**Short Documentation**

### **1. Retrieval-based Chatbots that we have created**

A retrieval-based chatbot is one that functions on predefined input patterns and set responses. Once the question/pattern is entered, the chatbot uses a heuristic approach to deliver the appropriate response. The retrieval-based model is extensively used to design goal-oriented chatbots to enhance the customer experience.

### **2. Generative Chatbots**

Unlike retrieval-based chatbots, generative chatbots are not based on predefined responses – they based on seq2seq neural networks.

**For Tranning model**

## **TensorFlow**

Developed by google ,

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks

**About Keras**

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

Keras is a deep learning API written in Python, running on top of the machine learning platform [TensorFlow](https://github.com/tensorflow/tensorflow). It was developed with a focus on enabling fast experimentation

\*\*\*Optimizers are **Classes or methods used to change the attributes of our machine/deep learning model** such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster.

We use keras.seqential model

Keras Sequential Model is to arrange the Keras layers in a sequential order

**Now for pre-processing part**

## What is Natural Language Processing (NLP)?

Natural language processing strives to build machines that understand and respond to text or voice data—and respond with text or speech of their own—in much the same way humans do.

The best example is auto correct on clipboard

NLP involves steps- 1st.Segementations = breaking of paragraphs into

Unit of sentences

2nd. Tokenization = braking sentences into words and assign as token

3rd. Removing special character and stop words

4th.Stemming= means removing prefix and suffix

5th.lemmetizing=**lemmatization** involves deriving the meaning of a word from something like a dictionary, For example, **runs, running, ran** are all forms of the word run, therefore run is the lemma of all these words

The question is how we implement in NLP in our code

For that reason we use NLTK written python

## What is NLTK?

**NLTK (Natural Language Toolkit)** Library is a suite that contains libraries and programs for language processing which contains packages to make machines understand human language and reply to it with an appropriate response.

It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning ect

**\*\*\*\*Stemming and Lemmatization** in Python NLTK are text normalization techniques for Natural Language Processing. These techniques are widely used for text preprocessing. The difference between stemming and lemmatization is that stemming is faster as it cuts words without knowing the context, while lemmatization is slower as it knows the context of words before processing.

1.load dataset intent.json

**2. Preprocess data**

When working with text data, we have performed various preprocessing on the data .

We have done Tokenization first thing we can do on text data. As we know Tokenizing is the process of breaking the whole text into small parts like words.

Here we iterate through the patterns and tokenize the sentence using nltk.word\_tokenize() function and append each word in the words list. We also create a list of classes for our tags

for intent **in** intents['intents']:

**for** pattern **in** intent['patterns']:

#tokenize each word

w = nltk.word\_tokenize(pattern)

words.extend(w)

#add documents in the corpus

documents.append((w, intent['tag']))

# add to our classes list

**if** intent['tag'] not **in** classes:

classes.append(intent['tag'])

Now we have lemmatize each word and remove duplicate words from the list. Lemmatizing means converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.

\*\*\* Python pickle module is used for serializing and de-serializing a Python object structure.

# lemmatize, lower each word and remove duplicates

words = [lemmatizer.lemmatize(w.lower()) **for** w **in** words **if** w not **in** ignore\_words]

words = sorted(list(set(words)))

# sort classes

classes = sorted(list(set(classes)))

# documents = combination between patterns and intents

print (len(documents), "documents")

# classes = intents

print (len(classes), "classes", classes)

# words = all words, vocabulary

print (len(words), "unique lemmatized words", words)

pickle.dump(words,open('words.pkl','wb'))

pickle.dump(classes,open('classes.pkl','wb'))

Now, we have create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn’t understand text so we will convert text into numbers.

# create our training data

training = []

# create an empty array for our output

woutput\_empty = [0] \* len(classes)

# training set, bag of words for each sentence

**for** doc **in** documents:

# initialize our bag of words

bag = []

# list of tokenized words for the pattern

pattern\_words = doc[0]

# lemmatize each word - create base word, in attempt to represent related words

pattern\_words = [lemmatizer.lemmatize(word.lower()) **for** word **in** pattern\_words]

# create our bag of words array with 1, if word match found in current pattern

**for** w **in** words:

bag.append(1) **if** w **in** pattern\_words **else** bag.append(0)

**3. Create training and testing data**

Now, we have created the training data in which we have provided the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn’t understand text so we will convert text into numbers.

# create our training data

training = []

# create an empty array for our output

output\_empty = [0] \* len(classes)

# training set, bag of words for each sentence

**for** doc **in** documents:

# initialize our bag of words

bag = []

# list of tokenized words for the pattern

pattern\_words = doc[0]

# lemmatize each word - create base word, in attempt to represent related words

pattern\_words = [lemmatizer.lemmatize(word.lower()) **for** word **in** pattern\_words]

# create our bag of words array with 1, if word match found in current pattern

**for** w **in** words:

bag.append(1) **if** w **in** pattern\_words **else** bag.append(0)

# output is a '0' for each tag and '1' for current tag (for each pattern)

output\_row = list(output\_empty)

output\_row[classes.index(doc[1])] = 1

training.append([bag, output\_row])

# shuffle our features and turn into np.array

random.shuffle(training)

training = np.array(training)

# create train and test lists. X - patterns, Y - intents

train\_x = list(training[:,0])

train\_y = list(training[:,1])

print("Training data created")

**4. Build the model**

now we have build a deep neural network that has 3 layers. We use the Keras sequential API for this.

\*\*After training the model for 200 epochs, we achieved 100% accuracy on our model. I have saved this model as ‘model.h5’.

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons

# equal to number of intents to predict output intent with softmax

model = Sequential()

model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(train\_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=**True**)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

#fitting and saving the model

hist = model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)

model.save('chatbot\_model.h5', hist)

print("model created")

**5. Predict the response (Graphical User Interface)**

To predict the sentences and get a response from the user

We have to created a new file ‘app.py’. flask GUI file

We load the trained model and then use a GUI that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

Again we import the necessary packages and load the ‘words.pkl’ and ‘classes.pkl’ pickle files which we have created when we trained our model:

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import pickle

import numpy as np

from keras.models import load\_model

model = load\_model('chatbot\_model.h5')

import json

import random

intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl','rb'))

classes = pickle.load(open('classes.pkl','rb'))

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class.

def clean\_up\_sentence(sentence):

# tokenize the pattern - split words into array

sentence\_words = nltk.word\_tokenize(sentence)

# stem each word - create short form for word

sentence\_words = [lemmatizer.lemmatize(word.lower()) **for** word **in** sentence\_words]

**return** sentence\_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

**def** bow(sentence, words, show\_details=**True**):

# tokenize the pattern

sentence\_words = clean\_up\_sentence(sentence)

# bag of words - matrix of N words, vocabulary matrix

bag = [0]\*len(words)

**for** s **in** sentence\_words:

**for** i,w **in** enumerate(words):

**if** w == s:

# assign 1 if current word is in the vocabulary position

bag[i] = 1

**if** show\_details:

print ("found in bag: %s" % w)

**return**(np.array(bag))

**def** predict\_class(sentence, model):

# filter out predictions below a threshold

p = bow(sentence, words,show\_details=**False**)

res = model.predict(np.array([p]))[0]

ERROR\_THRESHOLD = 0.25

results = [[i,r] **for** i,r **in** enumerate(res) **if** r>ERROR\_THRESHOLD]

# sort by strength of probability

results.sort(key=lambda x: x[1], reverse=**True**)

return\_list = []

**for** r **in** results:

return\_list.append({"intent": classes[r[0]], "probability": str(r[1])})

**return** return\_list

After predicting the class, we will get a random response from the list of intents.

def getResponse(ints, intents\_json):

tag = ints[0]['intent']

list\_of\_intents = intents\_json['intents']

**for** i **in** list\_of\_intents:

**if**(i['tag']== tag):

result = random.choice(i['responses'])

break

**return** result

**def** chatbot\_response(text):

ints = predict\_class(text, model)

res = getResponse(ints, intents)

**return** res

For gui we used flask

And the exception is that bot dosent understand these type of words like --

because these don’t have lemma and can’t be tokenize and stemmed so ,

it is kind of ringing for bot . so at that situation it will reply default sentence

and difficulty we have face is that we want to add voice assistance and speech recognition like a voiceful conversation b/w user and bot

but while doing we have destroyed our whole code and there is so much complexity and so we have decided first one thing then other .