Chatbot Project

Context

Chatbots streamline interactions between people and services they need. This enhances the consumer's experience while allowing a company to engage with customers. The interactions between customers and chatbots can improve operational efficiency (by reducing customer service agent load) if the chatbot can correctly identify the entities and intents of the customer.

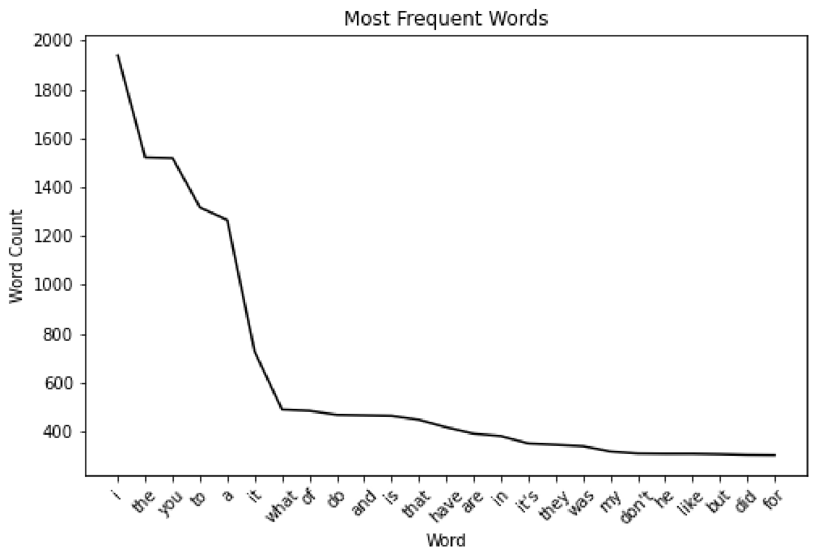
In this project, the goal was to create a chatbot that could appropriately identify the intent and entities during a generalized conversation. To understand the process involved with this project, I began by attempting to implement a chatbot entirely my own without the help of any chat-related python packages. Once I gained some understanding, I created a bot using Rasa.

Data Wrangling

Data for this project was a compilation of my personal chat history, supplemented with some chat content from kaggle. The data consists of a user input sentence and a bot response sentence. The data is not classified by intent, so the chatbot must figure out the intent and respond appropriately.

Exploratory Data Analysis

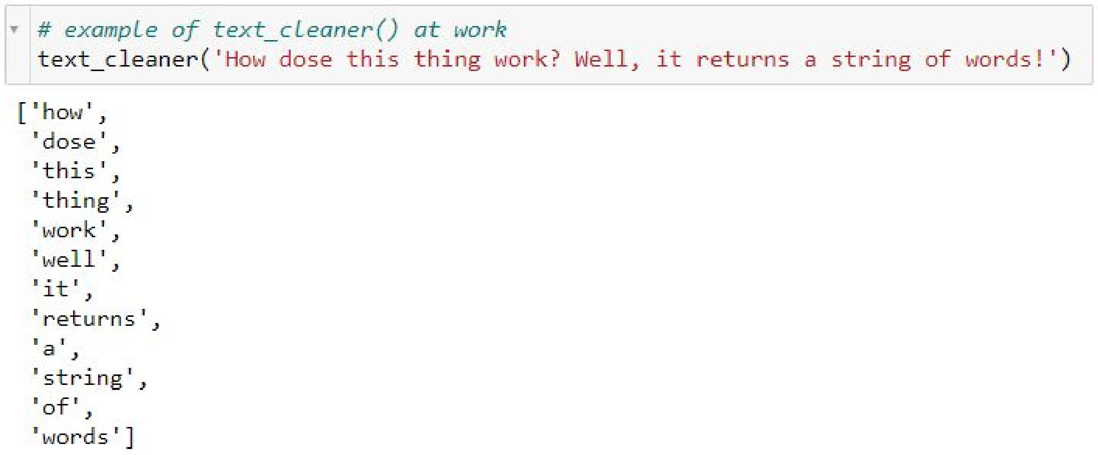
While exploring the data, the most frequent words in the dataset were explored. It became clear when looking at the most frequent words that pronouns and transitional words appeared more often than others. This is an important thing to note as these words will be less helpful in identifying the intent of the sentence.



It was also noted in EDA that the list of most frequent words was not necessarily correlated to the most frequent words within a sentence. In the case of words like “I” and “a”, these were high frequency words in both the dataset and within sentences, which is not unexpected since someone might use them a multitude of times within a sentence. Similarly, it isn’t unexpected that many words do not occur at all in a sentence sample. It also makes sense that most words, including common words like “what” or “and”, only occurred once or twice within a sentence.

Modeling Conclusions and Rasa

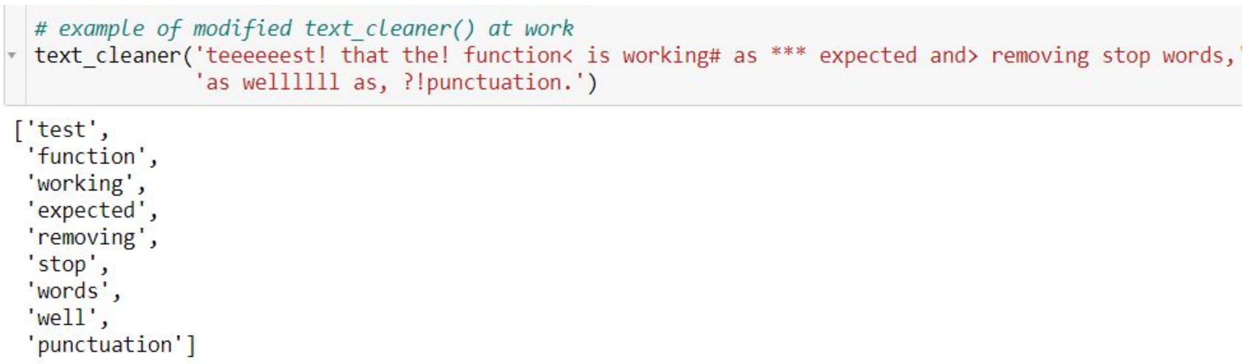
To create a model for the chatbot, a decision tree classifier was created in order to reduce the data into increasingly small groups until a response could be provided. To effectively implement Count Vectorization on each sentence, a novel function, text_cleaner, was created to split sentences into individual words, remove punctuation, and make all words lowercase.



```
# example of text_cleaner() at work
text_cleaner('How dose this thing work? Well, it returns a string of words!')
```

```
['how',
 'dose',
 'this',
 'thing',
 'work',
 'well',
 'it',
 'returns',
 'a',
 'string',
 'of',
 'words']
```

Above is an example of how the text_cleaner implemented in the initial pipeline worked. Note, that while this pipeline performed as expected, it did not handle things like misspellings, removal of other non-alphabetic characters, or remove common words (meaning that words like “I” which occur frequently even within sentences could be over-weighted). To account for these, a modified function was created that performed the sentence split, removed punctuation and other symbols, converted all words to lowercase, and corrected misspellings. The result is a more comprehensive text_cleaner function to use within Count Vectorizing that returns words more pertinent to the intent of the sentence.



```
# example of modified text_cleaner() at work
text_cleaner('teeeeeest! that the! function< is working# as *** expected and> removing stop words,'
             'as wellllll as, ?!punctuation.')
```

```
['test',
 'function',
 'working',
 'expected',
 'removing',
 'stop',
 'words',
 'well',
 'punctuation']
```

Interestingly, when using this function in the decision tree classifier pipeline, the results got worse. The initial results showed that the bot performed very well on lines it was trained on, so-so on lines similar to those it was trained on, and pretty poorly on lines not like anything it has seen, although these responses were pretty funny (one such example was when asked, “where is the lizard?” one bot responded “yes, old people don’t smell like fruit”)! However, when the new text_clean function was implemented the results were much worse. Even on lines that the bot had seen previously it responded incorrectly. For example, when asked “what did they say” the bot responded “but i’m worried about tiger” which seemingly has nothing to do with the initial question.

A few factors contributed to the issues seen with bot performance, most notably, the small dataset did not provide enough context for the bot. Moreover, the unlabeled data resulted in more the bot had to understand with that small dataset. With the more aggressive text_cleaner function, the bot had even less to work from since all the stop words were removed. Clearly, the bot was depending on these even though they did not provide information on intent. Finally, the bot had no feedback to improve the model when it performed incorrectly.

At this point, I felt I understood the non-trivial requirements for a well functioning bot and moved to the Rasa platform in order to help address the issues at hand. Using Rasa, I was able to build in intents and entities for the bot to understand. For example, the sentence “Hi, how are you today” could be grouped as a “greeting_and_convo” entity and appropriate responses for this group could be created based on previous conversations. Meanwhile, text like “Hi” could be classified as a “greeting” and things like “where is the lizard?” could be classified as “inquiry” as the intent and “lizard” as the entity. Once text was appropriately classified, responses could be easily set and used by the bot for good results each time. Moreover, the entities could be placed in memory and used in a response. And, when the bot ran into a situation where it did not understand the intent and/or entity, a generalized response like “let’s chat about that later” could be used.

Finally, and perhaps most importantly for making an effective model, the Rasa x platform allowed me to correct when the bot classified an intent or entity incorrectly. Then the model could be retrained to incorporate this new information.