Final Report

DS-5460 Big Data Scaling

Author: Preston Abraham

I. Introduction

This project is the development of a text generation model using n-grams to simulate a review of a Disneyland park. This is particularly interesting for a few reasons. Firstly, we've had quite a bit of experience in training models to recognize false data, especially false images, but we haven't gotten to approach the problem from the side of the adversary attempting to generate fake data that escapes detection by the filtering models. Secondly, the data is interesting and the results of the text generation can be quite entertaining. For example, the text that the model generates to simulate a review with a score of one out of five can often be quite funny to read. The use of n-grams to do text generation is not a new idea. Early forms of the Google search bar used this method to suggest queries that a user might want to make based on the first few letters or words that they type, and phone companies such as Apple and Android make use of this method for their Auto-Complete functionalities that suggest words as a user types. The goal of *this* project is to use those same methods to generate reviews word-by-word in an attempt to create realistic texts that could pass as real reviews left by a visitor of Disneyland.

II. Problem

This problem is more of a data analytics problem than a modeling problem, although the development of a basic model is a necessary component. Effectively, the problem is to use the reviews provided in the data to generate fake reviews that look similar enough to the real reviews that a discrimination model would not be able to tell the difference. To do this, the data provides several columns: review score, review date, park visited, reviewer nationality, and review text. From these columns, the objective is to return just the text and score so that the generated review could hypothetically be posted.

III. Methods:

(10pts)Describe the algorithm or approaches you designed
The tools and software used to implement the algorithm
What problems did you face?
How did you solve them?

The methods used for the eventual solution are two-fold. The first aspect of the project is the text-generation algorithm itself using n-grams. For clarity, n-grams are defined as all of the combinations of n number of continuous words in a sentence. For example, for the sentence 'I have a small dog' would be:

1-grams = [‘I’,’have’,’a’,’small’,’dog’]
2-grams = [‘I have’,’have a’,’a small’,’small dog’]
3-grams = [‘I have a’,’have a small’,’a small dog’]
4-grams = [‘I have a small’,’have a small dog’]
5-grams = [‘I have a small dog’]

This process can also be done on the character level instead of the word level, but for this project we will use the word-level definition for constructing sentences. The first step in the process of building this algorithm is getting the data in the right form to be learned. In general, the idea would be to get the list of n-grams from each review, but for a dataset where the words used greatly depend on one of the columns (rating score) it is beneficial to split the data into one set of reviews for each review score. For example, all reviews giving a score of one out of five go into one dataset, scores of two into another, and so on. Then for each of the five datasets, we split each of the reviews into its n-grams, adding a word [‘Start’] at the beginning of each review to designate that it is the start. This leaves us with a list of n-grams for all reviews with each score. This is all we need to begin generating a review. To do this, the first thing that we do is begin with the word [‘Start’] that we added to the front of each review in the beginning. Then, we take all the 2-grams of the reviews and choose a random one that has the word [‘Start’] as the first word. Because the same n-gram may be added to the total list more than once, a ‘likelihood’ is induced for each word based on how many times it appears across all of the reviews. Once we pick the first word to be added from the 2-grams, we add it to the review and look at all of the 3-grams with that have the chosen 2-gram as the first two words, this process is repeated until we hit our Max_n parameter, at which point we stop looking at the entire review to generate the the next word and only look at the most recent Max_n words and select a random (Max_n+1)-gram from the total list of n-grams and append the last word to the review. We stop the process when the character limit is reached, adding punctuation to the last word to simulate a sentence. This gives us a full review text as well as the review score which was used to determine which list of n-grams to use.

The second component of the project was to develop an application that allows for the request of a generated review of any score from one to five. To do this, we first process the data as before, saving each list of n-grams from each review score and saving them to their own files. This allows the app to take the review score as an input and load the correct reviews. The app can then get all of the n-grams for n in the range from two to twenty and follow the above process to generate the text for the review. In the current implementation, the model is trained every time the user wants a review in order to allow the changing of parameters like review_length and max_n which leads to slow runtimes, but in further development the models could be pre-trained, saved, and loaded to ensure near-instant generation of as many reviews as needed.

IV. Results

The raw results of the project is an application that can create reviews that on first glance do appear to be real reviews of Disneyland parks with a given review score. There are some grammar and spelling mistakes, but that is more an artifact of the actual customer reviews the mosel learns on having those same mistakes. A deeper evaluation of the results is difficult to do. Ideally, the best way to test the system would be to run the discrimination models that websites use to identify fake reviews against a combined dataset of a holdout of the real reviews plus an equal number of my generated reviews to see how well my system does at avoiding detection. Unfortunately, websites do not freely allow testing against their systems for obvious reasons. This leads to two potential methods of evaluation. First, we could train our own discrimination model to attempt to identify the false reviews. Second, we could assess the generated text samples manually by giving them a grade based on how real they feel on a scale from one to three with one seeming to be from a person and three being obviously fake. Then we go through a set of fake AND real reviews giving them a score, then see how the mean scores that the fake and real review got differs. In the end however, the dataset is simply too small for these sorts of analysis. Because we split the already-small review subset into five further sub-groups, there are not many reviews causing for example some five-word n-grams to only appear once, meaning that every time those five words are generated the sixth word is **always** the same. To really evaluate how well the app does at disguising itself as real, we'd like to have hundreds or even thousands of times more reviews. Still, the app serves as an interesting proof of concept.

V. Conclusion

In conclusion, this project has been an interesting first step into the world of adversarial data science. Using an interesting method of text generation using the probability that each word appears after certain words or sets of words, we can generate reviews that certainly *look* like real reviews that people would write. In the future, it would be interesting to dive deeper into finding how well our system can trick bot-detection algorithms and how much better it could do if allowed to learn on a much larger set of reviews.