*# coding: utf-8*

*# # Meet Robo: your friend*

​

**import** nltk

**import** warnings

warnings.filterwarnings("ignore")

​

*# nltk.download() # for downloading packages*

**import** numpy **as** np

**import** random

**import** string *# to process standard python strings*

*# For our example,we will be using the Wikipedia page for chatbots as our corpus.*

*# Copy the contents from the page and place it in a text file named ‘chatbot.txt’.*

*# However, you can use any corpus of your choice.*

*# We will read in the corpus.txt file*

f**=**open('chatbot.txt','r',errors **=** 'ignore')

raw**=**f.read()

raw**=**raw.lower()*# converts to lowercase*

*#nltk.download('punkt') # first-time use only*

*#nltk.download('wordnet') # first-time use only*

*# convert the entire corpus into a list of sentences and a list of words for further pre-processing*

sent\_tokens **=** nltk.sent\_tokenize(raw)*# converts to list of sentences*

word\_tokens **=** nltk.word\_tokenize(raw)*# converts to list of words*

​

sent\_tokens[:2]

word\_tokens[:5]

sent\_tokens[0]

word\_tokens[:5]

*#WordNet is a semantically-oriented dictionary of English included in NLTK.*

lemmer **=** nltk.stem.WordNetLemmatizer()

​

*# LemTokens will take as input the tokens and return normalized tokens.*

**def** LemTokens(tokens):

**return** [lemmer.lemmatize(token) **for** token **in** tokens]

​

​

remove\_punct\_dict **=** dict((ord(punct), **None**) **for** punct **in** string.punctuation)

**def** LemNormalize(text):

**return** LemTokens(nltk.word\_tokenize(text.lower().translate(remove\_punct\_dict)))

GREETING\_INPUTS **=** ("hello", "hi", "greetings", "sup", "what's up","hey",)

GREETING\_RESPONSES **=** ["hi", "hey", "\*nods\*", "hi there", "hello", "I am glad! You are talking to me"]

​

*# Checking for greetings*

*# define a function for a greeting by the bot i.e if a user’s input is a greeting,*

*# the bot shall return a greeting response.*

**def** greeting(sentence):

"""If user's input is a greeting, return a greeting response"""

**for** word **in** sentence.split():

**if** word.lower() **in** GREETING\_INPUTS:

**return** random.choice(GREETING\_RESPONSES)

*# the words need to be encoded as integers or floating point values*

*# for use as input to a machine learning algorithm, called feature extraction (or vectorization).*

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

​

*# find the similarity between words entered by the user and the words in the corpus.*

*# This is the simplest possible implementation of a chatbot.*

**from** sklearn.metrics.pairwise **import** cosine\_similarity

​

​

*# Generating response*

*# define a function response which searches the user’s utterance for one or more known keywords*

*# and returns one of several possible responses. If it doesn’t find the input matching any of the keywords,*

*# it returns a response:” I am sorry! I don’t understand you”*

**def** response(user\_response):

robo\_response**=**''

sent\_tokens.append(user\_response)

*# TF-IDF are word frequency scores that try to highlight words that are more interesting,*

*# e.g. frequent in a document but not across documents.*

*# The TfidfVectorizer will tokenize documents, learn the vocabulary and*

*# inverse document frequency weightings, and allow you to encode new documents.*

TfidfVec **=** TfidfVectorizer(tokenizer**=**LemNormalize, stop\_words**=**'english')

​

*# Learn vocabulary and idf, return term-document matrix*

*# Returns X : Tf-idf-weighted sparse matrix, [n\_samples, n\_features]*

tfidf **=** TfidfVec.fit\_transform(sent\_tokens)

*# print (tfidf.shape)*

*# Cosine similarity is a measure of similarity between two non-zero vectors.*

*# Using this formula we can find out the similarity between any two documents d1 and d2.*

*# Cosine Similarity (d1, d2) = Dot product(d1, d2) / ||d1|| \* ||d2||*

vals **=** cosine\_similarity(tfidf[**-**1], tfidf)

*# function is used to perform an indirect sort along the given axis using the algorithm*

*# specified by the kind keyword. It returns an array of indices of the same shape as arr*

*# that would sort the array.*

idx**=**vals.argsort()[0][**-**2]

*# Returns a new array that is a one-dimensional flattening of this array (recursively).*

*# That is, for every element that is an array, extract its elements into the new array.*

*# If the optional level argument determines the level of recursion to flatten.*

flat **=** vals.flatten()

flat.sort()

req\_tfidf **=** flat[**-**2]

​

**if**(req\_tfidf**==**0):

robo\_response**=**robo\_response**+**"I am sorry! I don't understand you"

**return** robo\_response

**else**:

robo\_response **=** robo\_response**+**sent\_tokens[idx]

**return** robo\_response

​

​

flag**=True**

print("ROBO: My name is Chatty. I will answer your queries about Chatbots. If you want to exit, type Bye!")

​

**while**(flag**==True**):

user\_response **=** input()

user\_response**=**user\_response.lower()

**if**(user\_response**!=**'bye'):

**if**(user\_response**==**'thanks' **or** user\_response**==**'thank you' ):

flag**=False**

print("ROBO: You are welcome..")

**else**:

**if**(greeting(user\_response)**!=None**):

print("ROBO: "**+**greeting(user\_response))

**else**:

print("ROBO: ",end**=**"")

print(response(user\_response))

sent\_tokens.remove(user\_response)

**else**:

flag**=False**

print("ROBO: Bye! take care..")

​