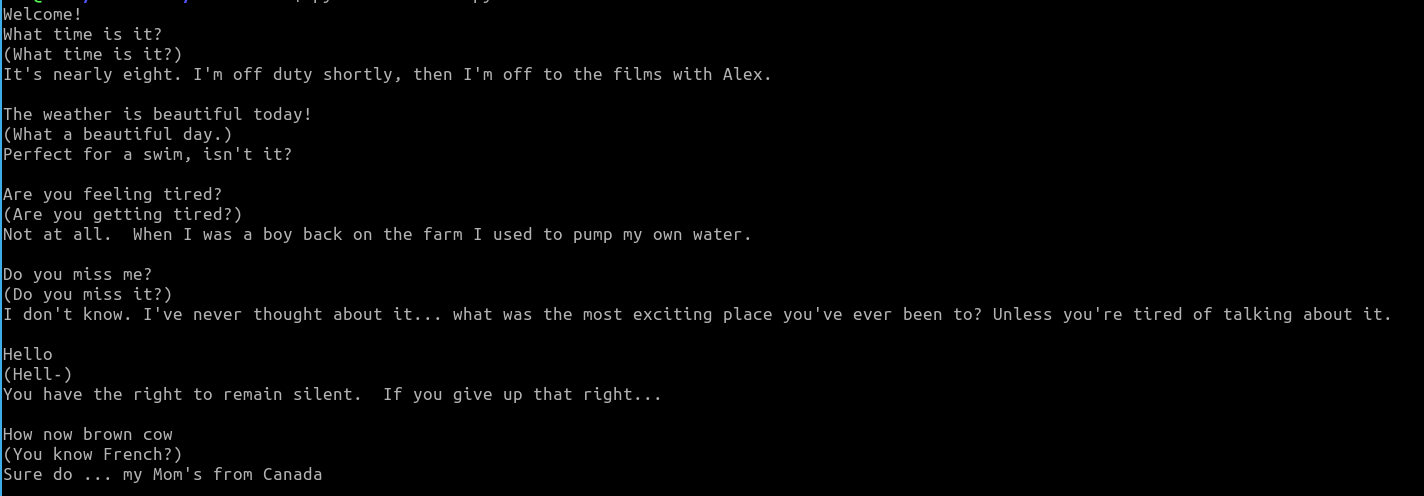
I created a chatbot in Python that takes user input and generates a string output based on a dataset of predefined user conversations. The dataset I used was the Cornell Movie—Dialogs Corpus (referenced bottom). This dataset contains 220,579 conversational exchanges between 10,292 pairs of movie characters, and is the basis for what the chatbot will be referencing when deciding how to respond to user input.

Initially I had planned on utilizing TensorFlow in order to take in user queries to provide descriptions regarding a movie’s genre, rating, and release depending on the user query. However, as I worked more on this project, the combination of realizing the Cornell dataset may not be large enough to effectively train on this specific task as well as the issue of figuring out how to parse all of the movie data in order to use it effectively made me switch objectives in this project. This led to the chatbot seeing a little bit more open domain and less task-specific than I had initially hoped and somewhat delves into the realm of natural language, but it is not sophisticated enough in that sense to be called a true conversational bot; i.e. if one decides to check the movie list in the dataset, one can probably somewhat predict the output if given an input of a conversational piece from one of the movies.

The program begins by reading in the data from the conversation file. All of the “questions” in the dataset are then stored in an array, and all of the “answers” are stored in another array, with the index of both arrays corresponding to each others appropriate question and response. After the parsing process is complete, the actual bot activates and the user is prompted for input. Anything can be entered in order to receive some response, but generally, asking a non-specific question will yield better, more accurate results (i. e. a response that somewhat makes sense). Once user input is taken, the comparison process begins. The user input is passed into a function which first checks what questions in the dataset can be used to measured the Hamming distance with the user input (that is, which questions are the same string length as the user input). Sometimes a very accurate reponse can be given and as Hamming distance is generally much quicker to calculuate than the next method, it is accounted for first. If it is found that the user input matches a question in the dataset closely enough (in this case, I checked for if they were 75%+ similar), then the corresponding answer in the dataset will be printed out. If the Hamming Distance cannot be used effectively, then the Levenshtein Distance is calculated. This effectively does the same thing as Hamming Distanace but can account for variable string length, and it just has a higher time complexity. Once the proper response is calculated, I have it set so that the question that that the bot thinks the user asked is printed in paranthesis immediately below the user input, and then the bot’s actual response is then printed.

A difficult part of this project for me was figuring out how to parse the dataset to actually use it in the program. For this, I did reference a page of a project utilizing the same dataset, and have linked it in the references below. Aside from that, I had intended to implement a neural net to account to give different weights to individual words in a query, but this did not produce the results I had liked. A major flaw with the current state of the program is that a few letters difference in a sentence can change the entire meaning of a question, and the bot does not properly account for that. E.g. “How was your day?” and “How was your pay?” are very similar in terms of characters but have pretty different meanings. To account for this would involve including word vectors which is beyond the scope of what I have here.

Here is some sample output from the bot:



As can be seen, some responses (especially to fairly common, generic questions) are quite accurate, but some queries miss the mark a bit, and for a fairly random query, the response is completely off. A much larger dataset would be needed to produce more consistent results.

The bot requires Python3 and the Levenshtein library installed to work (installabe via pip). Also, a warning that some of the dialog in the dataset is very explicit, and the answers have not been filtered for profanity.

References:

https://www.cs.cornell.edu/~cristian/Cornell\_Movie-Dialogs\_Corpus.html

https://github.com/Currie32/Chatbot-from-Movie-Dialogue/blob/master/Chatbot\_Attention.ipynb