Arjun Bala
Chatbot Project
CS 4395

Sheldonbot

The chatbot I built is based upon the character Sheldon Cooper from the hit TV show The Big Bang theory is a comedy show that revolves around the lives of four scientists in their daily lives. Arguably the most iconic character was Sheldon Cooper. Sheldon Cooper quickly became a fan favorite in the show due to the uniqueness of his character on TV. The character is well scientific mindset, extremely type A personality, and his social ineptitude often leading to complications. The idea I had was to build a chat bot to emulate Sheldon Cooper.

One of the most difficult aspects of this project was the data collection. The first hurdle was finding out how I could get enough data about Sheldon Cooper to create a chat bot based upon him. So, to start I need to find a way to get the script for every Big Bang Theory episode. The first place I looked was Kaggle. Kaggle is a site with the millions of datasets in every topic. When I checked on Kaggle I found one data set with Big Bang Theory scripts, however upon further inspection the data set only had a scene or two from each episode Rather than the entire episode. So, with that data set I wouldn't have had enough data. The next place I checked was Google. On Google I found a script sites. The script site had the scripts for every episode; however, it was on the web page, not in PDF or any other easily manner formatted inputs. So, to build up ideas of web the Wiz is age show 25 episodes of time period then once you click on one of the episodes it would take you to a page with the entire script for that episode. The first thing I wrote was code to grab the script for an episode when given a link. The way I did this was by looking for the paragraph tags in the web page and then getting the text from those paragraph tags. In each of the paragraph tags the site had one line or one stage direction from the episode. So, I put all that text into a list. The next goal I had was to figure out how to get the links to actually grab the scripts from. The first method I tried was trial and error by looking at the urls to get a list of them to pass on into the previously created function. So by going through the first season of the show I noticed that all the urls were the same but ended with a number which incremented by one for each episode script for example episode one had the url ending with 986, episode two had a url ending in 987, and so on and so forth. I initially started by writing my function to have the code go through the urls in numbered order, however after 30 ish iteration I realized that the url numbers were not consistent at all. There were big jumps in the numbers between and there were different shows mixed in using those numbers. So I had to figure out a better way to do so. My new approach was to go through the main big bang theory page and to go through and get the links from that page for each episode, however the issue with that is that there were only 25 links for episodes per page. My remedy to this was to then use the number in the link to virtually go through all the 8 pages of big bang theory episodes. So to crawl, I iterated through each of the main pages of the episodes, got the links, then passed those links to a function which got the script for that page.This worked well until season 8 of the show. After season 8 of the show, for some reason the site stopped listing the character who said each line. However, I had enough data since there were well over 100 episodes at that point so I ended the scraping at season 8.

After getting the list of each and every line in all the episodes up till season 8, I needed to filter the data out into an easier method for viewing and for input into the model. I wanted the end result to be a dataframe that has a prompt from one character in the show and the response from Sheldon. To do this I first had to identify speakers. The lines from the transcript were in the following format:

Character: (stage directions if any) "The character's line"

I broke the lines down into speaker and line first, by splitting on the colon. Next, I iterated through all the lines to find the places where a character spoke to Sheldon, and he responded. I then added them to separate lists and merged them into one to create a dataframe, one prompt and response in each of the rows of the dataframe. Now that we had this dataframe we needed to preprocess and prepare for the data for the model.

To preprocess the data there were a couple big aspects I needed to tackle. The first thing I did was remove the stage directions by using a regex function to delete everything within parenthesis. Next, I removed punctuation, lowercased the text, and removed unneeded spaces.

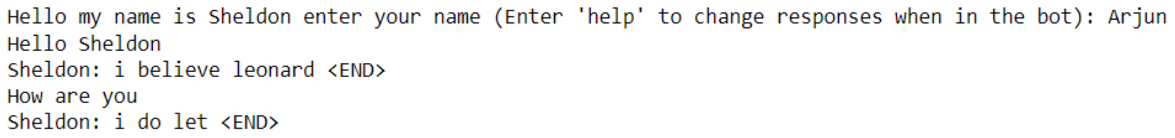
The goal of this project was to take in an input sequence and produce a responding output sequence of words as a response to that input sequence. So, I needed to find a model which could do that. After a little bit of digging around the internet I stumbled upon the seq2seq model. The seq2seq model is called that because it takes in a sequence and outputs a sequence. The way that it works is that the model takes in an input into an encoder, which is a LSTM which encodes the data in a way for the machine to make sense. Then that LSTMs internal weights and states are passed to a decoder which is also an LSTM which decodes it into an output vector.

The next thing I had to do was to prep the data for input into the model. The seq2seq model takes in 2 inputs, the input for the encoder and an input to the decoder and outputs 1 output from the decoder. The second input is exclusive to training and is used to help the decoder train less dependent upon the encoder. The output is the text that Sheldon gives in response to the prompt. To pass all this data into the model I used one hot encoding of the vocabulary to store what word is being passed into the model. The biggest issue I had/have with this model is that Sheldon as a character is someone who uses words which tend to be more unique as he is a scientist and therefore his vocabulary is on par with one. The issue comes up with creating the vectors for the input. Since Sheldon has such a unique vocabulary, I had to replace words that he used once or twice with an <OTHER> token as having all of Sheldon's vocabulary in the vocab balloons the vector to sizes in the double digits of gigabytes and that isn't really possible to train on a normal computer as there is not enough ram to store the vectors. So necessary shortcuts were taken and will be addressed later in the report.

To involve user models in the model what I did was add a help feature for Big Bang theory fans to update the model. Users enter their name in the model at the beginning of the chat and if they use the help feature then the users can help fix the model for them. Say for example the user says, "hello Sheldon, how are you?" and the bot responds with something the user believes that Sheldon would not say. The user can type in "help" and the program will then ask the user for a suggestion for what Sheldon would say. The program then adds that response along with the user's prompt to a dataset with their name and the next time the model is trained for that user the new adjusted dataset will be used.

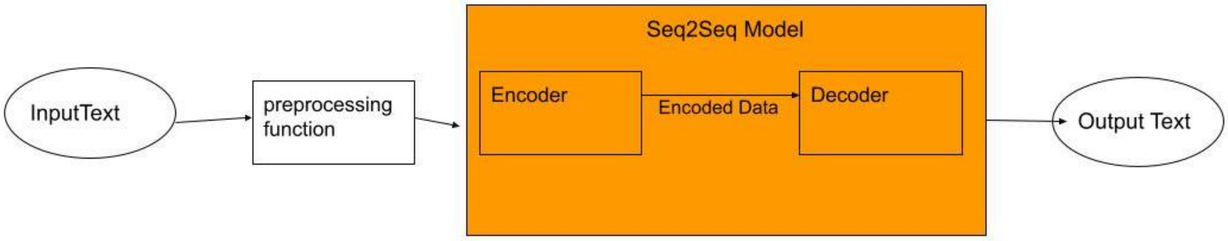
The evaluation I used for model was based on my opinions along with the opinions of some other Big Bang Theory fans I know. The strengths of the model are that it can give some strings of words which sound very Sheldon-like. However, there many weaknesses to this model which will need to be addressed in further iterations of the model. The biggest one I have is the ballooning of the vector sizes due to the uniqueness and size of Sheldon's vocabulary. Due to that I had to make a few shortcuts like reducing the size of Sheldon's vocabulary by removing words that only occur one or two times, but this is an issue as well as much of Sheldon's vocabulary is unique so to counter that I must remove the <OTHER> token as the model chooses that as the next best word quite often. A better solution, that I plan on implementing in the next iteration is to use a generator in order to break the training data into smaller pieces and pass it to the model to train in smaller chunks. Another addition to remedy this situation is to replace one hot encoded vector with word embeddings so that the model gets the meaning behind the words rather than just the word. To actually get the model to run I also created a mini model which works on the smaller set to get sample dialogue, but in my final project I used the final model.

Here I have provided some sample dialog from the model:



```
Hello my name is Sheldon enter your name (Enter 'help' to change responses when in the bot): Arjun
Hello Sheldon
Sheldon: i believe leonard <END>
How are you
Sheldon: i do let <END>
```

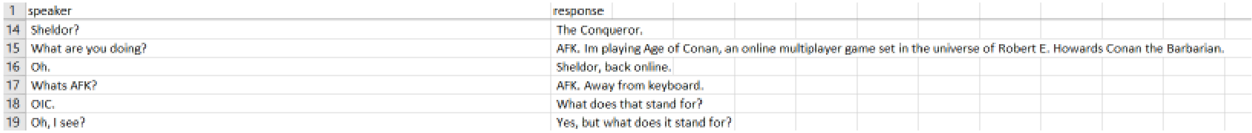
Logic:



Arjun Bala
Chatbot Project
CS 4395

Knowledge Base Appendix Sample:

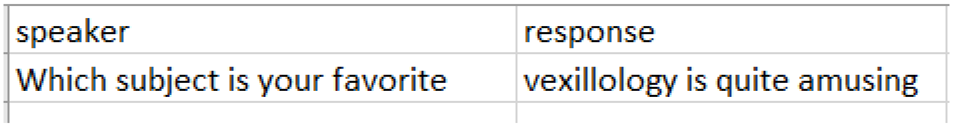
These are some examples of dialogue from the dataset knowledge base. These are all lines from the show. The left side is the speaker where the character says something to Sheldon. The response right side is the response that Sheldon gives in response the text that the other character says to Sheldon.



1	speaker	response
14	Sheldor?	The Conqueror.
15	What are you doing?	AFK. Im playing Age of Conan, an online multiplayer game set in the universe of Robert E. Howards Conan the Barbarian.
16	Oh.	Sheldor, back online.
17	Whats AFK?	AFK. Away from keyboard.
18	OIC.	What does that stand for?
19	Oh, I see?	Yes, but what does it stand for?

Sample User Model Database:

This is example of user given dialogue to the chatbot to fix the bot for a specific user. When they give suggestions for how to improve the bot the program makes a new csv to hold their recommended changes. Then the next time the model is trained for that user the model uses the original dataset along with the new suggested changes to the model.



speaker	response
Which subject is your favorite	vexillology is quite amusing

Arjun Bala
Chatbot Project
CS 4395

Bibliography:

"The Big Bang Theory Transcripts - Forever Dreaming." *Index - Forever Dreaming*, https://transcripts.foreverdreaming.org/viewforum.php?f=159.

Team, Keras. "Keras Documentation: Character-Level Recurrent Sequence-to-Sequence Model." *Keras*, https://keras.io/examples/nlp/lstm_seq2seq/.