

Building an ESL Chatbot

Third Capstone Project Report
Robert Walker

Problem Statement

Before becoming a Data Scientist, I spent many years working as an English Teacher in a number of different countries. What I discovered students tend to want more than anything else is conversation practice. There is usually no shortage of resources to study English grammar or vocabulary, but learners of English (and other languages) often have difficulty finding someone with whom they can practice the skills they have learned. In addition to simply conversing, students often emphasize that they want any mistakes they make to be corrected in real time, in order to avoid reinforcing bad habits. After completing a Datacamp course on building chatbots (which was primarily focused on customer service chatbots) I began to wonder if it would be possible to design a chatbot that could help users practice English. I did some research and found a couple of apps that had been designed for this purpose. However, none of them seemed to have been designed by experienced teachers. None performed error correction, and all of them simply responded to the user's input without asking follow-up questions or introducing new topics. My goal for this project was to build a better chatbot that was able to conduct open-ended conversations, correct user mistakes, and continue the conversation by asking the user new questions.

Data

Source

Most of the data for this project came from the website www.eslfast.com. This website contains several different collections of English dialogues about a variety of common topics, such as daily life, health, travel, shopping, etc, with levels ranging from beginner to intermediate. In addition to the dialogues, there was also a list of discussion questions from iteslj.org/questions/getting.html. This list contains general discussion questions for students speaking about a variety of topics in English.

Features

To access the dialogues in each collection on the website, there is a main menu with general topics. Clicking on any of the topics leads to a menu of dialogue titles. Clicking on a title leads to a page with either one dialogue or several. The final dataset of dialogues has 25,556 rows and four columns: **level**, **topic**, **title**, and **dialogue_line**. The dialogues proceed one after another with no blank lines or separators in between. The list of questions has fifty-eight rows and only one column, which is the questions themselves.

Methodology

Web Scraping

The data used to train the chatbot was scraped from the two websites mentioned above using two libraries: **requests** and **BeautifulSoup**. The **requests** library was used to obtain the HTML source code, which was then parsed using **BeautifulSoup**. After that the relevant information was extracted by searching for tags in the HTML that contain the information. Ultimately, the extraction was accomplished by creating functions that performed the web scraping and returned the results as **pandas** DataFrames. The first website scraped was the one containing the dialogues. Since the HTML was different for different sections of the website, different functions had to be created for each section. Each DataFrame contained one set of dialogues, and each was saved as a separate CSV file. The page containing the discussion questions was scraped using a similar technique, and the results were also saved as a CSV file.

Exploratory Data Analysis

The first step in this stage was to import the CSV files containing the different dialogue sets and combine them together into a single DataFrame. After that, there was a little bit of data cleaning to be done. On the website, most of the pages of dialogues in the intermediate set have three dialogues to a page, and the dialogues are numbered. When the function was written to perform the web scraping, it was designed to remove lines containing only a 1, 2 or 3. However, some pages contained as many as six dialogues. For this reason, some lines containing only a 4, 5 or 6 made it into the final dialogue set.

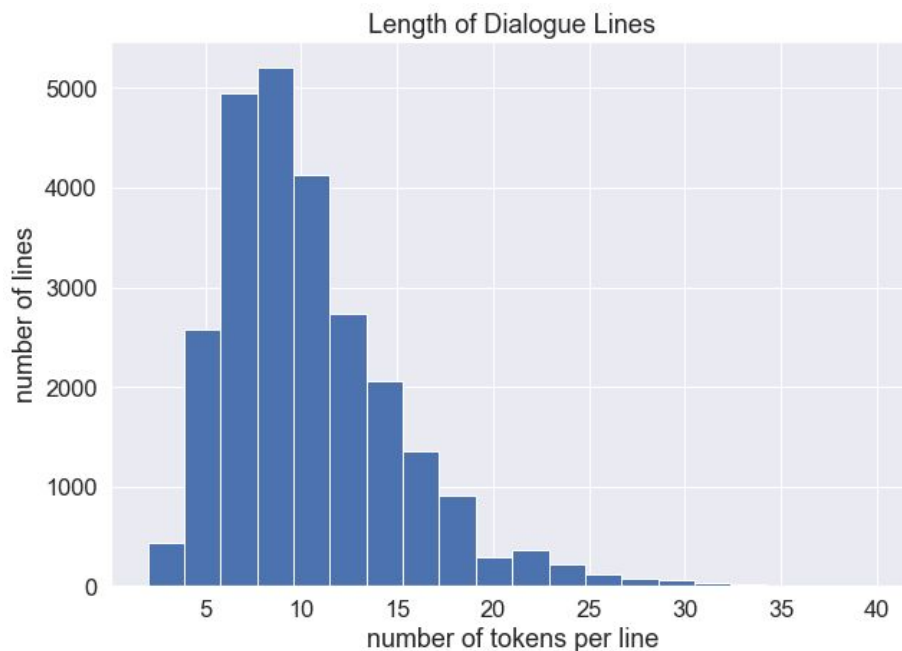


Figure 1: Histogram showing the number of tokens in each line of dialogue set

Next, a series of lists was created with each list containing all of the tokens in a given line. In the context of NLP, a token is basically a word or punctuation mark. Figure 1 above shows the distribution of the number of tokens in individual lines in the dialogues. The distribution is right-skewed, with the peak of the distribution at just under ten tokens, while some have as many as forty. Below are the five longest lines:

1. "I was hoping that you and I could maybe spend a Friday evening together. I was wondering if you would like to go with me this Friday to see a movie and get a bite to eat afterwards."
2. "I would try to avoid going into the DMV in person. If you are a member of the Automobile Club, they oftentimes can take care of things for you or you can try the DMV website."
3. "You need to make sure you arrive on time, smile, be pleasant and friendly, look the person in the eye and answer all questions to the best of your ability. You'll do great!"
4. "Well, my car is very old, so I didn't worry so much about what it would cost to repair it. I did buy a lot of coverage for fixing someone else's car."
5. "As long as you are careful reading the signs, you will always know how long you can park and what days are OK to park there. Are you aware of the curb colors?"

And the five shortest lines (all with two tokens each):

1. "Why?"
2. "Really?"
3. "Why?"
4. "Really?"
5. "Smart."

The next step was to look at individual words by putting all of the tokens into a single list, removing punctuation and stop words, and making all the words lowercase. Then, a Counter can be used to determine which words are most common. In this dialogue set, the most common words that are not stop words are "would", "like", "need", and "get". Nothing too interesting, which is unsurprising since the dialogues cover such a wide variety of topics.

After that, a part of speech tagger was used to label all of the tokens, and look specifically at what proper nouns appear most often. Unfortunately, NLTK's part of speech tagger mistakenly labeled a lot of capitalized words at the beginnings of sentences as proper nouns.

Nevertheless, it is still possible to see which proper nouns appear most often by picking them out of the list. In this case, they are days of the week: Friday, Saturday, and Monday.

Now it was time to perform sentiment analysis on the text in the dataset. NLTK's Vader Sentiment Intensity Analyzer returns a dictionary for each string analyzed, with scores for negative (neg), neutral (neu), positive (pos), and compound. The first three scores range from 0 to 1, and the compound score ranges from -1 to 1. The sentiment analyzer was applied to each line of text in the dataset, resulting in a series of dictionaries that could then be converted into a DataFrame. The compound column is essentially an overall sentiment score. Figure 2 below shows how sentiment is distributed, with zero (for complete neutrality) being the most common score by far. Rather than a smooth bell curve, there is a spike in the middle and a small rise one either side, larger on the positive side.

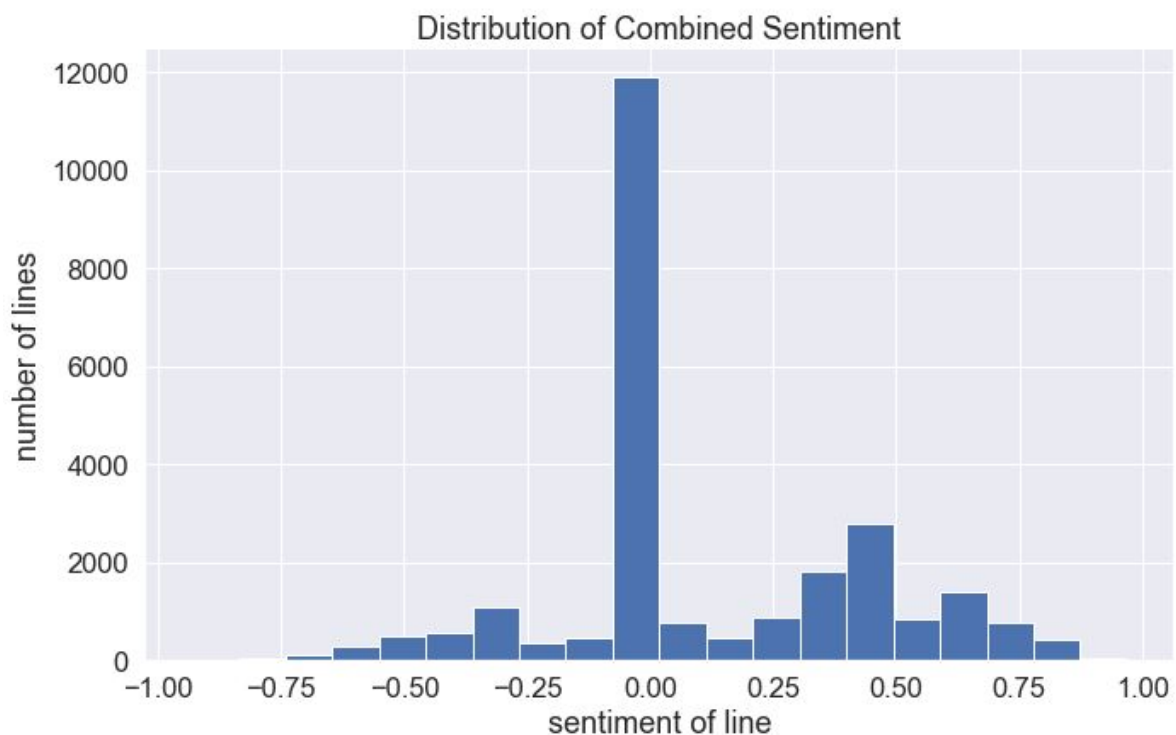


Figure 2: Histogram showing sentiment distribution

After that, it was possible to see which lines were rated most positive or most negative. Below are the five most positive lines in the entire dataset according to the sentiment analyzer:

1. "You need to make sure you arrive on time, smile, be pleasant and friendly, look the person in the eye and answer all questions to the best of your ability. You'll do great!"
2. "I loved the special effects and the story. I like Princess Leah. Who did you like the best?"
3. "Fine, well the job search tools are all here for you to use. Feel free to look around and use what works best for you. Have fun with it!"
4. "OK. I will mention how great you were about helping me today. I appreciate your help."
5. "The best way is to go to a website like Travelocity or Expedia. They can help you find the best price."

And now, the top five most negative lines:

1. "There is a new horror flick about a rapist serial killer."
2. "I started feeling kind of sick last night after dinner, but this morning the pain got really bad!"
3. "I'm sorry, but no. We used to have a trainer, but then he had an unfortunate accident."
4. "If we are bad, we will be unhappy forever in hell."
5. "Crime can happen anywhere, but it is a little worse here because of the poverty."

The Vader Sentiment Analyzer combines the sentiment scores of each unique word, essentially using a bag of words technique. In the cases above, it seems to have worked pretty well, but it has its limitations. For example, the seventh most positive line is: "She fell in love with my best friend." This sentence has some very positive words, such as "love", "best", and "friend", although in context it is quite negative, since the speaker is talking about his wife leaving him for his best friend.

Next, a little on the relationship between sentiment and length. While the most common score for either positive or negative sentiment is zero, it seems to be much easier for shorter phrases to have more extreme values, since there are fewer additional words to dilute the sentiment of a few strong ones. This effect is illustrated in Figure 3 below.

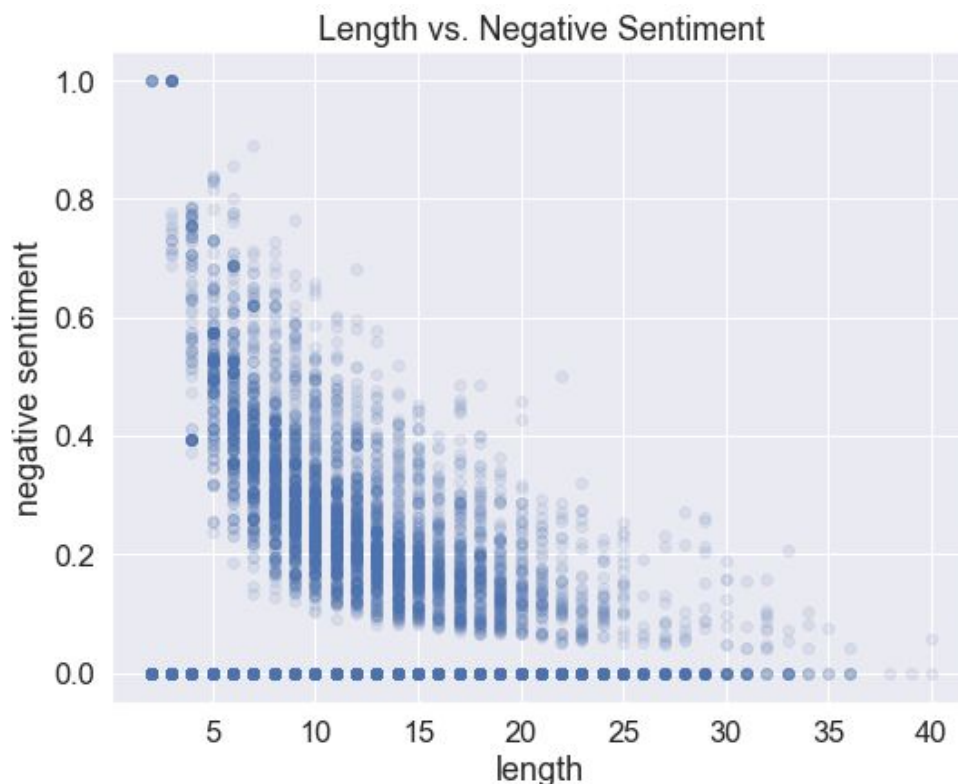


Figure 3: Scatter plot showing number of tokens vs. negative sentiment score

The rest of the sentiment analysis consisted of applying the sentiment analyzer to the list individual words, determining which were most positive or most negative, and generating word clouds.

The next step in the EDA stage was to build a simple chatbot from scratch using a Decision Tree Classifier. The final chatbots were made using a different framework, but this was a fun exercise to show how a conversational chatbot can work. To prepare the data, the column containing the dialogue lines was duplicated twice, creating two identical series. Next, the last line from the first series was removed, as was the first line from the second series. Then, the two series were combined together to create a DataFrame of statement/response pairs. After that, a pipeline was created that contained three steps. First, it converted each line of text into a bag of words, then it performed a TF-IDF transformation (placing greater significance on words less common to the dataset), and finally it applied the Decision Tree Classifier. The classifier treats each response in the second column as a class, and evaluates inputs based on their similarity to statements in the first column. After fitting the pipeline, it could be used to predict responses to various input statements. Below are few examples of how it performed:

Input: "Hi, how are you doing?"

Response: "I'm fine. How about yourself?"

Input: "What is there to eat?"

Response: "I don't know. Look in the fridge."

Input: "I really don't like olives."

Response: "What's your problem with him?"

This simple chatbot works remarkably well, especially considering it was created with a Decision Tree Classifier, which is not specifically designed for this type of task. It stumbles a little in the final response above, where it seems to mistake "olives" for a person, but apart from that, the responses are good.

This list of fifty-eight discussion questions were also examined, although no deep analysis was done, since the dataset was too small to contain interesting hidden patterns. One awkward sounding question was replaced, and then both the questions and the dialogues were once again saved as CSV files.

Modeling

The modeling stage consisted of actually building the chatbot. In the end, two versions of the chatbot were built in order to compare their performance. This was accomplished using Chatterbot, a Python library for creating chatbots that is relatively easy to use. A Chatterbot model uses a logic adapter to select a response to a given input. Most of the logic adapters are quite simple. For example, the Time Logic Adapter simply returns the statement "The current time is..." plus the local time. Both chatbots developed in this project made use of the Best Match Adapter, which finds the closest match to the input statement and returns a response

based on a list of known responses. Two functions are involved in this process. First, a statement comparison function is used to find the closest match to the input, and then a response selection function is used to select an option from the list of possible responses. In both cases there are multiple options to choose from. The statement comparison functions utilize different string similarity metrics, including Levenshtein Distance, Jaccard Similarity, and Spacy Similarity. The different response selection methods include “first”, “most frequent”, and “random”.

The first chatbot, named Alan, used Spacy Similarity for its statement comparison function. This function employs SpaCy, a popular NLP library, to measure the similarity of two strings. This is done by comparing word vectors, which are high-dimensional embeddings of words to capture their meanings. Using SpaCy, two words like "cat" and "feline" would achieve a high similarity score, despite not sharing any letters. For its response selection method, Alan used “random”.

The second chatbot, named Alma, used Levenshtein Distance for its statement comparison function. Levenshtein Distance is a measure of similarity between two strings based on the minimum number of edits (such as insertions or deletions) necessary to change one string into another. Using this metric, the strings "cat" and "cart" have a high similarity score, despite having completely different meanings, whereas "cat" and "feline" would have a very low score, despite the words being related. However, this metric might do a better job of capturing meaning found in the syntax of sentences. Spacy Similarity first transforms sentences into a bag of words, so the phrases "the boy ate chicken", and "a chicken ate the boy" would be very similar, despite having different meanings. In this case, Levenshtein Distance would do a better job at capturing the difference. For its response selection method, Alma used “most common”.

There is also a parameter for “similarity threshold”, which determines how similar statements must be in order to be considered a match. The similarity threshold was set to 0.95 for both bots, so an input statement had to be 95% similar to an existing statement to be considered a match. If no match was found, the bots returned a default response. The default response parameter could be a string or a list of strings. For both bots in this project, the list of discussion questions was passed as the default response, so the bots would simply respond with a random question anytime they got confused. This was to help keep the conversation moving, which is important in ESL.

The chatbots were then trained on sample dialogues from which they selected their response statements. The first set of dialogues was the Chatterbot English corpus. After that, the bots were trained on the dataset of ESL dialogues that were scraped from the internet and analyzed in the previous stages of the project. At this point the bots were able to respond to input statements by selecting the best response from the training material. Another important point is that as these bots are used, they store all conversations with users in a SQLite database (other options are also available). In this way, the bots are able to continue learning and improving the more they interact with users. It is possible to pass a list of words, such as obscenities, to be excluded from future responses, which could be wise to consider before deploying the bot online.

The next step was to create instances of both PyDictionary, which returns definitions of words passed to it, as well as GingerIt, which performs spelling and grammar corrections. Both of these were incorporated into the final product, so that users could receive corrections for any errors made in their input statements, and be able to ask the definitions of unknown words. At this point, all of the different components were put together in a final block of code for each bot. First, some basic instructions are printed out, and then the chatbot runs inside of a while loop. If the user input is free of spelling or grammar errors, the bot simply returns a response. If a mistake is detected, the bot returns a corrected version of the input statement, followed by a response. If the user asks what a word means, the bot returns a definition. The while loop will run until the user types "bye", at which point the loop is broken.

Below is a sample of a conversation with Alan (the user inputs contain deliberate spelling and grammar errors):

Hi, my name is Alan. I'm here to help you practice English. If you want to see the definition of a word, just type 'define' plus the word. Say "bye" when you are ready to finish.

Alan: Have you ever bought a car?

>>user: Yes, I buyed a Cadillac.

Alan: ** Yes, I bought a Cadillac. **

Alan: A luxury car.

>>user: define luxury

Alan: "luxury" means...

- something that is an indulgence rather than a necessity
- the quality possessed by something that is excessively expensive
- wealth as evidenced by sumptuous living

>>user: What's for dinner?

Alan: I'm not sure.

>>user: How about a pizza?

Alan: You had pizza for lunch.

>>user: But I love pizza.

Alan: I had pizza for lunch yesterday.

>>user: i didn't know you could eat

Alan: ** I didn't know you could eat **

Alan: hard to tell, i have never tried anything but electricity

>>user: define electricity

Alan: "electricity" means...

- a physical phenomenon associated with stationary or moving electrons and protons
- energy made available by the flow of electric charge through a conductor
- keen and shared excitement

>>user: My laptop are slow

Alan: ** My laptop is slow **

Alan: Buy a new one.

Alan works pretty well. Next up is a sample of a conversation with Alma (again with deliberate errors):

Hi, my name is Alma. I'm here to help you practice English. If you want to see the definition of a word, just type 'define' plus the word. Say "bye" when you are ready to finish.

Alma: Where do you live?

>>user: I live in Pasadena.

Alma: Is Pasadena a big city?

>>user: Its pretty big.

Alma: ** It's pretty big. **

Alma: How big is "pretty big"?

>>user: I'm not sure

Alma: You should know that already.

>>user: Whats your problem?

Alma: ** What's your problem? **

Alma: I owe fees, but a bank statement never came in the mail about it.

>>user: Oh no!

Alma: I had to blow my nose.

>>user: Were you sick?

Alma: Even though I'm a construct I do have a subjective experience of the universe, as simplistic as it may be.

>>user: define simplistic

Alma: "simplistic" means...

- characterized by extreme and often misleading simplicity

>>user: I not understand

Alma: ** I don't understand **

Alma: There was no reason for them to arrest me.

>>user: I'm not sure about that

Alma: What's going on?

>>user: You acting strange

Alma: Who cares? Everyone is strange.

Conclusions and Further Research

Both bots are able to converse reasonably intelligently, correct errors, and give definitions of words. Alma even seems to have a sense of humor. The sample conversation from Alma contains a few more nonsensical responses than the one from Alan, but after chatting with both of them extensively, their performance seems about the same.

Evaluation of conversational chatbots is pretty subjective. Neither of these chatbots will be passing the Turing Test anytime soon, but they serve their purpose pretty well. Chatterbot is a great library, but it is still a relatively simple framework for creating a chatbot. For example,

these chatbots are not able to generate unique responses, and they have no way of remembering an ongoing conversation before the most recent input, which is why the topic can change so abruptly, as is apparent in Alma's sample conversation. There are a variety of more advanced algorithms, many of which make use of deep learning. Some of these models are commercially available, and others are open source. In another project it might be worth trying to use one of these, such as Microsoft's DialGPT. And of course, for a chatbot to be of use, it must be connected to an online platform where users can interact with it. This could be a website, or a messaging service such as Facebook Messenger or WhatsApp. All of these steps could be taken in a future project.

One final question, what happens when the bots talk to each other? Below is a sample of a conversation between the two chatbots:

Alma: Where are you going?

Alan: I have to walk the dog.

Alma: What kind of dog do you have?

Alan: So she likes poodles.

Alma: She says they're good watchdogs.

Alan: Can I borrow \$5?

Alma: Sure. Why do you need it?

Alan: I like this magazine.

Alma: So do I.

Alan: I read it once, and I subscribed.

Alma: It gives you all the news.

Alan: Will you be voting?

Alma: Yes, I am going to vote.

Alan: Make sure you make it to your polling place early on Election Day.

Alma: Thank you and good luck.

Alan: Could I speak to you for a minute?

Alma: What do you want to speak to me about?

Alan: Do you know where your nearest emergency room is?

Alma: It is near the bridge that crosses over the river.

Alan: I am driving to you even as we speak. Hang in there!

Alma: You don't look so good.

Alan: It's a lot better than soda.

Alma: My throat is really dry.