MAY 15, 2016

NLP SENTENCE GENERATOR

GENERATING SENTENCES USING A TWEET CORPUS

JOSH WEIN
CSE 390 – NATURAL LANGUAGE PROCESSING
Final Project Report

Table of Contents

1 Introduction

For the final project I implemented a sentence generator using NLP techniques and strategies I learned during the semester as well as new ones I implemented to help with the process. The application is backed by a database of tweets with 604,862 tweets spread out into 273,010 topics. When I came up with the idea for this project it was to see if it was possible to generate at least pseudo-sensible sentences using samples of tweets around a given topic.

1.1 Uses

While this application does not provide a high accuracy model of grammatical correctness, it gives some insight into the similarities of different individuals' way of speaking. By taking all of the sentences surrounding a certain topic, it's possible to generate new sentences that can be passed off as a human generated sentences. This application gives an insight into the base layer of different AI techniques used in chat-bots and personal assistants. It's a good starting point for expanding towards more grammatically correct sentence generation and other uses.

1.2 Input and Output

Sentence generation is the process of outputting sentences in a human readable form given some input of some corpus of information that can be parsed and analyzed. To give the application a better form of structure it uses two separate inputs, a user chosen topic, and a precompiled list of sentences and topics in a server-side database. The details and structure of these inputs are discussed in more detail in Section 2: System Description. Using these two inputs, the application creates MLE based language models with which it randomly generates sentences using rejection sampling.

1.3 Example Outputs

To give an idea of how well the application can work here are a couple of examples of good output:

Input: *food* Output: *Attempting to clean a magasine named Sirene. Exiting! Soooon food! Haha.*

Input: *school* Output: *Seriously, school tommorrow... nooooooooooo!!!!!!!!*

And some bad output:

Input: *school* Output: *In I.T. Bored to start babysitting for school... all good mood. My good bye seniors*

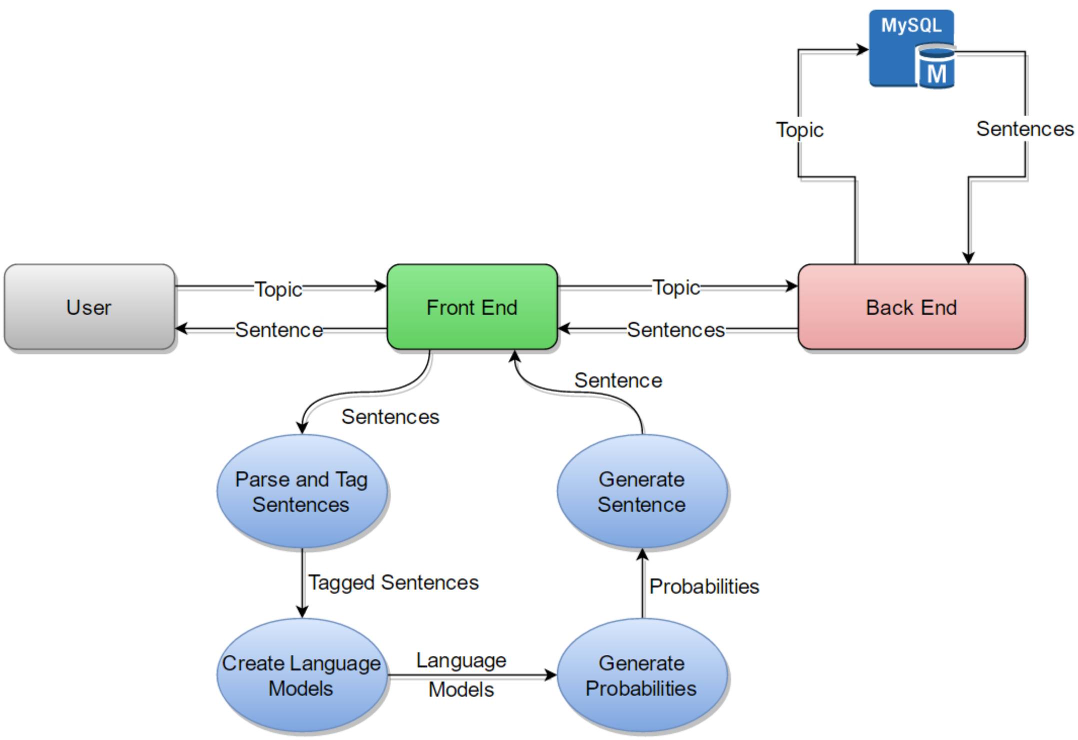
Input: *rain* Output: *good want rain. Why rain - back in. I'm mad man Rhode Island tonight. . uh oh, it looks to wait to rain... going scuba diving tomorrow.*

2 System Description

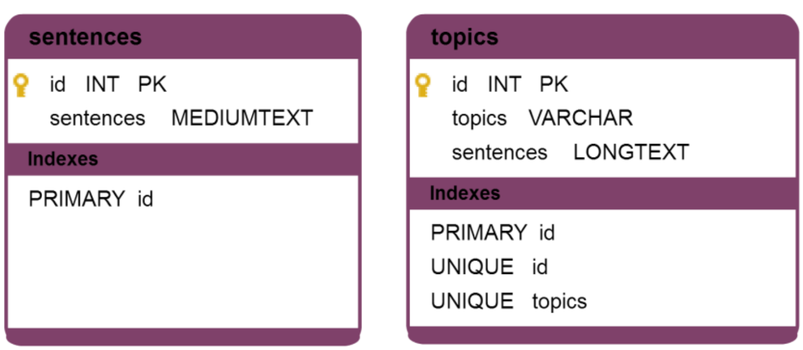
This section covers the specifics of how the application actually functions. The first part contains diagram overviews of the application and the second part discusses the steps taken by the application to generate the sentences. The way the database was created and how the data was initially parsed is not discussed here, but is discussed in more detail in Section 5: Discussions and Conclusions.

2.1 System Diagrams

2.1.1 Overall Application Layout



2.1.2 Database Table Layouts



2.2 System Overview

In order to use the application all the user needs to enter is a topic. As long as it's one word and shorter than 50 characters, the topic gets sent to the server. The server then checks the database for the topic and if it exists in the database, all of the sentences for that topic are pulled and sent back to the client front end.

The first step in the sentence generation is preparing the tweets. Each tweet is parsed, any empty tokens are removed, and starting and ending tokens are added to it [<s>, </s>]. Once every tweet has been parsed, the list of sentences gets sent to the next routine to generate the language models. The language models generated are unigrams (single-word) with counts and bigrams (double-word) with counts. The language models are used to create an MLE based probability model.

The final routine takes the bigrams and the probabilities to generate sentences using the probabilities as weights for rejection sampling. Starting with the start tag ('<s>'), a list of possible bigrams are generated and chosen from to create the sentence until it reaches the end of the sentence. This final routine will continue running until a sentence is generated before the upper limit on length and it contains the topic. It does this by checking for an ending tag ('</s>') after each word generation.

3 NLP Techniques Used

Three main techniques were used in implementing this application: MLE Probability Modeling, Rejection Sampling, Information Retrieval, and a back-off implementation for handling the random value.

3.1 MLE Probability Modeling

The maximum likelihood estimation model is a statistical model used for getting the likelihood of certain observations. MLE's simple equation gives the highest probability for training data. MLE is usually an unsuitable method for most NLP related tasks but we don't deal with any unknowns in this application; therefore MLE doesn't cause any problems for us.

The equation used for MLE is:

$$\frac{count(bigram)}{count(unigram)}$$

For example with the bigram "go home" the equation would be:

$$\frac{count("go\ home")}{count("go")}$$

This results in an even probability distribution across the board making it easy to filter out lower probabilities if needed and choose words.

3.2 Rejection Sampling

Rejection sampling is a technique used to distribute probabilities over a graph while keeping the density function the same allowing for a random variable to be sampled easily. I used this idea to create a table populated by the bigram indexes based on their probabilities to give them an accurate weight distribution.

3.3 Information Retrieval

Information retrieval is the process of finding information from somewhere that is stored in an unstructured way. I had to parse through hundreds of thousands of tweets and store them based on their topic to create the database. I also had to create an indexed document-class structure for storing the data to be able to access it for the user queries.

3.4 Back-off

I used the back-off technique we learned about earlier on in the semester to help create a routine to get the next random word. The user is able to specify a general randomness in the application from a scale of “Less Random” to “More Random”. The randomness specifies what probabilities to include or exclude from the calculations. If the rejection sampling does not yield any results then we back-off the randomness, by lowering the allowed probability until we get results to choose from. This means that that we always have choices no matter the randomness chosen.

4 Evaluation

It’s difficult to objectively evaluate the accuracy of this application since there isn’t any baseline for generating sentences. This is also due to the type of data being used; most of the data itself isn’t grammatically correct to begin with. If we were using a data set that was known to be grammatically correct then we could check if the sentences generated were grammatically correct as well. Since that’s not the case, it’s expected that sentences generated will not be grammatically correct.

With that being said, it’s still possible to look at the results and see that it does work to some extent. Some generated sentences seem indistinguishable from a tweet someone could have posted, while others are obviously nonsense. The third category of generated sentences rides the line between the two, in a sort of uncanny valley way, where it just barely misses being seen as human generated content like the sentence: “My fone is on the time on ebay, someone else needs 2 GO SHOPPING RIGHT NOW!!!!” Many of the generated sentences share these similar features where it almost makes sense, and the sentence structure works, but there’s obviously something off in the content.

5 Discussion and Conclusions

5.1 Project Challenges

The first challenge working on this project was getting the initial set of data. My first thought was to use the Twitter API to pull either live streams of tweet data or specific hashtags of data. This way I could be constantly building up a larger and larger database of tweets and when a user enters a topic that doesn't exist I could make an API call to get some more tweets to use. The Twitter API ended up having a lot of limitations such as only being able to make 180 API calls every 15 minutes. This comes out to 1,200 tweets a minute which is a decent amount but could be a much bigger issue if multiple people were using the application at once.

Luckily I found an excel file containing around 1.3 million tweets from Sentiment140. I was able to convert it to a text file and wrote a small Java program to strip all the extra content from each line to get a plain text file containing all of the tweets. This led to the next issue of how to store the data in the easiest most efficient way possible.

I could have just created a table with two columns, one containing a topic and the other containing every sentence that matched it. This would waste a huge amount of space since the sentences would have to be duplicated for every single topic. To fix this I instead gave each sentence an id number and stored it in a different table. I then matched the sentence id to the topic which ended up with a table looking like this:

is	1, 6, 9, 15, 17, 24, 25, 30, 34, 38, 39, 48, 51, 54, 55, 62, 68, 71, 81, 86, 90, 96…
upset	1, 263, 508, 581, 848, 1593, 1722, 1848, 2881, 2964, 3091, 3890, 4521, 4718…
that	1, 14, 17, 23, 23, 30, 45, 49, 68, 70, 82, 95, 126, 127, 128, 134, 136, 161, 161…
he	1, 154, 211, 231, 236, 236, 254, 281, 281, 284, 387, 394, 479, 576, 595, 730…
can't	1, 22, 67, 100, 106, 115, 130, 138, 143, 212, 264, 312, 349, 369, 369, 418, 420…
update	1, 221, 457, 794, 1002, 1116, 1286, 1363, 1475, 1486, 1493, 2217, 4113, 4257…

This was a lot easier to store and access since I could just grab the list of ids for a topic and grab every matching sentence from the second table.

5.2 Results

I think for the most part the results were a success. The quality of the sentences could be improved and an even larger dataset would help with that. This could also be transferred over to a different base dataset like Shakespeare and would likely generate more elegant sentences.

5.3 Application Enhancements

There are a couple of enhancements I would make to the application to try and increase its usefulness and capability. It has many places where it could be expanded upon.

1) Add different probability models and see how that affects the sentence generation. It would be interesting to compare and contrast MLE with Laplace or Katz Back-Off models.
2) Support for different n-gram options. Usually higher n-gram models give higher accuracy with prediction models so it would be interesting to see how generating with trigrams or quadrigrams would affect the sentence generation.
3) User entered data: Implement a way for the user to enter their own data to use to generate a sentence.

5.4 Conclusions

I enjoyed putting this together and learned a lot about language structure as well as how difficult it is to accurately model human language without extra input. Hopefully, I'll have the time to go back and implement the enhancements I mentioned earlier.

Resources and Referenced Material

Application Hosting for Frontend and Database – OpenShift: https://www.openshift.com/

Information Retrieval - http://nlp.stanford.edu/IR-book/pdf/01bool.pdf

Rejection Sampling - https://en.wikipedia.org/wiki/Rejection_sampling

Tweet Data – Sentiment140: http://help.sentiment140.com/for-students/

Twitter API - https://dev.twitter.com/

Twitter API Exchange Wrapper - https://github.com/J7mbo/twitter-api-php