twitter (200 dimensions) are used. Every word in a sentence is transformed from an index in the vocabulary to a dense vector of fixed dimension 200. A dropout layer is

attached, with a dropout probability of 0.4.
- Two bidirectional LSTM layers with hidden dimension 128 are chained together. A dropout layer is attached to the last LSTM with a dropout probability of 0.2.
- A dense layer with hidden dimension 128 using ReLU activation function is applied. A dropout layer is attached, with a dropout probability of 0.4.
- A dense layer with hidden dimension 64 using ReLU activation function is applied. A dropout layer is attached, with a dropout probability of 0.5.
- A sigmoid activation function is applied to generate a number between 0 and 1 representing the output probability of label 1.

Tweet Generators

Two text generation models are trained and compared: One is a Markov Chain text generator, a traditional statistical approach. The other is an LSTM text generator, a modern neural network approach.

Markov Chain Text Generator

Markov Chain is based on the idea that: the possible next states are entirely dependent on the present state.
Three Markov Chain text generators implemented in this project: a bigram one, a trigram one, and a trigram one with a part-of-speech (POS) taggers.

A bigram is a sequence of two adjacent elements from a string of tokens. It helps provide the conditional probability of a token given the preceding token. Likewise, a trigram is a sequence of three adjacent elements from a string of tokens. It helps provide the conditional probability of a token given the two preceding tokens. In addition to the trigram model, the third model, i.e. the trigram + POS one uses spaCy's part-of-speech tagger to generate a Markov model that obeys sentence structure better than a naive model.

LSTM Text Generator

Two LSTM text generators are trained: a character-level one and a word-level one.

The character-level one is a bidirectional LSTM model with 3 layers, each of which has 128 cells. (I wanted to train a model with 256 cells per layer, but it would too very long time – about 6 hours and Google Colab would halt the cell before it finishes running.) Up to 30 characters are considered before predicting the next token. No dropout is applied.

As for the word-level one, it is a bidirectional LSTM model with 3 layers, each of which has 256 cells. Up to 8 previous words are considered before generating the next word. A maximum of 10000 words are modeled, the rest with lower frequencies are ignored. Similar to the character-level model, no dropout is applied.

Experiment and Result

Text Classifier

Data is shuffled and split into 80% training set (38977 tweets), 10% validation set (4873 tweets), and 10% test set (4872 tweets). Trump tweets are labeled 1 and the non-Trump tweets are labeled 0.

	precision	recall	f1-score	support
0	0.9804	0.9772	0.9788	4822
1	0.9777	0.9809	0.9793	4923
accuracy			0.9791	9745
macro avg	0.9791	0.9790	0.9791	9745
weighted avg	0.9791	0.9791	0.9791	9745

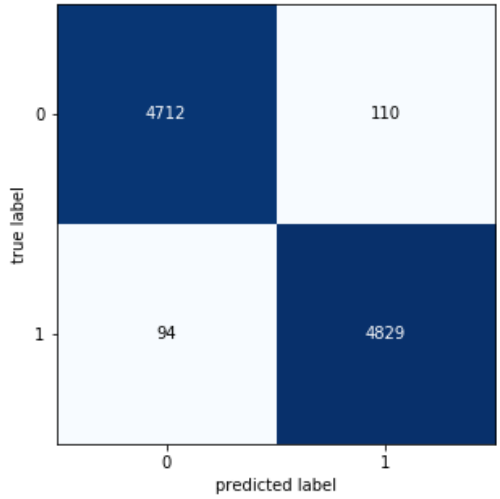

Figure 1 confusion matrix of the Multinomial Naive Bayes classifier

The model is trained with the training set, cross-validated with the validation set, and tested with the test set.

As the classification report and confusion matrix show, the overall accuracy of the LSTM classifier is 0.8955. For non-Trump tweets, the precision is 0.9008, the recall is 0.8885, and the F1-score is 0.8946. For Trump tweets, the precision is 0.8905, the recall is 0.9025, and the F1-score is 0.8964.

In conclusion, the Multinomial Naive Bayes classifier is chosen as the final text classifier used to discriminate the fake tweets generated by text generators since it has a higher test accuracy.

Text generators

As discussed above, the Multinomial Naive Bayes classifier is chosen to evaluate the performance of the text generators. For each of the generators, I generate 1000 tweets and input them into the Multinomial

Multinomial Naive Bayes Text Classifier

For the Multinomial Naive Bayes classifier, after grid search, the best hyperparameters for the model are: using unigram, bigram, and trigram; using raw counts instead of tf-idf, and using a smoothing coefficient of 0.1.

The model is trained with the training set and tested with the validation set and test set combined. (Because the best parameter with selected by 10-fold grid search over the training set, so the validation set actually stays untouched, so I also used it for testing.)

As the classification report and confusion matrix shows, the overall accuracy of the Multinomial Naive Bayes classifier is 0.9791. For non-Trump tweets, the precision is 0.9804, the recall is 0.9772, and the F1-score is 0.9788. For Trump tweets, the precision is 0.9777, the recall is 0.9809, and the F1-score is 0.9793.

LSTM Text Classifier

Quite surprisingly, the LSTM model performs worse than the simple Multinomial Naive Bayes model.

	precision	recall	f1-score	support
0	0.9008	0.8885	0.8946	2431
1	0.8905	0.9025	0.8964	2441
accuracy			0.8955	4872
macro avg	0.8956	0.8955	0.8955	4872
weighted avg	0.8956	0.8955	0.8955	4872

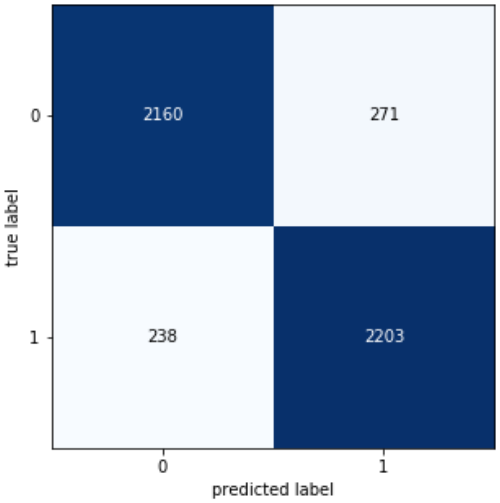

Figure 2 confusion matrix of the LSTM classifier

Naive Bayes classifier. The proportion of made-up tweets that successfully fooled the classifier is calculated. Based on the probability score produced by the classifier, I also output the tweet with the highest probability to be Trump's, the one with the 75th percentile probability value, and the one with a median probability value. (To make the output clear for reading and comparison, I put them into tables.)

Markov Chain Text Generator

Bigram model

Among the 1000 tweets generated by this model, the proportion of tweets fooled the classifier is 0.93. The table below shows the best, 75th percentile, and median tweets generated by this model.

	Made-up tweets	Probability[1]
Most "Trumpy" tweet	From a large shipment of military weapons and supplies to my GREAT members at Trump International Golf Links in Scotland!	0.99997
Tweet of 75th percentile	Watch it, should be ashamed for using the celebrity's name followed by new champion Chris Weidman!	0.98488
Tweet of median probability	#Debates .USA has the guts to even greater unemployment.	0.94061

Trigram model

Among the 1000 tweets generated by this model, the proportion of tweets fooled the classifier is 0.946.

	Made-up tweets	Probability
Most "Trumpy" tweet	The Rigged Witch Hunt headed by 13 Angry Democrats and others who are totally corrupt and/or conflicted.	0.99999
Tweet of 75th percentile	Thank you, a very wise move that Ted Cruz FORGOT to file.	0.98758
Tweet of median probability	In the meantime they continue to be drawn to it.	0.95545

Trigram + POS model

Among the 1000 tweets generated by this model, the proportion of tweets fooled the classifier is 0.947.

	Made-up tweets	Probability
Most "Trumpy" tweet	We want victory .. Trump International Golf Links Scotland .	0.99998
Tweet of 75th percentile	I will beat Hillary easily , but Lindsey Graham says I wo n't be around much longer , it 's called Envy .	0.98675
Tweet of median probability	.On behalf of an entire Nation , THANK YOU for today 's update and GREAT WORK !	0.95023

[1] The probability produced by the classifier, which indicates the probability of the input tweet to be Trump's.

LSTM Text Generator

Character-level model

Among the 1000 tweets generated by this model, the proportion of tweets fooled the classifier is 0.985.

	Made-up tweets	Probability
Most "Trumpy" tweet	.@marklevinshow interview discussing the Celebrity Apprentice and Crooked Hillary Clinton will only get born in the world. I will be the best thing to have the haters and the truth in the New York Times - just reported that I am the only one that the Fake News Media will do a g	0.99999
Tweet of 75th percentile	My @SquawkCNBC interview discussing the boardroom of the United States of Americans will be a great deal for the fact that China is always going to be the greatest of the world, and all others, get ready for a winner!	0.99595
Tweet of median probability	A MUST has spent records and positive process of Edwarding to President Obama for a failed policy in the tragic deal done by @SecretaryZinkens will be a great guy who spends and the field in the U.S. Military will be a disgrace to the American people and the victims of the Stoc	0.97986

Word-level model

Among the 1000 tweets generated by this model, the proportion of tweets fooled the classifier is 0.967.

	Made-up tweets	Probability
Most "Trumpy" tweet	he big story that the fake news media refuses to report me for @ cnn . my great honor to speak at the @ ussarizona . thank you ! # trump2016	0.99996
Tweet of 75th percentile	" i have great respect for the people that represent the united us care . remember that the great president ? he is a nice guy !	0.97720
Tweet of median probability	. @ hillaryclinton is on the front page of the illinois . . . but write a . that ' s because it should be an economic failure ' s to mexico . the problem ' 12 . you are both more .	0.89935

Conclusion

In terms of the tweet classification, simple traditional Naive Bayes models outperformed expensive LSTM models on the text classification task for distinguishing Trump's tweets from other people's tweets. I think it is because text classification is a relatively easy task and our dataset is not large enough, so complex models like LSTM might suffer from overfitting. In other words, if we have more data, the performance of LSTM might get better.

As for the tweet generation, by reading the generated tweets, I think both the Markov Chain models and the LSTM models did a pretty good job at mimicking Trump's tweet. Their performance of fooling the classifier confirmed my observation – at least 93% of their tweets fooled the classifier.

Although by my human judgment, tweets generated by Markov Chain and LSTM are all pretty good, the LSTM generator seems to be better at fooling the classifier – a higher proportion of its tweets successfully fooled the classifier.

The architecture of this project was inspired by the Generative Adversarial Networks (GAN). A GAN model has two entities – Generator and Discriminator, and it learns from the constant battle between the two entities. However, due to the time constraint, this project did not include such models. So, for future exploration, I would like to implement Generative Adversarial Networks (GAN) to boost the performance of both the text generator and the text classifier.

Reference

Baker, J. (1975). The DRAGON system--An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *23*(1), 24-29.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137-1155.

Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2017, August). Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 933-941). JMLR. org.

Dos Santos, C., & Gatti, M. (2014, August). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers* (pp. 69-78).

Inzaurralde, B. (2019, April 28). Analysis | This linguist studied the way Trump speaks for two years. Here's what she found. Retrieved December 17, 2019, from https://www.washingtonpost.com/news/the-fix/wp/2017/07/07/this-linguist-studied-the-way-trump-speaks-for-two-years-heres-what-she-found/.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, *64*(4), 532-556.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Le, Q., & Mikolov, T. (2014, January). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188-1196).

Mikolov, T. (2012). Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, *80*.

Schwenk, H. (2007). Continuous space language models. *Computer Speech & Language*, *21*(3), 492-518.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, *27*(3), 379-423.

Shen, Y., He, X., Gao, J., Deng, L., & Mesnil, G. (2014, November). A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management* (pp. 101-110). ACM.

Silva, J., Coheur, L., Mendes, A. C., & Wichert, A. (2011). From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, *35*(2), 137-154.

Wang, S., & Manning, C. D. (2012, July). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the association for computational linguistics: Short papers-volume 2* (pp. 90-94). Association for Computational Linguistics.

Wermter, S. (2000). Neural network agents for learning semantic text classification. *Information Retrieval*, *3*(2), 87-103.