Generating “Bull”

Uma Dwivedi
Department of Sociology
Yale University
New Haven, Connecticut
uma.dwivedi@yale.edu

Murage Kibicho
Department of Mathematics
Yale University
New Haven, Connecticut
murage.kibicho@yale.edu

Abstract

This article describes the development of a meaning-agnostic text generator based on statistical models of textual datasets. We analyze the “hidden” structures of written text and develop mathematical models to represent this underlying structure. We take into account the transitional nature of textual data and use these transitions to train a bigram hidden markov model for text generation. We detail the key components of our hidden markov model and describe our model’s ability to generate new text. In particular, we hope to answer the following questions : (i) Can we create a mathematical model capable of imitating an author or genre’s style? (ii) Given a set of genres, how can we train our computational model to generate genre-specific text? (iii) How intelligible is our randomly generated text, and if it is coherent, is our generated “bullshit” indistinguishable from human-authored text?

1 Introduction

The development of intelligent natural language generators capable of conveying tone, style and general textual meaning is an active area of research. Early textual systems such as Eliza (Weizenbaum 1966) were designed to mimic human behavior based on a set of hard-coded rules. Despite their success, they were limited by their inability to generate meaningful text outside of a set of predetermined parameters. In this article, we describe the use of statistical methods to overcome the limitations of generating text based on a set of hand-written rules. Specifically, we use hidden markov models to create dynamic computational models of textual data. Then, we present the implementation of our computational models as a meaning-agnostic text generator in Python.

2 Methods

We chose to use bigram hidden markov models as the basis of our computational models. In this section, we shall discuss our implementation of hidden markov models, as seen in the code provided with our paper. We shall take you through the process of generating bigram pairs and bigram frequency counts. Then, we shall explain how we used lambda smoothing to generate our transition probabilities. Finally, we shall show how each part is incorporated into the final text generator.

2.1 Implementing Bigram Logic

In this section, we shall describe how our function “GenerateBigrams” works. We chose to use default dictionaries as our main data structure.

We have two dictionaries, one to store counts of each word within our training text and another to store all possible bigram pairs and the number of times each pair has been observed. We use a for loop to iterate through each line of our training data. Every word within our training data is saved as a dictionary entry and a count of each word’s occurrence is stored alongside the word. In the second default dictionary, we also store each word, but we also store all its possible successors and how often each word pair has been observed. Thus, our second default dictionary holds each bigram pair and its total count. So, our function returns two dictionaries: (1) The single word dictionary. (2) The bigram pair dictionary.

2.2 Generating Transition Probabilities

This section is about our function "FillTransitionMatrix". The function takes in a dictionary of all possible bigram pairs and a dictionary of each word's frequency.
We chose to find transition probabilities using the joint probability formula:

$$P(t,w) = P(t)P(w|t) \qquad (1)$$

Implementing this formula in Python involves dividing the total number of occurrences of each bigram pair by the total number of occurrences of the first word in each bigram pair. After performing division, we use lambda smoothing to prevent 0 probabilities being stored within our frequency counts. We use a lambda value of 0.00005 and an upper bound of 0.0001 to determine whether we should add or subtract the lambda value. The calculated probabilities are then stored alongside their associate bigram pairs. This dictionary with probabilities is our transition matrix and it is the only value returned by our function.

2.3 Obtaining and Cleaning Data

This section is about the training data we use for our hidden markov model. The basic model, as described above, was trained on the Brown corpus of data (Francis et. al. 1979), which contains one million words, separated into sentences, of text from a variety of sources (including news articles, literature, academic articles, religious texts, etc.). The variety of natural language sources in the Brown corpus gives it data in a variety of registers; as our program generates text in a variety of registers, the Brown corpus seemed like a good fit.

We used five topic-specific datasets in our program, one each for jokes, sci-fi, regency England, finance, and medicine.

The jokes dataset (Pungas 2017) pulled text from wocka.com. We cleaned this data to get rid of dialogue tags (i.e. the underlined portions in, "<u>Customer:</u> *dialogue* /n <u>Me:</u> *dialogue*); newline characters; numbers; whitespace characters; joke categories, identification numbers, and titles; and asterisks marking expletives. We flattened the text of every joke into one long string, which we then split on periods, exclamation points, and question marks to get list items each containing one sentence. These sentences were then further tokenized into sublists, with each item of the sublist containing one word or punctuation mark.

The sci-fi dataset (Broughton 2018) consisted of the complete scripts of every episode or movie in the Star-Trek world. Cleaning for this dataset was similar to cleaning the jokes dataset: we removed dialogue tags, removed all characters that were neither letters nor punctuation, and flattened all the data so that text was not differentiated by series or episode. Like above, we separated the flattened text data into sentences and tokenized those sentences.

The regency England dataset consisted of the full text for Jane Austen's six complete novels. We downloaded the plain text versions of these novels from Project Gutenberg (Austen 1994a, 1994b, 1994c, 1994d, 1994e, and 1998) and manually removed Gutenberg's boilerplate text from the front and back of each book, leaving only the main text of each novel behind. We then loaded these six text files into a single data structure, which was then cleaned and tokenized using the same methods as previously described.

The financial dataset (bot_developer 2020) scraped headlines pertaining to approximately 6000 stocks and compiled them into a csv. We read in this csv, lowercased the title-case text, removed numerical and financial symbols (#, %, $, -, all numbers), and removed the 'title' label. As these were already single sentences, we did not need to split them by sentence, but we tokenized them using our standard methods.

The medical dataset (Boyle 2018) consists of medical transcription samples scraped from the website mtsamples.com. Medical data is difficult to obtain legally and ethically, due to HIPAA regulations, but the medical transcription samples on mtsamples.com are thoroughly scrubbed for any and all identifying information. This data was read in, cleaned, split into sentences, and tokenized in the manner described above.

The user is guided by the command-line interface to select one of the five topics. Based on their selection, one of the datasets is read in, cleaned, and tokenized.

2.4 Calculating Topic-Specific Bigram Counts

To calculate topic-specific bigram counts, we run the tokenized, sentence-split data through GenerateBigrams (described above) to generate word frequency counts and a dictionary storing bigram counts. The frequency counts and bigram count dictionary are combined with the base-language model's frequency counts and bigram count dictionary; these four data structures are then inputted into UpdateTransitionMatrix to calculate updated transition probabilities.

2.5 Updating Transition Probabilities to Reflect Topic-Specific Data

This section will describe the function UpdateTransitionMatrix, which we use to update our transition probabilities based on the topic-specific dataset selected by the user.

As was the case for the base language model's transition matrix, UpdateTransitionMatrix uses lambda smoothing to prevent probabilities of 0. We use a lambda value of 0.00005 and an upper bound of 0.0001 to determine whether we will add or subtract the lambda value. The function then iterates through all bigrams present in topic_dict (a dictionary counting bigrams in the topic-specific text) and calculates the probability of each bigram based on the quotient of counts for the bigram and counts for the first word of the bigram. This value is stored as topic_prob, and lambda smoothing is then applied to it. If a bigram is present both in the topic_dict and the bse language model's bigram dictionary, we add the counts of the bigram occurring in the base model to the counts of the bigram occurring in the topic-specific data. This sum is divided by another sum: the base-language model counts of the first word in the bigram are added to the topic-specific counts of the first word in the bigram. This quotient gives us an updated bigram probability, which is stored in a new transition matrix. If the bigram at hand is not in the base language model, the updated matrix uses topic_prob (based only on counts in the topic-specific data) to reflect a bigram's probability.

2.6 Generating "Bull" Sentence

In this section, we shall describe how a user runs the program. This section deals with the structure of our main function and the steps taken to generate a sentence using the "GenerateRandomPhrase" function .

When the user runs 'python markov.py', the program first trains the hidden markov model by generating all possible bigram pairs and filling in the transition matrix. Once the model is generated, the user is prompted to pick a category of "bullshit". The user can choose to generate jokes, science fiction, medical text, financial text or English text from the Regency Era.

Based on the user's choice, a random word is selected from all words in the topic-specific data (this choice is weighted by counts of that word divided by total word count). This word is then used as the starting word for our sentence. The random word and the transition matrix is passed to the "GenerateRandomPhrase" function.

The "GenerateRandomPhrase" function takes a starting word and generates a list of every possible bigram that can be generated using the starting word. The function also generates a list of transition probabilities for these bigrams. The function stops adding words to the list when it encounters a punctuation mark, or the current word cannot form any bigrams with our training data. Once the function stops adding new words to the phrase, it returns the generated phrase.

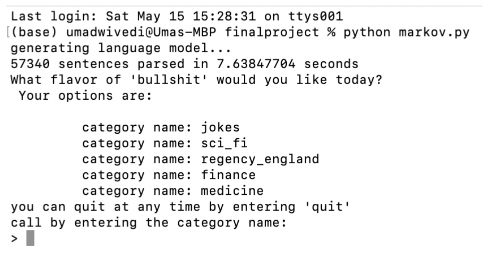
The model also calculates the likelihoods described in the section below. The generated phrase and these likelihoods are printed to the user's terminal.

2.7 Calculating Likelihoods

In order to evaluate the output generated by our program, we calculated three likelihood values: one, the likelihood of our model generating the sentence it generated; two, the likelihood of our model generating some random sentence present in the text; and three, the mean likelihood of our model generating each sentence across all sentences in the topic-specific text. If the likelihood of the sentence generated is much, much smaller than the randomly-selected sentence's likelihood, it is likely not terribly representative of the text data our model is trained on. The mean likelihood is present to provide a common reference point so all generated and random likelihoods are somewhat contextualized. All three likelihood values are printed for the user with labels alongside the generated sentence. Our program is structured so that a user can generate bull as many times in a row as their heart desires, varying the topic if they wish.

2.8 Running the Program

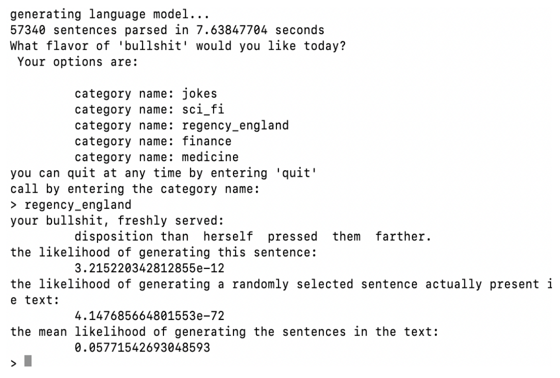
Assuming that a user has successfully installed all the necessary imports, they will see the following when they run "python markov.py" in the appropriate directory:



```
Last login: Sat May 15 15:28:31 on ttys001
(base) umadwivedi@Umas-MBP finalproject % python markov.py
generating language model...
57340 sentences parsed in 7.63847704 seconds
What flavor of 'bullshit' would you like today?
 Your options are:

         category name: jokes
         category name: sci_fi
         category name: regency_england
         category name: finance
         category name: medicine
you can quit at any time by entering 'quit'
call by entering the category name:
>
```

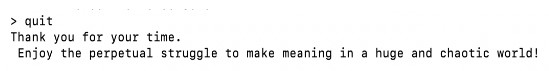
Figure 1: Initial screen



```
generating language model...
57340 sentences parsed in 7.63847704 seconds
What flavor of 'bullshit' would you like today?
 Your options are:

         category name: jokes
         category name: sci_fi
         category name: regency_england
         category name: finance
         category name: medicine
you can quit at any time by entering 'quit'
call by entering the category name:
> regency_england
your bullshit, freshly served:
         disposition than  herself  pressed  them  farther.
the likelihood of generating this sentence:
         3.215220342812855e-12
the likelihood of generating a randomly selected sentence actually present i
e text:
         4.147685664801553e-72
the mean likelihood of generating the sentences in the text:
         0.05771542693048593
>
```

Figure 2: Sample prompt screen



```
> quit
Thank you for your time.
 Enjoy the perpetual struggle to make meaning in a huge and chaotic world!
```

Figure 3: Quit screen

3 Results and Discussion

Notable output:

- "The sacrifice, he feared, would speak." (Regency England)
- "The dekendi's favourite is defenceless." (Sci-Fi)
- "The bandages were touching the mole that turned away cephalically and preparations were consistent." (Medicine)
- "The alvera trees, and utter oblivion" (Sci-Fi)
- "The dalai lama implied that without making sandwiches" (Jokes)
- "The country gives, and preach such universal improvement, though young persons was restlessly miserably forever!" (Regency England)
- "The daughter, formed such true generosity and leisure hours, mrs." (Regency England)
- "The camel, where you're halfway across, and coolant" (Jokes)
- "The midline, non painful" (Medicine)
- "The LOS sales seeing major project stakes demand declining" (Finance)
- "The kendi system toenlist the mechanism to ziyal" (Sci-Fi)
- "The meeting you home work, get outside door" (Jokes)
- "The grail again?" (Jokes)
- "The young-uns and stud back tail but don't trust google!" (Jokes)
- "The inflation industrial product names no grace your pocket calculator" (Finance)
- "The suturing spaces four carbohydrate serving" (Medicine)
- "The world nor the interval would increase his nephew" (Regency England)

4 Conclusions

We were initially drawn to this project because we were interested in what language looks like absent meaning. By developing a meaning-agnostic program that generated sentences, we got to look at how a topic can "flavor" language in a way that does not hold semantic meaning. We can have sentences inflected by, for example, Jane Austen that do not carry meaning and that are, in many cases, grammatically correct. What does this tell us about the meaning of Jane Austen in relation to her stylistics? The very fact that we can mimic the stylistics of Jane Austen in a way recognizable to the user means that there are stylistic characteristics of her work, entirely separate from semantic characteristics of her work, that typify it. That this style can be produced without meaning suggests the potential for vast quantities of text stylistically similar to great thinkers and writers but totally absent new meaning or meaning-related contribution. In other words: the fact that our machine can imitate Jane Austen in nonsense sentences implies the existence of writing that mimics Jane Austen's stylistics (and gains some access to prestige through this mimicking) without doing the meaning-work she did. In a broader sense, this generator beseeches us to more carefully examine written works, asking ourselves if they truly express new meanings and ideas or if they simply replicate the stylistics we lend credence to.

References

Accessing Text Corpora and Lexical Resources." n.d. Accessed May 15, 2021. https://www.nltk.org/book/ch02.html.

Austen, Jane. 1994a. *Persuasion*. https://www.gutenberg.org/ebooks/105.

———. 1994b. *Northanger Abbey*. https://www.gutenberg.org/ebooks/121.

———. 1994c. *Mansfield Park*. https://www.gutenberg.org/ebooks/141.

———. 1994d. *Emma*. https://www.gutenberg.org/ebooks/158.

———. 1994e. *Sense and Sensibility*. https://www.gutenberg.org/ebooks/161.

———. 1998. *Pride and Prejudice*. https://www.gutenberg.org/ebooks/1342.

bot_developer. 2020. "Daily Financial News for 6000+ Stocks." Kaggle. 2020. https://kaggle.com/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests.

Boyle, Tara. 2018. "Medical Transcriptions." Kaggle. October 15, 2018. https://kaggle.com/tboyle10/medicaltranscriptions.

Broughton, Gary. 2018. "Star Trek Scripts." Kaggle. 2018. https://kaggle.com/gjbroughton/start-trek-scripts.

"Csv — CSV File Reading and Writing — Python 3.9.5 Documentation." n.d. Accessed May 15, 2021. https://docs.python.org/3/library/csv.html.

Francis, W. N., and H. Kucera. 1979. "Brown Corpus Manual." 1979. http://icame.uib.no/brown/bcm.html#tc.

Iderhoff, Nicolas. (2016) 2021. *Niderhoff/Nlp-Datasets*. https://github.com/niderhoff/nlp-datasets.

Weizenbaum, Joseph. 1966. ELIZA—a computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (Jan. 1966), 36–45. DOI:https://doi.org/10.1145/365153.365168

Kuchling, A. M. n.d. "Regular Expression HOWTO — Python 3.9.5 Documentation." Accessed May 15, 2021. https://docs.python.org/3/howto/regex.html.

McKinney, Trenton, and anselm. n.d. "Python - How Can I Iterate over Files in a given Directory?" Stack Overflow. Accessed May 15, 2021. https://stackoverflow.com/questions/10377998/how-can-i-iterate-over-files-in-a-given-directory.

Neekhara, Aman. 2019. "Convert JSON to Dictionary in Python." *GeeksforGeeks* (blog). December 12, 2019. https://www.geeksforgeeks.org/convert-json-to-dictionary-in-python/.

"NumPy User Guide — NumPy v1.20 Manual." n.d. Accessed May 15, 2021. https://numpy.org/doc/stable/user/index.html.

Pungas, Taivo. (2017) 2021. *Joke Dataset*. Python. https://github.com/taivop/joke-dataset.

"Python File Open." n.d. Accessed May 15, 2021. https://www.w3schools.com/python/python_file_open.asp.

"Python Programming/Operators - Wikibooks, Open Books for an Open World." n.d. Accessed May 15, 2021. https://en.wikibooks.org/wiki/Python_Programming/Operators.

Python, Real. n.d. "Python Command Line Arguments – Real Python." Accessed May 15, 2021. https://realpython.com/python-command-line-arguments/.

"Python RegEx." n.d. Accessed May 15, 2021. https://www.w3schools.com/python/python_regex.asp.

"Python Split() Function: Learn to Split String Variables." n.d. Accessed May 15, 2021. https://www.bitdegree.org/learn/python-split.

"Re — Regular Expression Operations — Python 3.9.5 Documentation." n.d. Accessed May 15, 2021. https://docs.python.org/3/library/re.html.

Rizvi, Mohd Sanad Zaki. 2019. "Language Model In NLP | Build Language Model in Python." *Analytics Vidhya* (blog). August 8, 2019. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-language-model-nlp-python-code/.