Hackathon Report: Gricean Maxmemes

IIIT-H Confessions Generator based on Neural methods

Dataset

We scraped ~21k Confessions from Facebook pages of various US Colleges, using a Selenium Bot. This was done in reference to this project on mental health. This process took time due to the anti-bot features of Facebook which we circumvented using timeouts and random behavior. This data had to be majorly cleaned as it was not in the UTF-8 Encoding standard we were basing our model on. Thus, we used Pandas to read the csv and encode it in UTF-8, and then only use ASCII characters from the data left. We ran multiple regular expressions on this model for cleaning, and getting read of residual encoding errors, URLs, and so on. Further on, to avoid dealing with the complex challenge of code-mixing, we used a modified version of the NLTK English Language Wordlist, adding text-speech to properly illustrate college language, and then created a Non-English Rate, and removed all sentences with N-E rate above 30%. This was chosen as several words like “sooooo” or such were detected as N-E, and we wished to preserve them. This brought our dataset down to 13k sentences. This data was stored as quoted strings due to errors from automatic delimiters in CSV renderers such as Libre. We used regular expressions to "nativize" our data for consistent content words (such as IIIT instead of all colleges). This was done by checking the data and regularly running it through various such scripts.

The Neural Network && The Database Generation

We have used a LSTM model with embeddings to capture character level dependencies. The embeddings layer converts the words into 256 dimensional tensors, which are then fed into the LSTM layer. The LSTM layer takes into account all the previous characters it has seen till now. The speciality of the LSTM cell is that it is able to capture long ranged dependencies as well, as is seen in the case of gender inflections. The short term dependencies helps it get the spellings correct. At the end, we put the returned sequences into a Dense layer, which now predicts the character to be output next. After the model has been trained for a sufficient number of epochs, we start our database generation. We now give the model an empty string to work upon, and it generates character after character. It looks like a sci-fi movie in real time! For one final layer of checking, we pass the database through a python script, and then convert the resultant final data in json format for displaying it on the website.

Website

To display our model, we designed a Vanilla web application, which can be accessed at https://prajneya.github.io/Confessions-Generator/ . This web-app serves as a single page application, where the user can input a name on which they desire the confession to be generated. We do not run the model each time the user wants to generate a confession, since it would lead to a huge consumption on the backend, making the application slower, especially on devices with low-bandwidth internet connections. Recognizing this problem, the model generates a large number of confessions, which are stored in a .json file. The website fetches confessions from this file and uses local browser cookies to remember which confession has already been displayed, thus displaying a new confession each time.