Benny Enriquez, Lily McAboy, Alex Miller
Professor K. Arnold
CS376-B
4/30/2024

Suggesting, Summarizing, and Sensing: Helping Turn Random Speech Into Coherent Text

I. Introduction

The team further developed Professor K. Arnold's Hugging Face text application that takes an inputted document and rewrites the document to fulfill a user's prompt. We wanted to extend the application further, by allowing a user to choose what tone of writing the document should be written in; for example, a user could select to make their document more concise or more complex.

In order to start the process, we started rewriting Professor Arnold's application. The application used software libraries to create a browser application. The original app allowed a user to choose their desired model, but since we want this to be more user friendly to those unaware of different language models, we chose to set the model as Google's Gemma-1.1-2b instruct.

We implemented a dropdown menu that allows the user to choose the tone that they wanted their document to be changed to. The input text and resulting text remained the same as the previous application. Instead of a user prompt text box, the chosen tone is used as the prompt, which allows the user to not have to specify what they want after choosing an option. This prompt is put into the model along with the text.

The second implementation is a summarize button. We realized that a user may not want to rewrite a document, but rather understand what they mean. This summarize button iterates over all of the input text and returns a summarized version of the text for easier user readability. This addition also helps with our third implementation.

We implemented a function that allows the user to be able to input text and then get a detailed response of what the tone of their text is providing. This would be helpful when drafting an email to a supervisor or a professor, to make sure it is professional and not demanding. It would also help if one were to draft a note for a friend, to not come off as offensive or too vague.

Some smaller contributions featured prompt engineering, in which we created a few buttons that could give suggestions on how to shorten or lengthen a particular document. It also had another button that allowed one to receive how they should write a response. For example, if a person wrote a very condescending email and the user wanted to know how to respond, the model would give a suggestion on how to respond.

When we were beginning our project, we thought about what we wanted to accomplish. We believed that accessibility can be something that limits a person's ability to communicate. One of our biggest goals was to write an application that would help somebody understand text and tone, especially for people who may lack some emotional intelligence or are at a cognitive level in which they need help understanding something. For example, freshmen in college may not have experience reading academic papers, and it can often seem overwhelming when reading a peer reviewed journal for the first time. A summarizer can help someone understand what the text is saying so that they can still gain knowledge but in a more efficient and less overwhelming manner.

II. Modeling, Datasets, and More

As said before, We chose to use Gemma 1.1 2b instruct. Gemma is a pre-trained language model, so we did not have to train a dataset, which allowed us to be more efficient when writing our application.

The algorithm for choosing a tone is as follows: the user chooses a desired tone. This tone, written as a string, is combined with a prompt ("Rewrite my text to be "), to create a prompt like, "Rewrite my text to be more complex." The prompt is tokenized and put into the model so that the model takes the user document and applies the prompt to it.

When choosing to summarize a piece of text, the function to summarize the text tokenizes the text and then applies the model. The function contains an option to choose the number of beams desired for the text. Beam search is a heuristic algorithm that looks for the most likely sequence of words. Though we later used prompt engineering to simply ask our model to summarize the text, we implemented this heuristic algorithm for the model to give an emoticon along with the response. This emoticon displays a range of emotions, such as angry, worried, disappointed, love, fear, confusion, happiness, and much more.

Reading the tone, summarizing the document, giving suggestions on adding and subtracting sentences, and the response all used some sort of prompt engineering. Each button was programmed to call a function that performed whatever prompt given. For example, the button that said “Suggest -” had called the perform function with the prompt “(in two or 3 sentences) suggest some ways to expand upon the document.”

III. Code

We started our code using a baseline that allowed us to tokenize the input text and then adapt it based on Professor Arnold’s Hugging Face application and the prompt engineering lab from CS376. After getting a basic understanding of Streamlit and the libraries we planned to use, we started to tear our code apart to further refine our code towards our specific goal.

We believed that it would be difficult to measure “exact” accuracy from our results, as we don’t have a predestined result to test it on, and even if we did, it would be difficult to capture the accuracy word for word. We evaluated accuracy based on if the results were viable or not. If a user wanted their prompt to be more complex, the results generating a simpler text would not be the correct case. This would be an example of not being accurate. If the user wanted a more sarcastic tone, and the resulting text was condescending and mean, then we would mark that result as correct. We evaluated a few documents: old papers we’ve written for other classes, convoluted, peer reviewed journal articles about mathematics, text messages sent to friends, and more. These were each chosen to evaluate how well our model could summarize, understand, and rewrite text given a variety of options.

Overall, our model works very well; we believe that it is less about our prompt engineering and more that we used a pretrained model (Gemma), which did not require a lot of (if at all) fine-tuning. The whole group ran multiple examples of texts into the model separately and definitively agreed that the model was able to do its job properly.

One way that we could technically “measure” our model’s success was to compute the loss of our next generated tokens. When we put it through our model and asked it to rewrite the text to be more flirtatious, an example input was “Dear Benny,” (the start of an email), which translated to “My Benny,” (significantly more romantic). The next token was computed and the original token’s loss and the loss ratio, which is the token loss divided by the highest computed loss, is shown. These also were represented in different colors, where the words’ colors change

based on how similar the token is to the most likely token. If the original token and the predicted next token are extremely different, the color is more blue/purple. If it's a moderate or smaller difference, the word is red.

IV. Application

We used the Streamlit library to create an app that allowed us to deploy it on Hugging Face Spaces. This included a dropdown menu that allows a user to choose how they want their text to be rewritten, or if they desire any summarization or suggestions.

V. Conclusions and Results

We realize that there are limits to our application. One crucial limit that our project has is that we felt as though our model was slow. We had to use and test different GPUs, but in general, running our model took a long time. We also didn't use a dataset to fine-tune our model. A better practice would have been to write our application to train on different formalities to "learn" what each formality truly means. Even though our model did well, there could possibly be some improvement. If we were to expand this application again, we may change some of the features in order to summarize and explain more technical subjects, such as math or general programming, but this would involve changing prompts and getting rid of the tone recognizer.

Though three of us tested our application and believed that it worked properly, it may not work for everyone. In order for our prompt engineering to be better tuned, it would've helped to ask people who were not computer science majors or in the realm of AI/ML in order to gauge if the application is helpful and useful. Regardless, we hand tested approximately 20 input documents across forms of mediums of communication (email, text, speech, essay) to judge the usefulness of the results, and found that gemma 2b almost always gave "correct" results. Summarization always works; we never once had an incident in which it didn't perform correctly. However, it did seem to struggle with the audience of the response description. In those cases, the results were still usable, but not as intended. Overall, we found that these results were useful around 70 to 80 percent of the time, otherwise the results were simply too broad or vague to be considered useful (for example, it favored the phrase "expand upon details" when asked to suggest ways to lengthen the input). In addition, we also briefly tested the 7 billion parameter version of the model, and found that the results were generally more useful, but not greatly so.

Overall, we decided that the 2 billion parameter version was indeed the correct choice, as it's lightweight computational requirements and speed were worth the tradeoff.

One particular challenge that we faced was that Hugging Face Spaces, for some reason, would not give Lily authentication for Gemma 2b and the final was deployed on her Hugging Face profile. Benny had to use his own machine to write code and transport it to the rest of the team.

From a Christian worldview, we believe that this application helps achieve our goal of accessibility for those who may need help understanding and refining text. As stewards of the earth and Christians, it is our job to look out for those who may be in need. The book of Philippians says, "Do nothing from selfish ambition or conceit, but in humility count others more significant than yourselves. Let each of you look not only to his own interests, but also to the interests of others" (Philippians 2:3-4 ESV). As a group, though we may feel confident about our analytics skills or writing abilities, we shouldn't neglect useful and helpful tools for people who would greatly benefit them. We hope to promote good communication, harmony, and thoughtfulness with our project.

We do realize that this application could be used for wrong, such as cheating on an assignment or skipping work because it seems easier. It also may be argued that we are promoting lazy work, as the summarizer isn't challenging people. We understood the risks that come with AI; we learned in class that many people don't use AI as a tool, but rather to their advantage to plagiarize or take "the easy way out." We can't necessarily block people who will use this app maliciously, but we can hope that people will have more integrity when using our model. It also does help that we included a small "fail safe" mechanism, in which users cannot have the app rewrite the entire essay, but rather compute next tokens to help the user get a head start or just receive suggestions. This helps a "copy and paste" method that ChatGPT allows, which can be a source of a lot of cheating.