μText

Group 114

Jake Writer, Bill Xu

Abstract

N-Gram, LLM, and Word2Vec models that generate text segments based on given texts. The models fuses different writing styles together and uses vocabularies from both input texts to generate the output.

Introduction

We want to experiment with generating texts that fuses together different writing styles. For example, given Harry Potter and the Bible, we want to generate a story that uses dictions from both books and tells a story of Harry Potter in Bible style. We implemented a baseline N-gram model, an improved N-gram model, a model based on LLM, and a Word2Vec model. The baseline N-gram has the worst output with no coherence, the improved N-gram model has better output with sentence level coherence, the LLM model has the best output with a complete story with introduction, rising action, climax, falling action, and resolution, and the Word2Vec model produced moderate outputs with limited words substituted compared to the original text.

Methods

We implemented four models using three different methods – N-gram, LLM, and Word2Vec

1. N-gram

Baseline Model:

The baseline model is similar to the one implemented in Homework 5. This model preprocesses the text by cleaning it then adding in certain symbolic tags (like <unk> and <e>) to help in the construction of sentences. It emphasizes the use of longer n-grams for more precise context matching and introduces randomness in word selection to more "creatively" generate output. This method prioritizes accuracy through longer contextual matches (it will sound more coherent ideally) and ensures variability in the generated text. We did some grammatical parsing, but there are still some capitalization and punctuation issues.

Improved Model:

The improved model takes in two different texts, the N-gram size, and the max number of words to generate. After preprocessing the data, it constructs a dictionary, where each N-gram is mapped to a list of all words that may follow the N-gram. Words in the list can be repeated, so words that occur more frequently after the N-gram would also occur more frequently in the list. When generating the output, the model first chooses a random N-gram that starts with an English word, and repeatedly randomly chooses the next word based on the previously populated dictionary. When an end of sentence is reached, the model appends a period to the sentence and randomly chooses another N-gram that starts with a word to be the beginning of the next sentence. The model then postprocesses the output by combining parts of words that are split up due to tokenization during preprocessing (eg. do n’t, I ‘ve, you ‘re), combining single punctuations with the end of previous word (eg. however , to however,), and adding capitalization.

2. LLM
The LLM model utilizes OpenAI’s API to generate a story. The model first generates an outline of the story with exposition, rising action, climax, falling action, and resolution. It then prompts the LLM to generate each section of the story based on the last section, and in the end combine them to form the complete output that includes all five sections of the story. We also have the ability to control API costs by deciding to either use a more advanced model for better results, or a simpler model (3.5-turbo).

3. Word2Vec
Unlike the other models, the goal of our Word2Vec model was to transform an existing text to match the styling of another. This was to be accomplished by swapping out words with similar encodings between the two texts. If we wanted to “Shakespearify” a book like “The Great Gatsby”, we would end up replacing new english words with old english words of the same meaning. This is done by collecting all the vector encoding for words in both books, then when we find a word with a similar encoding that is in the Shakespearean book, we swap it in. For example words like “your” could be replaced by their Shakespearean counterpart: “thy”. Our model works by using Word2Vec to find similarly encoded words between both books, then replacing them. Ultimately this model was less successful than the others; as it turns out, the way in which words are encoded with word2vec makes this much more difficult. This simple approach does not cut it. Using our above example, when we try to find words similar to “your”, “thy” doesn’t even make the list of the top 20 most similar. Because this was a smaller aspect of our project, we decided to conclude our exploration into this method here. Given more time, I would have liked to look into the vector differences between these old and new english words and perhaps that difference could be used to translate these words in a meaningful way.

Data
The data we used include full length texts of the books Harry Potter and The Prisoner of Azkaban, Atlas Shrugged, and Macbeth. However, our models are designed to work with any text input.

Preprocessing:
Baseline N-gram model:
The preprocessing steps for the baseline model include splitting given text to sentences based on line breaks and tokenizing the sentences using nltk tokenizer after converting them to all lowercase. We also produce a vocabulary list of words containing only words above the

minimum frequency threshold, and replace words in data that do not appear in the vocabulary with the <unk> marker.

Improved N-gram model:
In the improved model, we split sentences based on periods and line breaks. The sentences are then stripped of some unwanted punctuations like ``, –, and '', converted to lowercase, and tokenized using nltk tokenizer.

Results

Sample text generated by baseline model (text: Prisoner of Azkaban, N-gram size: 3, word limit: 100):
About it, but he agreed. the dementors were closing in, barely ten feet from them. he looked at ron. the whole of gryffindor house took up the stairs for harry to come down and get ready to welcome their best men in azkaban -- the spy, sirius black! black's face contorted. the dementors were closing in on sirius. i don't want to hear explanations, said lupin shortly. harry, ron, and, to ron's disgust, percy. the door of number twelve was ajar and percy was shouting. harry, ron, and hermione. i'm afraid he won't be there for another couple of hours.

Sample text generated by improved model (text: Prisoner of Azkaban + Atlas Shrugged, N-gram size: 3, word limit: 100):
Declare: I am proud to pay tribute tonight, said snape, his face was distorted by terror, eddie's remained sternly serene. The newspapers call a frozen train. While seeing it for a moment, brief and sudden like an electric bulb glaring in the cave of a subway entrance and a lighted billboard advertising soda crackers on the black situation, I was the president of the atlantic southern's gross income ahead of other taxes; the california oil companies had gone out for that purpose. Reserved exclusively for missions of public need. Sparkled on the windows of every house in sight. Wanted the sight of the country

LLM generated segment (first two paragraphs)

In a society shrouded by the omnipresent surveillance of Big Brother, where the air buzzes with the static of propaganda and the streets are lined with posters proclaiming loyalty and love for the ever-watchful eyes, there exists an anomaly. At the edge of this world lies the majestic mansion of Jay Gatsby, a beacon of opulence and a whispered symbol of resistance against the totalitarian regime. The mansion, with its lavish parties and the air of mystery that surrounds its owner, sits like a palace in a wasteland, drawing curious glances from those who dare to dream.

Winston Smith, a low-ranking member of the Party in this dystopian world, lives a life marked by monotony and fear under the watchful eye of Big Brother. His days are spent altering records and rewriting history to suit the propaganda of the regime, his nights haunted by dreams of rebellion and freedom. A grey existence, devoid of color and hope, until a peculiar piece of cream-colored card slips through the mail slot of his door one evening.

Discussion

The output generated by the baseline model is a little incoherent and lacks complete punctuation to properly separate out parts of sentences. Previously, we implemented this model with far too much randomness by allowing for the model to choose words that don't make sense (although this was much less likely). The issue with this is that even though the top few words may have a way higher probability than the rest, because there are so many words, all of those probabilities added together meant that it was actually more likely a bad word would be chosen. We have since corrected this approach in the baseline model.

In comparison to the baseline model, the output of the improved model is more coherent on sentence level. The addition of punctuations also makes the sentences feel more logical. However, because we start each sentence with a random N-gram, there is no consistent logic between sentences.

The LLM model produced the best result, as the story it generated has a complete plot and is very coherent within sentences, between sentences, and between sections. The better performance compared to N-gram is to be expected, as N-grams can only capture local context, while transformer based LLM can focus on different parts of the input and capture long-term dependencies.

References

https://github.com/aparrish/rwet/blob/master/ngrams-and-markov-chains.ipynb

Appendix

Baseline model: n-gram-model.py
Improved model: markov_chain.py
LLM model: llm-based.py
Word2Vec model: WordToVecSubset.py