

**APPENDIX 1**

**CHATBOT**

**END TERM REPORT**

*by*

**AMRITPAL, SUBHANKAR, TARIKUL, DEEPAK KUMAR**

(Section: K18JF)

(Roll Number(s): 05,17,25,27)



**Department of Intelligent Systems  
School of Computer Science Engineering  
Lovely Professional University, Jalandhar  
APRIL-2020**

## **APPENDIX 2**

### **Student Declaration**

This is to declare that this report has been written by me/us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be copied, I/we are shall take full responsibility for it.

Name: AMRITPAL  
Reg No: 11808067  
Roll number:05

Name: SUBHANKAR  
Reg No. 11808092  
Roll number:17

Name: TARIKUL  
Reg No: 11808017  
Roll number:25

Name: DEEPAK KUMAR SONI  
Reg No.  
Roll number:27

Place: LOVELY PROFESSIONAL UNIVERSITY  
Date: 08-APRIL-2020

## APPENDIX 3

### TABLE OF CONTENTS

TITLE	PAGE NO.
1. INTRODUCTION.....	05
2. BACKGROUND.....	06
3. APPLICATION OF CHATBOT.....	07-09
4. STEPS IN CREATING CHATBOT.....	09-10
5. DESCRIPTION OF PROJECT.....	11- 19
5.1 TRAINNING THE DATA.....	11
5.2 LIBRARIES USED.....	12-13
5.3 LOADING THE DATASET.....	14
5.4 EXTRACTING THE DATASET.....	14
5.5 WORD STEMMING .....	15
5.6 BAG OF WORDS.....	15
5.7 DEVELOPING THE MODEL.....	16
5.8 TRAINNING AND SAVING THE MODEL.....	16
5.9 MAKING PREDICTIONS .....	16
5.10DULL CODE FOR THE PROJECT.....	17-20
6. CONTRIBUTION FROM MEMBERS.....	20

## APPENDIX 4

### BONAFIDE CERTIFICATE

Certified that this project report “ CUSTOMER SERVICE CHATBOT ” is the bonafide work of “AMRITPAL ,SUBHANKAR KHANDAI,TARIKUL ISLAM,DEEPAK KUMAR SONI” who carried out the project work under my supervision.

Signature of the Supervisor

Name of supervisor

Academic Designation

ID of Supervisor

Department of Supervisor

# INTRODUCTION

A **chatbot** is a software application used to conduct an on-line chat conversation via text or text-to-speech, in lieu of providing direct contact with a live human agent. Designed to convincingly simulate the way a human would behave as a conversational partner, chatbot systems typically require continuous tuning and testing, and many in production remain unable to adequately converse or pass the industry standard Turing test. The term "ChatterBot" was originally coined by Michael Mauldin (creator of the first Verbot) in 1994 to describe these conversational programs.

Chatbots are typically used in dialog systems for various purposes including customer service, request routing, or for information gathering. While some chatbot applications use extensive word-classification processes, Natural Language processors, and sophisticated [AI](#), others simply scan for general keywords and generate responses using common phrases obtained from an associated library or database.

Today, most chatbots are accessed on-line via website popups, or through virtual assistants such as Google Assistant, Amazon Alexa, or messaging apps such as Facebook Messenger or WeChat. Chatbots are typically classified into usage categories that include: commerce (e-commerce via chat), education, entertainment, finance, health, news, and productivity.

# BACKGROUND

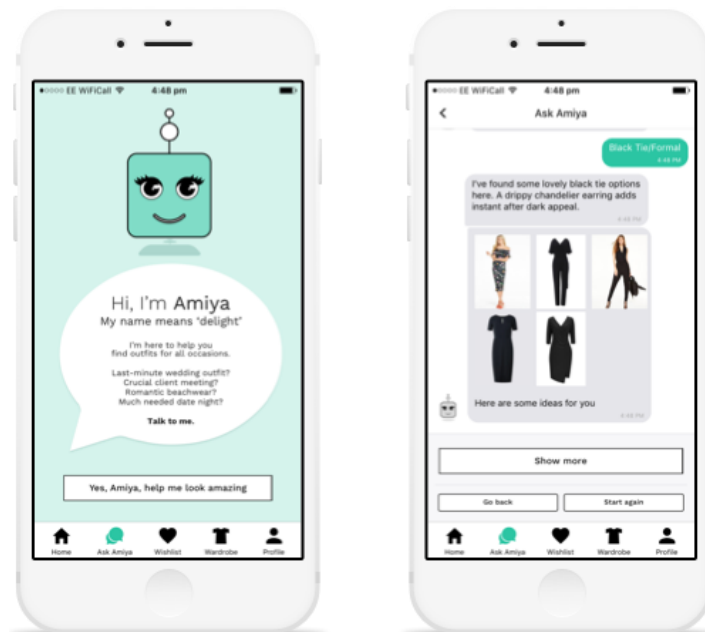
In 1950, Alan Turing's famous article "Computing Machinery and Intelligence" was published, which proposed what is now called the Turing test as a criterion of intelligence. This criterion depends on the ability of a computer program to impersonate a human in a real-time written conversation with a human judge to the extent that the judge is unable to distinguish reliably—on the basis of the conversational content alone—between the program and a real human. The notoriety of Turing's proposed test stimulated great interest in Joseph Weizenbaum's program ELIZA, published in 1966, which seemed to be able to fool users into believing that they were conversing with a real human.

Interface designers have come to appreciate that humans' readiness to interpret computer output as genuinely conversational—even when it is actually based on rather simple pattern-matching—can be exploited for useful purposes. Most people prefer to engage with programs that are human-like, and this gives chatbot-style techniques a potentially useful role in interactive systems that need to elicit information from users, as long as that information is relatively straightforward and falls into predictable categories. Thus, for example, online help systems can usefully employ chatbot techniques to identify the area of help that users require, potentially providing a "friendlier" interface than a more formal search or menu system. This sort of usage holds the prospect of moving chatbot technology from Weizenbaum's "shelf ... reserved for curios" to that marked "genuinely useful computational methods".

# APPLICATION OF CHATBOT

## 1. FASHION ASSISTANT

Chatbots in customer service can act as a personal fashion stylist. Price comparison allows customers to browse through different products. By chatting with a company's customer service chatbot in real time, the buying process can become simpler for online shoppers. Chatbots also recommend products and this makes the entire customer experience more enjoyable and stress-free.

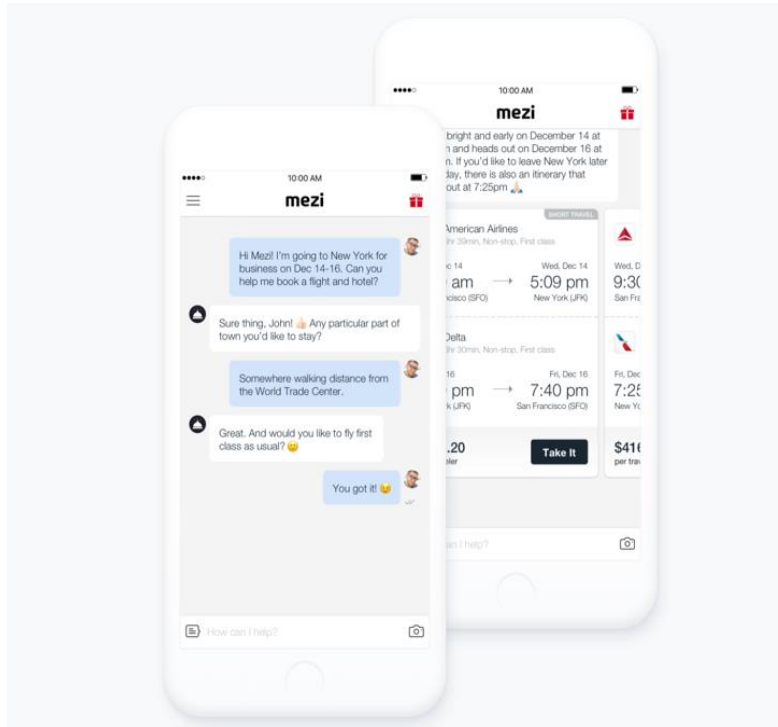


## 2. SMART TRIP ASSISTANT

For the travel industry, such customer service chatbots could prove to be a boon. Besides the benefit of booking and scheduling flights, chatbots can help integrate additional services via social media platforms. Passenger experience can be free of stress if chatbots are employed during the journey.

With open APIs, vertical platforms can be created, linking additional services like Airbnb, Uber, and Lyft. In this way, travel companies can expand their footprints and gain a wider customer base by simply deploying chatbots across different checkpoints.

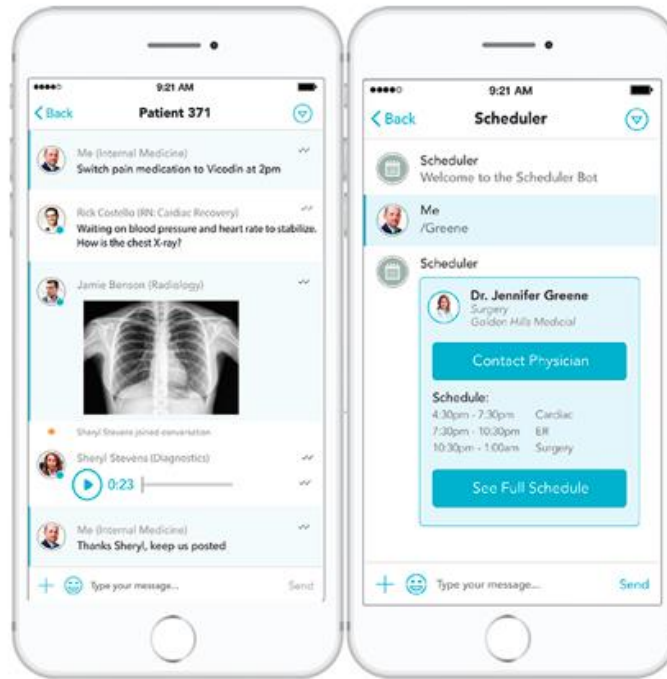
For travelers, this would mean that they do not need to worry about small issues like where to change money, where to hail a cab, wait time in the security check line at airports. The only requirement for this automated travel would be a messaging app downloaded on a smartphone.



### 3. HEALTHCARE ASSISTANT

One of the most useful applications of a chatbot is in the healthcare department. An average patient spends 30 minutes trying to get to the right service in a local hospital. But deploying conversational chatbots in the healthcare sector can significantly reduce long waits and free up times for, patients, nurses, and doctors. From registration to coverage and claims to compliance, chatbots are in popular demand in healthcare services.





# STEPS IN CREATING A CHATBOT

*1 Data pre-processing*

*2 Building the sequence to sequence model*

*3 Training the sequence to sequence model*

*4. Testing the sequence to sequence model*

*5. Improving and tuning the sequence to sequence model*

**1. Data pre-processing:** - Data Pre-processing for Machine learning in Python. Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Pre-processing is a technique that is used to convert the raw data into a clean data set

**2. Building the sequence to sequence model:** - Sequence To Sequence model introduced in Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation has since then, become the Go-To model for Dialogue Systems and Machine Translation. It consists of two RNNs (Recurrent Neural Network): An Encoder and a Decoder. The encoder takes a sequence(sentence) as input and processes one symbol(word) at each timestep. Its objective is to convert a sequence of symbols into a fixed size feature vector that encodes only the important information in the sequence while losing the unnecessary information. You can visualize data flow in the encoder along the time axis, as the flow of local information from one end of the sequence to another. Each hidden state influences the next hidden state and the final hidden state can be seen as the summary of the sequence. This state is called the context or thought vector, as it represents the intention of the sequence. From the context, the decoder generates another sequence, one symbol(word) at a time. Here, at each time step, the decoder is influenced by the context and the previously generated symbols.

**3. Training the sequence to sequence model:** - As we need this sequence to sequence model learns and then uses its intelligence to solve our problems which will translate our language to our desired one. So, we need to train our chatbot to learn and help us. In practice, training for a single sentence is done by “forcing” the decoder to generate gold sequences, and penalizing it for assigning the sequence a low probability. Losses for each token in the sequence are summed. Then, the summed loss is used to take a step in the right direction in all model parameters

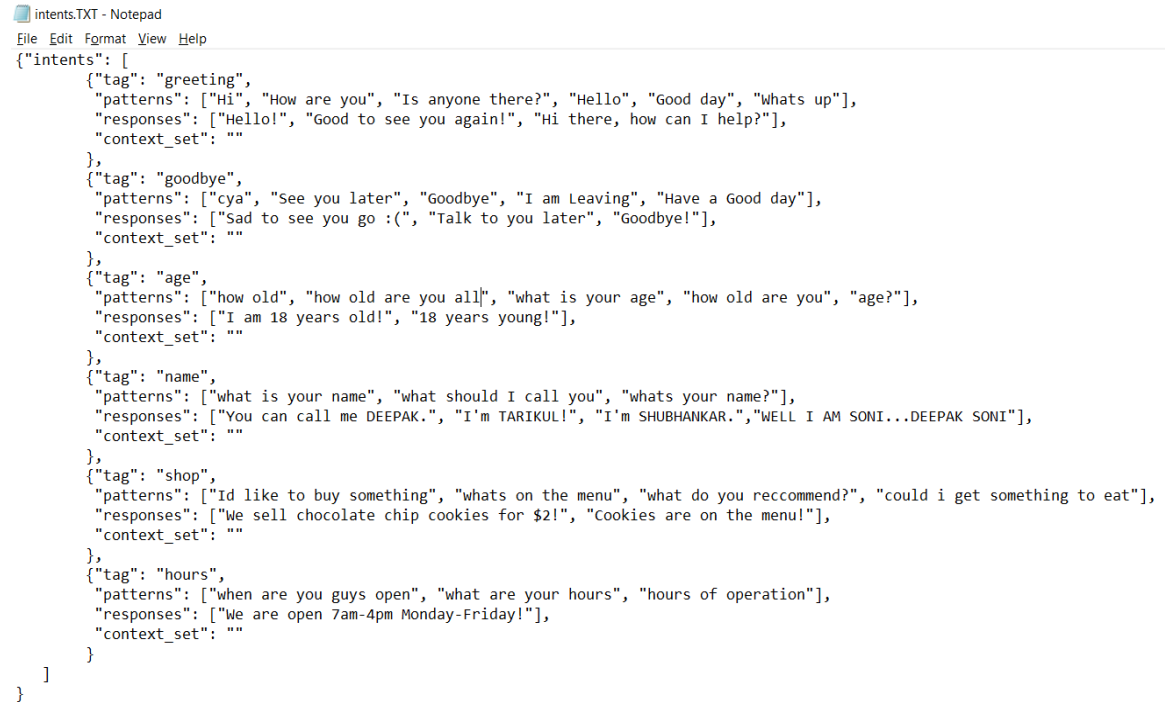
**4. Testing the sequence to sequence model:** - Here we will test our model. We will see our model work as we expected or not. If the model work as it is then our model is successful model.

**5. Improving and tuning the sequence to sequence model:** - *Here we will* improve our model as our expectation and turn our model what our need is. Then our model will be the model which we want as expected.

# DESCRIPTION OF THE PROJECT

## 1. TRAINING THE DATA

We will just use data that we write ourselves. To follow along with the tutorial properly you will need to create a .JSON file that contains the same format as the one seen below. I've called my file "intents.json".



```
intents.TXT - Notepad
File Edit Format View Help
{"intents": [
  {
    "tag": "greeting",
    "patterns": ["Hi", "How are you", "Is anyone there?", "Hello", "Good day", "Whats up"],
    "responses": ["Hello!", "Good to see you again!", "Hi there, how can I help?"],
    "context_set": ""
  },
  {
    "tag": "goodbye",
    "patterns": ["cya", "See you later", "Goodbye", "I am Leaving", "Have a Good day"],
    "responses": ["Sad to see you go :((", "Talk to you later", "Goodbye!"],
    "context_set": ""
  },
  {
    "tag": "age",
    "patterns": ["how old", "how old are you all", "what is your age", "how old are you", "age?"],
    "responses": ["I am 18 years old!", "18 years young!"],
    "context_set": ""
  },
  {
    "tag": "name",
    "patterns": ["what is your name", "what should I call you", "whats your name?"],
    "responses": ["You can call me DEEPAK.", "I'm TARIKUL!", "I'm SHUBHANKAR.", "WELL I AM SONI...DEEPAK SONI"],
    "context_set": ""
  },
  {
    "tag": "shop",
    "patterns": ["Id like to buy something", "whats on the menu", "what do you reccommend?", "could i get something to eat"],
    "responses": ["We sell chocolate chip cookies for $2!", "Cookies are on the menu!"],
    "context_set": ""
  },
  {
    "tag": "hours",
    "patterns": ["when are you guys open", "what are your hours", "hours of operation"],
    "responses": ["We are open 7am-4pm Monday-Friday!"],
    "context_set": ""
  }
]
}
```

## 2. LIBRARIES USED

Simply go to CMD and type: **pip install "package name"**. Where you will replace "package\_name" with all of the entries listed above.

- **NLTK:**

NLTK stands for Natural Language Toolkit. This toolkit is one of the most powerful NLP libraries which contains packages to make machines understand human language and reply to it with an appropriate response. Tokenization, Stemming, Lemmatization, Punctuation, Character count, word count are some of these packages which will be discussed in this tutorial.

Code used is

```
import nltk  
from nltk.stem.lancaster import LancasterStemmer  
stemmer = LancasterStemmer()
```

- **NUMPY:**

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical [functions](#) to operate on these arrays.

Code used is

```
import numpy as np
```

- **TFLEARN:**

TFlearn is a modular and transparent deep learning library built on top of Tensorflow. ... Full transparency over Tensorflow. All functions are built over tensors and can be used independently of TFlearn. Powerful helper functions to train any TensorFlow graph, with support of multiple inputs, outputs and optimizers.

Code used is

```
import tflearn
```

- **Tensorflow:**

TensorFlow is a rich system for managing all aspects of a machine learning system; however, this class focuses on using a particular TensorFlow API to develop and train machine learning models. TensorFlow APIs are arranged hierarchically, with the high-level APIs built on the low-level APIs. Machine learning researchers use the low-level APIs to create and explore new machine learning algorithms.

Code used is

*import tensorflow as tf*

- **Random:**

In Python, a random module implements pseudo-random number generators for various distributions including integer, float (real). This function of the module is used in predicting the output

Code used is

*import random*

- **JSON:**

The json library can parse JSON from strings or files. The library parses JSON into a Python dictionary or list. It can also convert Python dictionaries or lists into JSON strings.

Code used is

*import json*

- **PICKLE:**

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream.

Code used is

*Import pickle*

### 3. LOADING THE DATASET

```
with open("intents.json") as file:
    data=json.load(file)
```

### 4. EXTRACTING THE DATASET

We need all of the patterns and which class/tag they belong to. We also want a list of all of the unique words in our patterns. For each pattern we will turn it into a list of words using *nltk.word\_tokenizer*, rather than having them as strings. We will then add each pattern into our docs\_x list and its associated tag into the docs\_y list.

```
5
6
7     words=[]
8     labels=[]
9     docs_x=[]
10    docs_y=[]
11    for intent in data["intents"]:
12        for pattern in intent["patterns"]:
13            wrds=nltk.word_tokenize(pattern)
14            words.extend(wrds)
15            docs_x.append(wrds)
16            docs_y.append(intent["tag"])
17
18        if intent["tag"] not in labels:
19            labels.append(intent["tag"])
20
21    words=[stemmer.stem(w.lower()) for w in words if w in "?"]
22    words=sorted(list(set(words)))
23    labels = sorted(labels)
24
```

### 5. WORD STEMMING

Stemming a word is attempting to find the root of the word. For example, the word "thats" stem might be "that" and the word "happening" would have the stem of "happen". We will use this process of stemming words to reduce the vocabulary of our model and attempt to find the more general meaning behind sentences.

```

words=[stemmer.stem(w.lower()) for w in words if w in "?"]
words=sorted(list(set(words)))
labels = sorted(labels)

```

## 6. BAG OF WORDS

neural networks and machine learning algorithms require numerical input. So our list of strings won't cut it. We need some way to represent our sentences with numbers and this is where a bag of words comes in. What we are going to do is represent each sentence with a list the length of the amount of words in our model's vocabulary. Each position in the list will represent a word from our vocabulary. If the position in the list is a 1 then that will mean that the word exists in our sentence, if it is a 0 then the word is not present. We call this a bag of words because the order in which the words appear in the sentence is lost, we only know the presence of words in our model's vocabulary.

As well as formatting our input we need to format our output to make sense to the neural network. Similarly to a bag of words we will create output lists which are the length of the amount of labels/tags we have in our dataset. Each position in the list will represent one distinct label/tag, a 1 in any of those positions will show which label/tag is represented.

```

training=[]
output=[]
out_empty=[0 for _ in range(len(labels))]
for x, doc in enumerate(docs_x):
    bag=[]
    wrds=[stemmer.stem(w) for w in doc]

    for w in words:
        if w in wrds:
            bag.append(1)
        else:
            bag.append(0)

    output_row=out_empty[:]
    output_row[labels.index(docs_y[x])]=1

    training.append(bag)
    output.append(output_row)

training = np.array(training)
output=np.array(output)

```

## 7. DEVELOPING A MODEL

we have preprocessed all of our data we are ready to start creating and training a model. For our purposes we will use a fairly standard feed-forward neural network with two hidden layers. The goal of our

network will be to look at a bag of words and give a class that they belong too (one of our tags from the JSON file).

```
1 tf.reset_default_graph()
2
3 net=tflearn.input_data(shape=[None,len(training[0])])
4 net=tflearn.fully_connected(net,8)
5 net=tflearn.fully_connected(net,8)
6 net=tflearn.fully_connected(net,len(output[0]),activation="softmax")
7 net=tflearn.regression(net)
8
9 model=tflearn.DNN(net)
```

## 8. TRAINING AND SAVING THE MODEL

We have setup our model its time to train it on our data! To do these we will **fit** our data to the model. The number of epochs we set is the amount of times that the model will see the same information while training.

```
model.fit(training,output,n_epoch=1000,batch_size=8,show_metric=True)
model.save("model.tflearn")
```

## 9. MAKING PREDICTIONS

Ideally we want to generate a response to any sentence the user types in. To do this we need to remember that our model does not take string input, it takes a bag of words. We also need to realize that our model does not spit out sentences, it generates a list of probabilities for all of our classes. This makes the process to generate a response look like the following:

- Get some input from the user
- Convert it to a bag of words
- Get a prediction from the model
- Find the most probable class
- Pick a response from that class



```

def bag_of_words(s, words):
    bag=[0 for _ in range(len(words))]
    s_words=nlk.word.tokenize(s)
    s_words=[stemmer.stem(word.lower()) for word in s_words]

    for se in s_words:
        for i,w in enumerate(words):
            if w== se:
                bag[i]=1

    return np.array(bag)

def chat():
    print("start talking with the bot(type quit to stop)")
    while True:
        inp=input("You:")
        if inp.lower()=="quit":
            break
        results=model.predict([bag_of_words(inp, words)])[0]
        results_index=np.argmax(results)
        tag=labels[results_index]

        if results[results_index]>0.7:
            for tg in data["intents"]:
                if tg['tag']==tag:
                    responses=tg['responses']

            print(random.choice(responses))
        else:
            print("Sorry I didnt get that")

```

## FULL CODE OF THE PROJECT

```

# -*- coding: utf-8 -*-
"""

```

Created on Thu Mar 19 19:06:03 2020

```

@author: Shubhankar
"""

```

```

import nltk
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()
import numpy as np
import tflearn
import tensorflow as tf

```

```

import random
import json
import pickle

with open("intents.json") as file:
    data=json.load(file)
try:
    with open("data.pikle","rb") as f:
        words,labels,training,output=pickle.load(f)

except:

    words=[]
    labels=[]
    docs_x=[]
    docs_y=[]
    for intent in data["intents"]:
        for pattern in intent["patterns"]:
            wrds=nltk.word_tokenize(pattern)
            words.extend(wrds)
            docs_x.append(wrds)
            docs_y.append(intent["tag"])

        if intent["tag"] not in labels:
            labels.append(intent["tag"])

    words=[stemmer.stem(w.lower()) for w in words if w in "?"]
    words=sorted(list(set(words)))
    labels = sorted(labels)

    training=[]
    output=[]
    out_empty=[0 for _ in range(len(labels))]
    for x , doc in enumerate(docs_x):
        bag=[]
        wrds=[stemmer.stem(w) for w in doc]

        for w in words:
            if w in wrds:
                bag.append(1)
            else:
                bag.append(0)

    output_row=out_empty[:]

```

```

output_row[labels.index(docs_y[x])]=1

training.append(bag)
output.append(output_row)

training = np.array(training)
output=np.array(output)

with open("data.pikle","wb") as f:
    pickle.dump((words,labels,training,output),f)

tf.reset_default_graph()

net=tflearn.input_data(shape=[None,len(training[0])])
net=tflearn.fully_connected(net,8)
net=tflearn.fully_connected(net,8)
net=tflearn.fully_connected(net,len(output[0]),activation="softmax")
net=tflearn.regression(net)

model=tflearn.DNN(net)
try:
    model.load("model.tflearn")
except:
    model.fit(training,output,n_epoch=1000,batch_size=8,show_metric=True)
    model.save("model.tflearn")

def bag_of_words(s,words):
    bag=[0 for _ in range(len(words))]
    s_words=nlk.word.tokenize(s)
    s_words=[stemmer.stem(word.lower()) for word in s_words]

    for se in s_words:
        for i,w in enumerate(words):
            if w== se:
                bag[i]=1

    return np.array(bag)

def chat():
    print("start talking with the bot(type quit to stop)")
    while True:
        inp=input("You:")
        if inp.lower()=="quit":
            break

```

```
results=model.predict([bag_of_words(inp,words)])[0]
results_index=np.argmax(results)
tag=labels[results_index]

if results[results_index]>0.7:
    for tg in data["intents"]:
        if tg['tag']==tag:
            responses=tg['responses']

    print(random.choice(responses))
else:
    print("Sorry I didnt get that")

chat()
```

## CONTRIBUTION FROM MEMBERS:

**1.AMRITPAL:** DATA TRAINNING,LOADING AND EXTRACTING THE DATA,

**2.SUBHANKAR:** STEMMING AND BAG OF WORDS, FINAL REPORT

**3.TARIKUL:** MAKING PREDICTIONS,DATA TRAINING

**4.DEEPAK:** DATA TRAINNING,DEVELOPING THE MODEL, TRAINNING AND SAVING THE MODEL