05_Simple_Chatbot

February 19, 2022

1 Building a Simple Chatbot using NLTK

1.1 Import necessary libraries

[1]:
```
import io
import random
import string # to process standard python strings
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer # convert a␣
↪collection of raw documents to a matrix of TF-IDF features
from sklearn.metrics.pairwise import cosine_similarity
```

1.2 Downloading and installing NLTK

[2]:
```
#!pip install nltk
```

1.2.1 Installing NLTK Packages

[3]:
```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('popular', quiet=True) # for downloading packages
nltk.download('punkt') # first-time use only
nltk.download('wordnet') # first-time use only
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[3]:
```
True
```

1.3 Reading in the corpus

For our example, we will be using the text from a file as our corpus. However, you can use any corpus of your choice.

[4]:
```
f=open('input.txt','r',errors = 'ignore')
raw=f.read()
raw = raw.lower()# converts to lowercase
```

The main issue with text data is that it is all in text format (strings). However, the Machine learning algorithms need some sort of numerical feature vector in order to perform the task. So before we start with any NLP project we need to pre-process it to make it ideal for working. Basic text pre-processing includes:

- Converting the entire text into **uppercase** or **lowercase**, so that the algorithm does not treat the same words in different cases as different
- **Tokenization**: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e words that we actually want.
  - Sentence tokenizer can be used to find the list of sentences and
  - Word tokenizer can be used to find the list of words in strings.

1.4 Tokenisation

[5]:
```
sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words
```

1.5 Preprocessing

We shall now define a function called LemTokens which will take as input the tokens and return normalized tokens.

[6]:
```
lemmer = nltk.stem.WordNetLemmatizer()
#WordNet is a semantically-oriented dictionary of English included in NLTK.
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)))
```

1.6 Keyword matching

Next, we shall 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.

[7]:
```
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",\
                   "how are you?")
```

```
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", \
                      "I am glad! You are talking to me", \
                      "I am fine! How about you?"]
def greeting(sentence):
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

1.7 Generating Response

1.7.1 Bag of Words

After the initial preprocessing phase, we need to transform text into a meaningful vector (or array) of numbers. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Why is it is called a "bag" of words? That is because any information about the order or structure of words in the document is discarded and the model is only **concerned with whether the known words occur in the document, not where they occur in the document.**

The intuition behind the Bag of Words is that documents are similar if they have similar content. Also, we can learn something about the meaning of the document from its content alone.

For example, if our dictionary contains the words {Learning, is, the, not, great}, and we want to vectorize the text "Learning is great", we would have the following vector: (1, 1, 0, 0, 1).

1.7.2 TF-IDF Approach

A problem with the Bag of Words approach is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much "informational content". Also, it will give more weight to longer documents than shorter documents.

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like "the" that are also frequent across all documents are penalized. This approach to scoring is called Term Frequency-Inverse Document Frequency, or TF-IDF for short, where:

Term Frequency: is a scoring of the frequency of the word in the current document.

`TF = (Number of times term t appears in a document)/(Number of terms in the document)`

Inverse Document Frequency: is a scoring of how rare the word is across documents.

`IDF = 1+log(N/n), where, N is the number of documents and n is the number of documents a term t`

1.7.3 Cosine Similarity

Tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus

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

where d1,d2 are two non zero vectors.

To generate a response from our bot for input questions, the concept of document similarity will be used. We 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"

[8]:
```
def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    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
```

Finally, we will feed the lines that we want our bot to say while starting and ending a conversation depending upon user's input.

[9]:
```
flag=True
print("SABot: My name is SABot. How can I assist you?. \
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("SABot: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("SABot: "+greeting(user_response))
            else:
                print("SABot: ",end=" ")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("SABot: Bye! take care..")
```

```
SABot: My name is SABot. How can I assist you?. If you want to exit, type Bye!
hi
SABot: *nods*
what is computer?
SABot:  computer
a computer is a programmable device that stores, retrieves, and processes data
and gives output.
what is data science?
SABot:  data science
data science is the interdesciplinary field.
what is AI?
SABot:  artificial intelligence (ai) is the ability of a computer or a robot
controlled by a computer to do tasks that are usually done by humans because
they require human intelligence and discernment.
bye
SABot: Bye! take care..
```

[ ]: