**AI CHATBOT :**

**Index:**

**INTRODUCTION**

**ABSTRACT**

**Each student module**

**MODULE(11805316-k18uw-12)**

**1. Generate response**

**2. Downloading and installing NLTK Pre-requisites**

**3. Pre-requisites**

**MODULE (11804882-k18uw-44)**

**4. Nlp**

**5 Text Pre- Processing with NLTK**

**6. Bag of Words**

**MODULE(11805297-k18uw-16)**

**7. Importing the necessary libraries**

**8.corpus**

**9. Reading the data**

**MODULE (11805225-k18uw-05)**

**10.pre processing the raw – text**

**11. keyword matching**

**12.output of the code**

**INTRODUCTION:**

Chatbots are not a recent development. They are simulations which can understand human language, process it and interact back with humans while performing specific tasks. For example, a chatbot can be employed as a helpdesk executive. The first chatbot was created by Joseph Weinbaum in 1966, named Eliza. It all started when Alan Turing published an article named “Computer Machinery and Intelligence”, and raised an intriguing question, “Can machine think?”, and ever since, we have seen multiple chatbots surpassing their predecessors to be more naturally conversant and technologically advanced. These advancements have led us to an era where conversations with chatbots have become as normal and natural as with another human.

ABSTRACT:

The technology at the core of the rise of the chatbot is natural language processing (“NLP”). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing a great deal of additional research which should lead to continued improvement in the effectiveness of chatbots in the years to come.

A simple chatbot can be created by loading an FAQ (frequently asked questions) into chatbot software. The functionality of the chatbot can be improved by integrating it into the organization’s enterprise software, allowing more personal questions to be answered, like“What is my balance?”, or “What is the status of my order?”.

MODULE(11805316-k18uw-Ro.12)

Generate response:

To generate a response from our bot for input questions, the concept of document similarity will be used. So we begin by importing the necessary modules.

* From scikit learn library, import the [TFidf vectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) to convert a collection of raw documents to a matrix of TF-IDF features.

from sklearn.feature\_extraction.text import TfidfVectorizer

* Also, import[cosine similarity](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html) module from scikit learn library

from sklearn.metrics.pairwise import cosine\_similarity

This will be used to find the similarity between words entered by the user and the words in the corpus. This is the simplest possible implementation of a chatbot.

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”

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 the user’s input.

flag=True  
print("ROBO: My name is Robo. 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..")

**Downloading and installing NLTK**

1. Install NLTK: run pip install nltk
2. Test installation: run python then type import nltk

**Installing NLTK Packages:**

import NLTK and run nltk.download().This will open the NLTK downloader from where you can choose the corpora and models to download. You can also download all packages at once

**Pre-requisites:**

Hands-On knowledge of **scikit** library and **NLTK** is assumed.

**MODULE (11804882-k18uw-44)**

**Nlp**:

The field of study that focuses on the interactions between human language and computers is called Natural Language Processing, or NLP for short. It sits at the intersection of computer science, artificial intelligence, and computational linguistics[Wikipedia].NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

**Downloading and installing NLTK**

1. Install NLTK: run pip install nltk
2. Test installation: run python then type import nltk

**Installing NLTK Packages:**

import NLTK and run nltk.download().This will open the NLTK downloader from where you can choose the corpora and models to download. You can also download all packages at once.

**Text Pre- Processing with NLTK:**

The main issue with text data is that it is all in text format (strings). However, 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 work. 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.
* Removing **Noise** i.e everything that isn’t in a standard number or letter.
* Removing **Stop words.**Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words
* **Stemming**: Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form — generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stemtization”, the result would be a single word “stem”.
* **Lemmatization**: A slight variant of stemming is lemmatization. The major difference between these is, that, stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of Lemmatization are that “run” is a base form for words like “running” or “ran” or that the word “better” and “good” are in the same lemma so they are considered the same.

**Bag of Words:**

After the initial preprocessing phase, we need to transform the 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.

**MODULE (11805297-k18uw-16)**

# Importing the necessary libraries

import nltk  
import numpy as np  
import random  
import string # to process standard python strings

# Corpus

For our example, we will be using the Wikipedia page for [chatbots](https://en.wikipedia.org/wiki/Chatbot) 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.

# Reading in the data

We will read in the corpus.txt file and convert the entire corpus into a list of sentences and a list of words for further pre-processing.

f=open('chatbot.txt','r',errors = 'ignore')raw=f.read()raw=raw.lower()# converts to lowercasenltk.download('punkt') # first-time use only  
nltk.download('wordnet') # first-time use onlysent\_tokens = nltk.sent\_tokenize(raw)# converts to list of sentences   
word\_tokens = nltk.word\_tokenize(raw)# converts to list of words

# MODULE(11805225 -k18uw-Ro.5)

# Pre-processing the raw text

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

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)))

# 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.ELIZA uses a simple keyword matching for greetings. We will utilize the same concept here.

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"]def greeting(sentence):  
   
 for word in sentence.split():  
 if word.lower() in GREETING\_INPUTS:  
 return random.choice(GREETING\_RESPONSES)

Output of the code