Semantic-Similarity-Chatbot

Let us first I know what is semantic similarity – It is defined as Semantic -Similarity or Semantic Textual Similarity, is a task in the area of NLP that scores the relationship between texts or documents using a defined metric. Semantic Similarity has various applications, such as information retrieval, text summarization, sentiment analysis, etc.

Process involving building this chatbot is as follows:

Defining a Conversation

Taking a turn

Word Vectors - A word vector is an attempt to mathematically represent the meaning of a word.

Sentence vectors - A sentence vector is an attempt to mathematically represent the meaning of a sentence in dimensional spaces.

Writing a code - With your understanding of these concepts, we can actually start writing some code. For our semantic similarity chatbot, we need:

* Pre-trained word vectors
* A corpus of conversations
* Some code to parse conversations into turns and map each turn to its response
* Some code that can average the word vectors in some text to produce a sentence vector
* A database that will allow us to store sentence vectors and look them up by similarity
* Some code to take an incoming conversational turn, turn it into a sentence vector, and then look up the most similar vector in the database.

Step1: Installing NLP library spacy which helps us in text parsing, text classification and tokenize text.

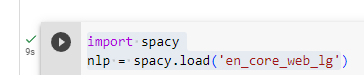
! pip install spacy



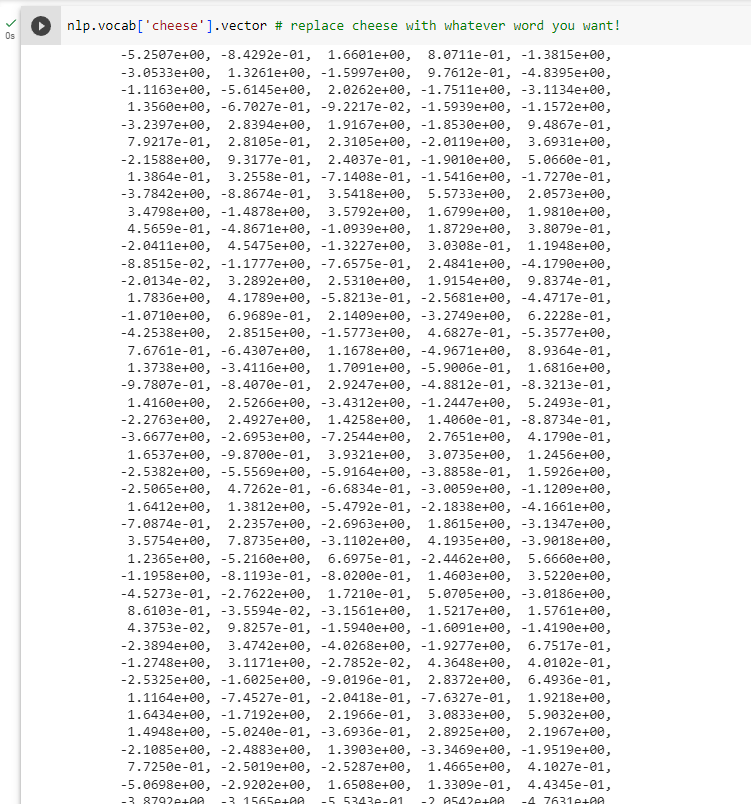
Requirement Satisfied

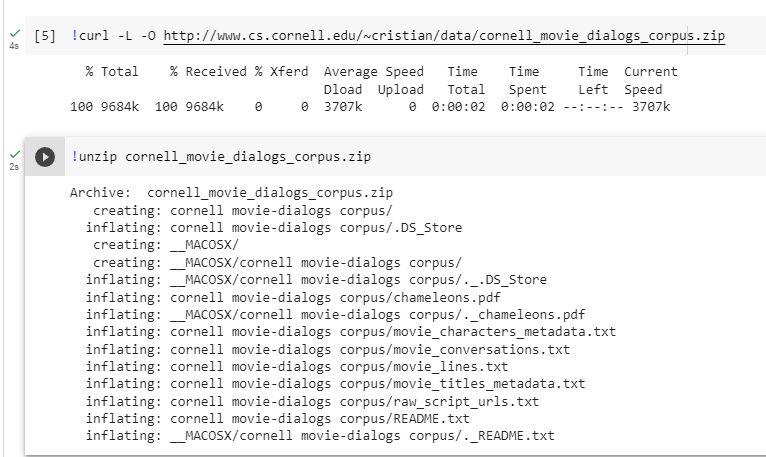
Step2: It turns out that spaCy requires a "model" file, which is a bundle of statistical information that allows the library to parse text into words and parts of speech. Also importing to code to load spacy





Step3: Vectorization of words to store in the form of array and on mathematical format .to get one dimensional data and to provide accurate output.



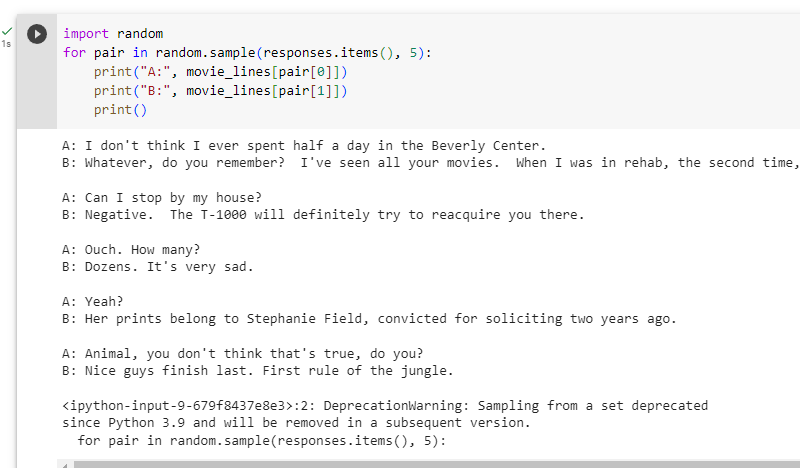
Step4: Downloading Conversational Data from an external website and unzipping it for our use.

Step5: Error checking

Step6: In this code is reading the file "movie\_lines.txt" from the "cornell movie-dialogs corpus" directory and creating a dictionary called "movie\_lines". Each line of the file is split into 5 parts using the delimiter "+++$$$+++" and the first part is used as the key and the fifth part is used as the value in the dictionary. If a line does not have 5 parts, an empty string is used as the value



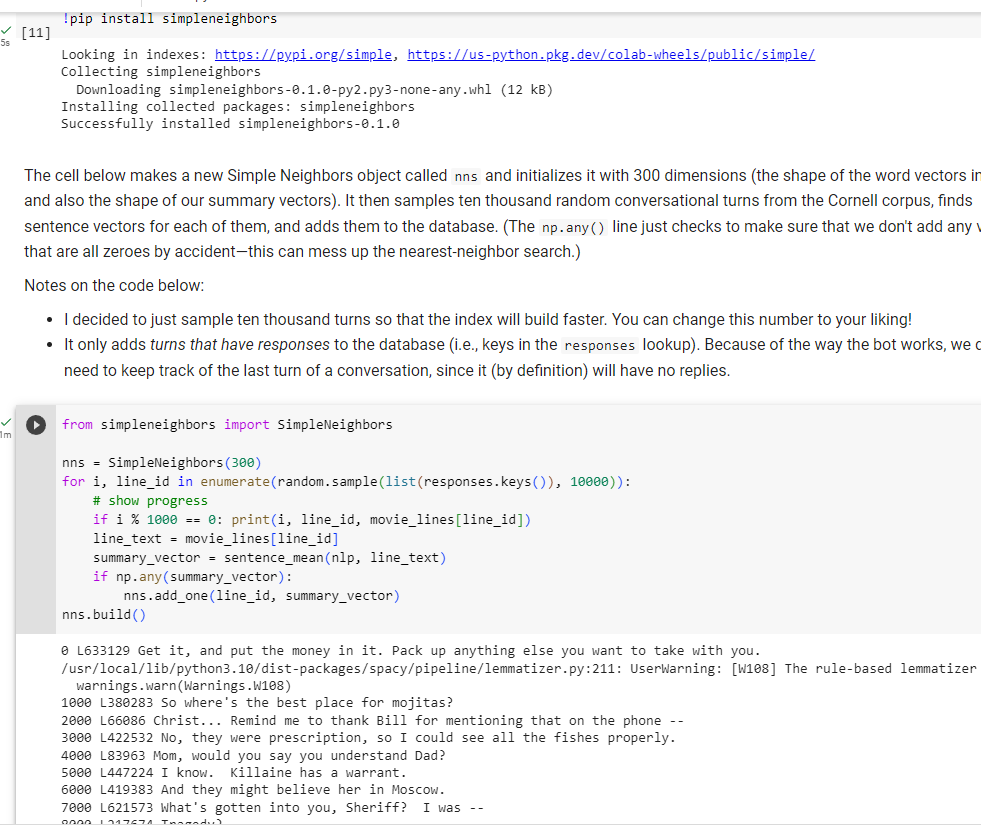
Step7: Just to make sure everything works, the cell below prints out five random pairs of conversational turns from the corpus:



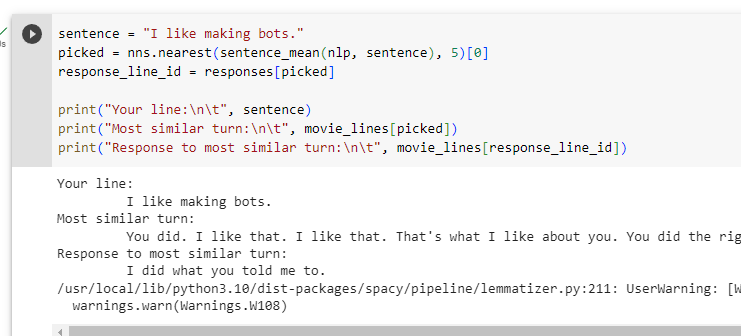
Step8: Making a sentence Vector, we defining it in function called sentence\_mean, first we load nlp to parse and tokenize then find the mean using numpy mean.



Step9: we use K Nearest Neighbour, is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure. The below code is creating a SimpleNeighbors object with 300 dimensions and then adding 10000 random movie lines to it. For each line, it calculates the mean vector of its words using the `sentence\_mean` function and adds the line ID and its summary vector to the SimpleNeighbors object. Finally, it builds the index for the SimpleNeighbors object. This code is likely part of a larger project that involves using the SimpleNeighbors object to perform nearest neighbor searches on the movie lines.



Step 10: This code Identifies response candidate with context.

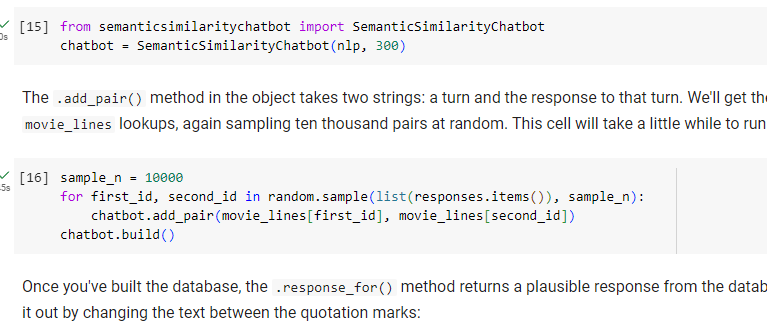


Step11: Putting it all Together

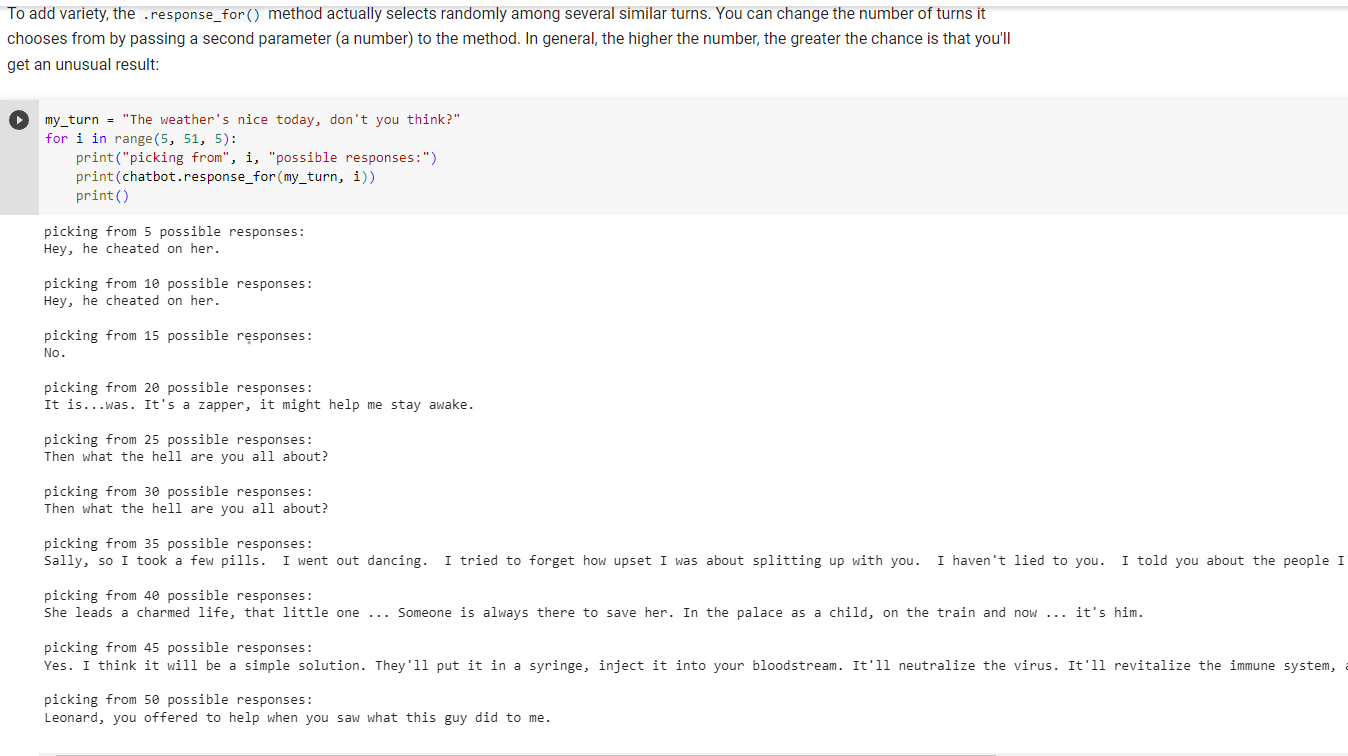
! pip install https://github.com/aparrish/semanticsimilaritychatbot/archive/master.zip

Requirement Satisfied.

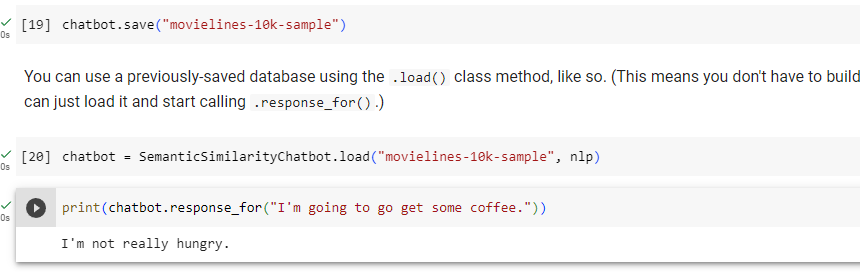
Then create a chatbot object, passing in the spaCy language object (nlp) and the number of dimensions:



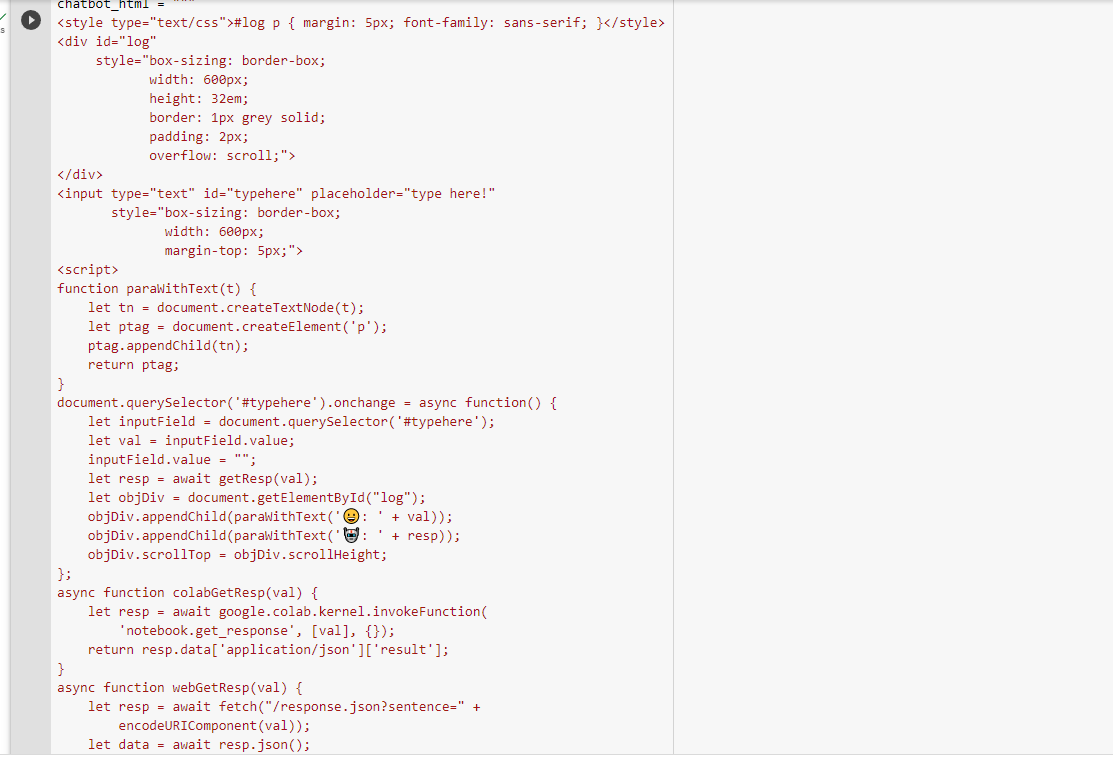
Step12: Generating response to the statement

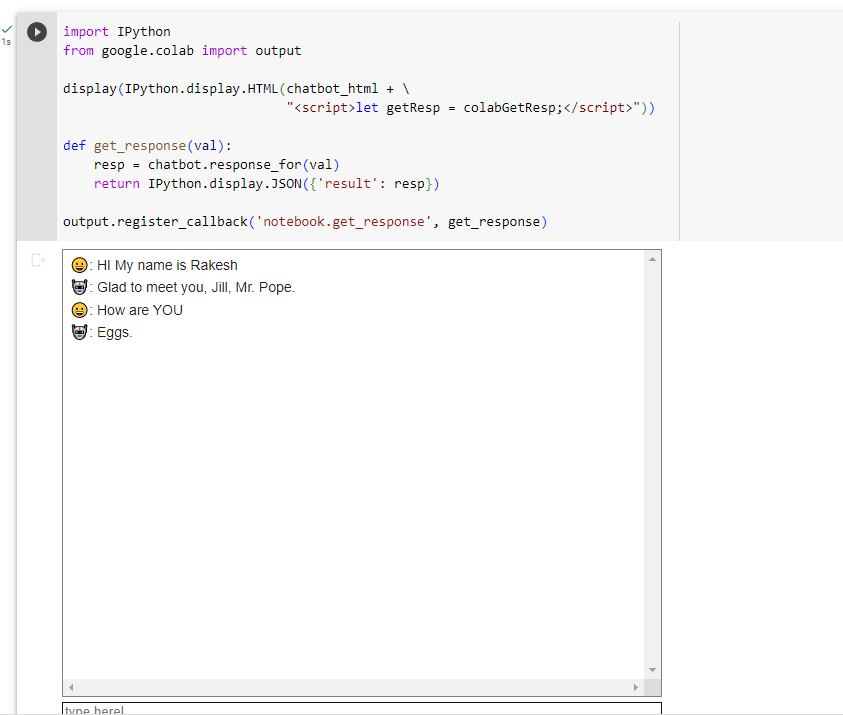


Step13: Saving it to database and printing response from the database



Step14: Making It interactive running chatbot on collab





Drawbacks it runs on user data stored in database and does not have the capability to think and just produce out based on collected dataset.