Lebanese University

Faculty of Engineering

Branch III

Chat-bot Project

Bot created using NLTK and Keras in Python

Prepared by:

Fatima Kassir 5587

Ali Al Hakim 5470

Presented to:

Dr. Mohammad Awde

Abstract

The natural language, in its oral or textual form, represents the main medium for communicating between human beings. Over time, technological advances have provided new means and tools through which human beings can express themselves and communicate with each other, without altering the original mode of interaction. In recent years, the search for new forms of interaction between users and systems has led to the diffusion of a new possible communication method that exploits a conversational approach based on natural language, which is referred to by using the term chatbot. This paper aims at exploring new chatbot-based conversational interfaces that allow users to exploit the most used interaction strategies by human beings, that is the natural language.
Among the many possible application domains, this paper focuses on the introduction of a chatbot for supporting users in interacting with services for public administration (PA), health and wellbeing and home automation.

Table of contents

Introduction 4

Why Chatbots 5

- Why are chatbots important for a business? 5
- Benefits of chatbots 5
- How does a chatbot work? 6

Dataset 7

RNN & LSTM 7

- What are recurrent neural networks? 7
- How recurrent neural networks learn? 8
- Bidirectional recurrent neural networks 9
- RNN challenges and how to solve them 9
- Long short-term memory units 9

Training code 10

1. Import and load the data file 10
2. Preprocess data 11
3. Create training and testing data 12
4. Build the model 12
5. Predict the response (Graphical User Interface) 13

Testing GUI code 15

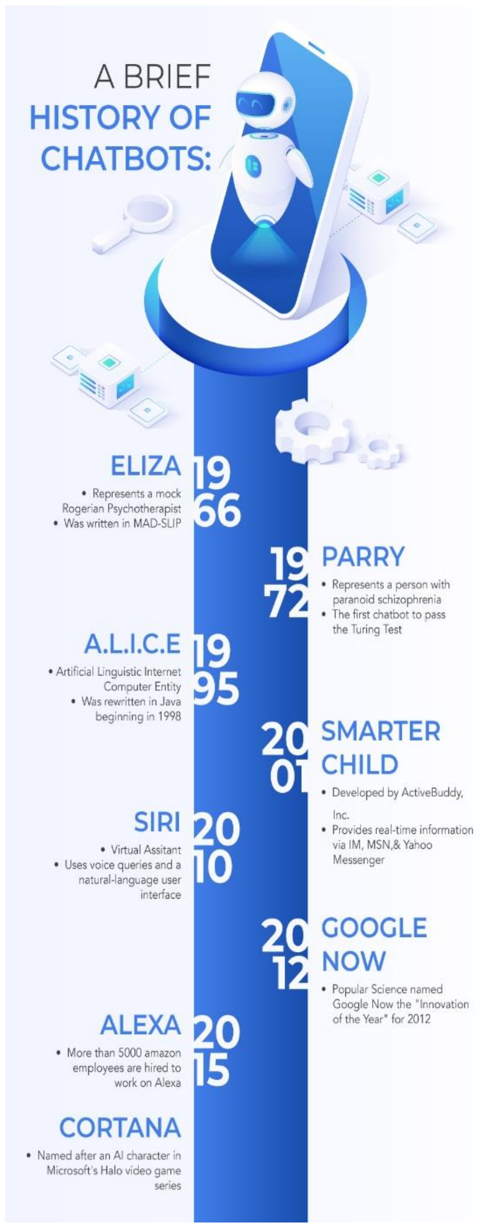
Results & Discussion 16

Conclusion 17

References 17

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 Wiesenbaum 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.
Today, almost all companies have chatbots to engage their users and serve customers by catering to their queries. As per a report by Gartner, Chatbots will be handling 85% of the customer service interactions by the year 2020. Also, 80% of businesses are expected to have some sort of chatbot automation by 2020 (Outgrow, 2018). We practically will have chatbots everywhere, but this doesn't necessarily mean that all will be well-functioning. The challenge here is not to develop a chatbot, but to develop a well-functioning one.


A BRIEF HISTORY OF CHATBOTS:
ELIZA 1966
• Represents a mock Rogerian Psychotherapist
• Was written in MAD-SLIP
1972 PARRY
• Represents a person with paranoid schizophrenia
• The first chatbot to pass the Turing Test
A.L.I.C.E 1995
• Artificial Linguistic Internet Computer Entity
• Was rewritten in Java beginning in 1998
2001 SMARTER CHILD
• Developed by ActiveBuddy, Inc.
• Provides real-time information via IM, MSN,& Yahoo Messenger
SIRI 2010
• Virtual Assitant
• Uses voice queries and a natural-language user interface
2012 GOOGLE NOW
• Popular Science named Google Now the "Innovation of the Year" for 2012
ALEXA 2015
• More than 5000 amazon employees are hired to work on Alexa
CORTANA
• Named after an AI character in Microsoft's Halo video game series

Why Chatbots

A chatbot is an artificial intelligence (AI) application that can imitate a real conversation with a user in their natural language. Chatbots enable communication via text or audio on websites, messaging applications, mobile apps, or telephone.

Chatbots use machine learning to identify communication patterns. With continued interactions with humans, they learn to mimic real-life conversations and react to verbal or written requests, in turn, delivering a particular service. Since chatbots use AI, they understand language, not merely commands. Thus, they become more intelligent as they have more conversations with users. Worth noting is that aside from AI-based chatbots, there are chatbots that use multiple-choice scripts; in essence, option X leads to path Y and so on.

Why are chatbots important for a business?

- Help companies to provide 24-hour service. 64% of respondents to this survey reported that 24-hour service is one of the main advantages of using chatbots. With bots, businesses can ensure that they respond to customer queries regardless of the time, thus boosting customer experience — an essential factor for success in business.
- Allows brands to reach out to more customers. Going by this report, 69% of users prefer bots over apps when interacting with businesses. It, therefore, makes economic sense to use this channel if a brand passes information regarding its products and services to a broader demographic.

Benefits of chatbots

- Better customer engagement. Chatbots shine at engaging both prospects and customers, leading to more sales. While a brand can go overboard with the traditional customer service by providing more information than is required, chatbots only give bits of data based on the input provided by users at each time. They do not end up annoying users with irrelevant information. Instead, they keep the customers engaged for longer by sharing the compelling content flowing.
- Cost savings. Business owners have to pay their employees to manage a chatbot 24/7. These expenditures keep increasing as the business grows and can soon reach uneconomical levels. Chatbots are a one-off investment that assists firms by cutting down on staffing expenses. Brands can easily integrate customer support chatbots to cater to simple concerns of prospective customers and pass on the complex issues to customer service agents.
- Tracking gaining insights and consumer data. Chatbots collect useful feedback from customers, and this information can go a long way to help brands improve their services. Moreover, this feedback can prompt brands to optimize their sites by making changes to low-converting pages.
- Successful lead generation, qualification, and nurturing. Chatbots receive consumer information that enables them to send personalized messages to

customers at various stages of the sales funnel. Bots can ask relevant questions, persuade the prospect, in turn, generate a lead for a brand. Moreover, bots can help businesses to find out unqualified leads via key performance indicators (timeline, relevancy, resources, budget, etc.); thus, barring businesses from dealing with time-consuming leads.

- Easy operation in global markets. Chatbots help enterprises offer excellent customer service in multiple languages all year round. In turn, brands can scale up their operations in new markets.

How does a chatbot work?

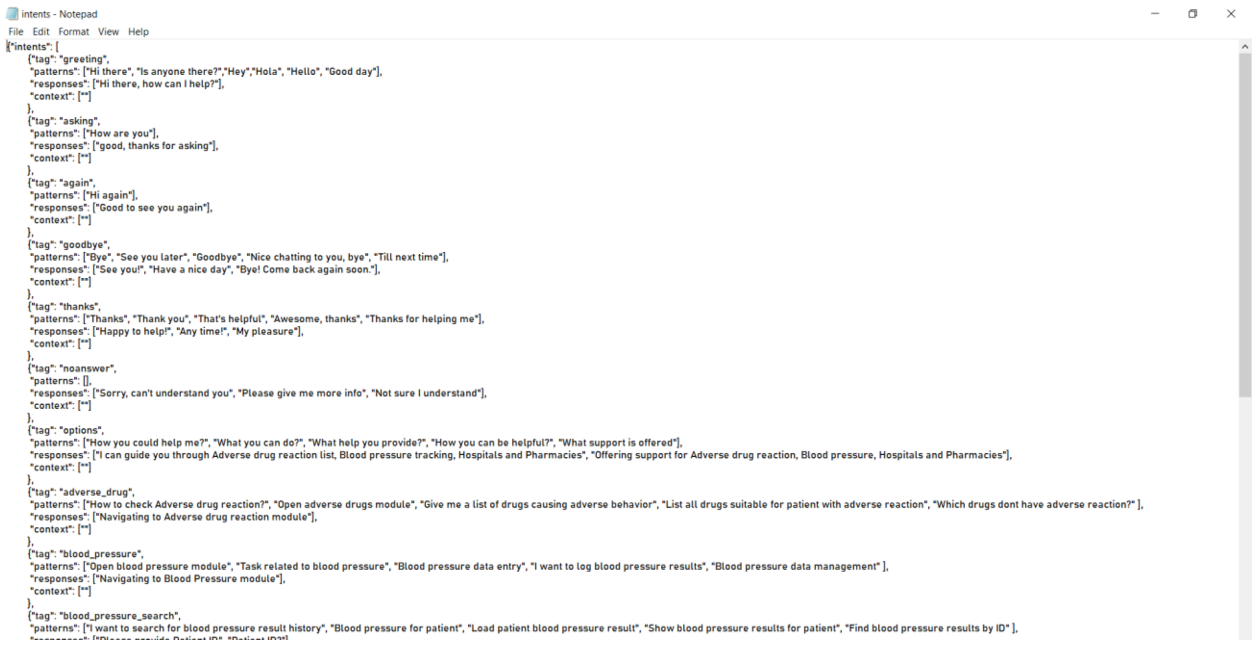
Chatbots primarily use artificial intelligence to talk to people and give relevant content or suggestions. They can function based on a set of instructions or use machine learning. A chatbot that works based on rules is usually quite limited. That is designed to respond to fixed commands. So, if a person asks it the wrong thing, the bot will not understand what the question means, and therefore, will not provide an appropriate response. The intelligence of the bot solely depends on how it is programmed.

On the other hand, a chatbot that uses machine learning works better because it has an artificial brain. The bot understands not only commands but also language. The user, therefore, does not have to use precise words to get accurate or useful responses. Moreover, the bot learns from interactions it has with users and can deal with similar situations when they arise later. It essentially becomes smarter as it talks to more people.

Apart from an AI-based chatbot, there is another type of chatbot that proves useful for marketers. This type of chatbot is more straightforward, even a beginner, can work with it. This bot deals with sending group messages. Brands use it to empower their email marketing and web push strategies. Facebook campaigns allow you to increase outreach, boost sales, and improve customer support.

Dataset

The data file which has predefined patterns and responses is saved in file called intents.json



```
{"intents": [
      {"tag": "greeting",
       "patterns": ["Hi there", "Is anyone there?","Hey","Hola", "Hello", "Good day"],
       "responses": ["Hi there, how can I help?"],
       "context": [""]
      },
      {"tag": "asking",
       "patterns": ["How are you"],
       "responses": ["good, thanks for asking"],
       "context": [""]
      },
      {"tag": "again",
       "patterns": ["Hi again"],
       "responses": ["Good to see you again"],
       "context": [""]
      },
      {"tag": "goodbye",
       "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
       "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],
       "context": [""]
      },
      {"tag": "thanks",
       "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
       "responses": ["Happy to help!", "Any time!", "My pleasure"],
       "context": [""]
      },
      {"tag": "noanswer",
       "patterns": [],
       "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],
       "context": [""]
      },
      {"tag": "options",
       "patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is offered"],
       "responses": ["I can guide you through Adverse drug reaction list, Blood pressure tracking, Hospitals and Pharmacies", "Offering support for Adverse drug reaction, Blood pressure, Hospitals and Pharmacies"],
       "context": [""]
      },
      {"tag": "adverse_drug",
       "patterns": ["How to check Adverse drug reaction?", "Open adverse drugs module", "Give me a list of drugs causing adverse behavior", "List all drugs suitable for patient with adverse reaction", "Which drugs dont have adverse reaction?" ],
       "responses": ["Navigating to Adverse drug reaction module"],
       "context": [""]
      },
      {"tag": "blood_pressure",
       "patterns": ["Open blood pressure module", "Task related to blood pressure", "Blood pressure data entry", "I want to log blood pressure results", "Blood pressure data management" ],
       "responses": ["Navigating to Blood Pressure module"],
       "context": [""]
      },
      {"tag": "blood_pressure_search",
       "patterns": ["I want to search for blood pressure result history", "Blood pressure for patient", "Load patient blood pressure result", "Show blood pressure results for patient", "Find blood pressure results by ID" ],
```

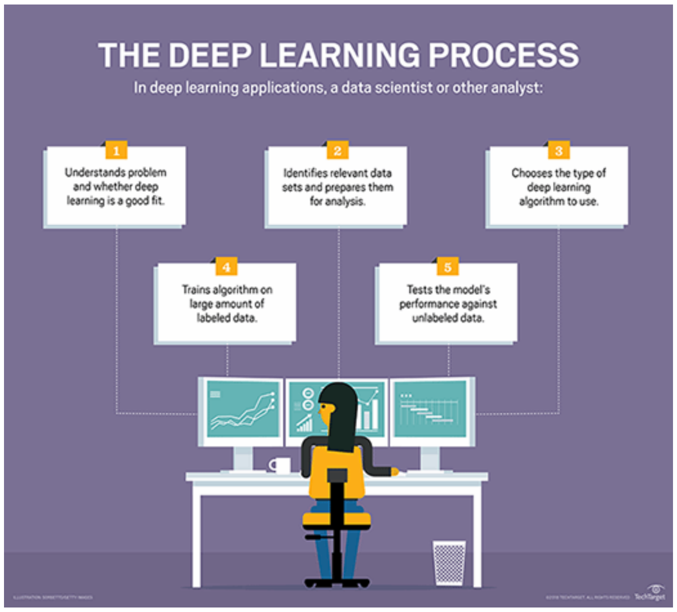
RNN & LSTM

What are recurrent neural networks?

A recurrent neural network is a type of artificial neural network commonly used in speech recognition and natural language processing. Recurrent neural networks recognize data's sequential characteristics and use patterns to predict the next likely scenario.

RNNs are used in deep learning and in the development of models that simulate neuron activity in the human brain. They are especially powerful in use cases where context is critical to predicting an outcome, and are also distinct from other types of artificial neural networks because they use feedback loops to process a sequence of data that informs the final output. These feedback loops allow information to persist. This effect often is described as memory.

RNN use cases tend to be connected to language models in which knowing the next letter in a word or the next word in a sentence is predicated on the data that comes before it. A compelling experiment involves an RNN trained with the works of Shakespeare to produce Shakespeare-like prose successfully. Writing by RNNs is a form of computational creativity. This simulation of human creativity is made possible by the AI's understanding of grammar and semantics learned from its training set.


THE DEEP LEARNING PROCESS
In deep learning applications, a data scientist or other analyst:
1 Understands problem and whether deep learning is a good fit.
2 Identifies relevant data sets and prepares them for analysis.
3 Chooses the type of deep learning algorithm to use.
4 Trains algorithm on large amount of labeled data.
5 Tests the model's performance against unlabeled data.

The deep learning process illustrated.

How recurrent neural networks learn?

Artificial neural networks are created with interconnected data processing components that are loosely designed to function like the human brain. They are composed of layers of artificial neurons -- network nodes -- that have the ability to process input and forward output to other nodes in the network. The nodes are connected by edges or weights that influence a signal's strength and the network's ultimate output.

In some cases, artificial neural networks process information in a single direction from input to output. These "feed-forward" neural networks include convolutional neural networks that underpin image recognition systems. RNNs, on the other hand, can be layered to process information in two directions.

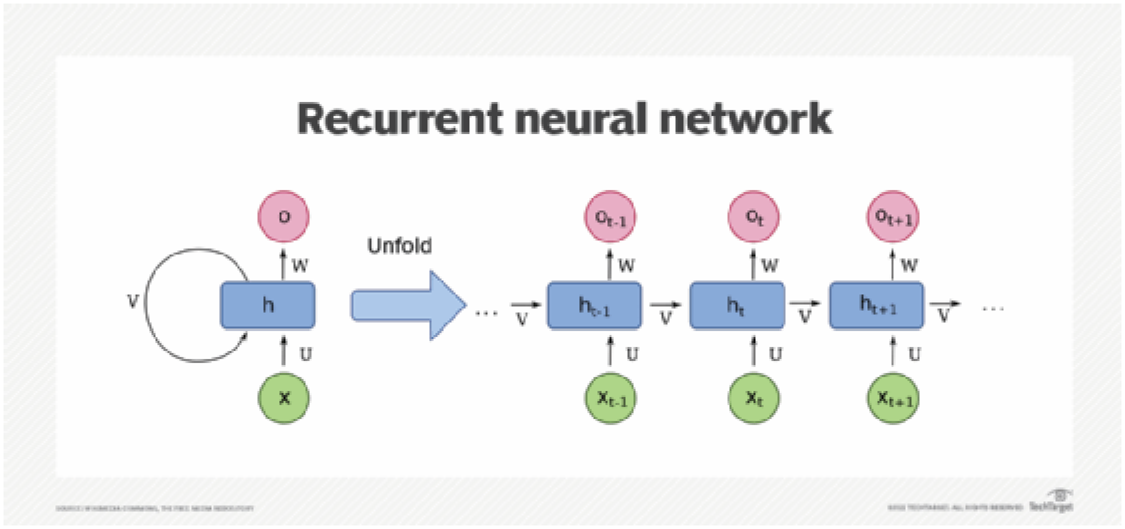
Like feed-forward neural networks, RNNs can process data from initial input to final output. Unlike feed-forward neural networks, RNNs use feedback loops, such as backpropagation through time, throughout the computational process to loop information back into the network. This connects inputs and is what enables RNNs to process sequential and temporal data.

A truncated backpropagation through time neural network is an RNN in which the number of time steps in the input sequence is limited by a truncation of the input sequence. This is useful for recurrent neural networks that are used as sequence-to-sequence models, where the number of steps in the input sequence (or the number of time steps in the input sequence) is greater than the number of steps in the output sequence.

Bidirectional recurrent neural networks

Bidirectional recurrent neural networks (BRNNs) are another type of RNN that simultaneously learn the forward and backward directions of information flow. This is different from standard RNNs, which only learn information in one direction. The process of both directions being learned simultaneously is known as bidirectional information flow.

In a typical artificial neural network, the forward projections are used to predict the future, and the backward projections are used to evaluate the past. They are not used together, however, as in a BRNN.



A diagram -- courtesy of Wikimedia Commons -- depicting a one-unit RNN. The bottom is the input state; middle, the hidden state; top, the output state. U, V, W are the weights of the network. Compressed version of the diagram on the left, unfold version on the right.

RNN challenges and how to solve them

The most common issues with RNNS are gradient vanishing and exploding problems. The gradients refer to the errors made as the neural network trains. If the gradients start to explode, the neural network will become unstable and unable to learn from training data.

Long short-term memory units

One drawback to standard RNNs is the vanishing gradient problem, in which the performance of the neural network suffers because it can't be trained properly. This happens with deeply layered neural networks, which are used to process complex data.

Standard RNNs that use a gradient-based learning method degrade as they grow bigger and more complex. Tuning the parameters effectively at the earliest layers becomes too time-consuming and computationally expensive.

One solution to the problem is called long short-term memory (LSTM) networks, which computer scientists Sepp Hochreiter and Jurgen Schmidhuber invented in 1997. RNNs built with LSTM units categorize data into short-term and long-term memory cells. Doing so enables RNNs to figure out which data is important and should be remembered and looped back into the network. It also enables RNNs to figure out what data can be forgotten.

Training code

The 5 steps to create a chatbot in Python from scratch:

1. Import and load the data file
2. Preprocess data
3. Create training and testing data
4. Build the model
5. Predict the response

1. Import and load the data file

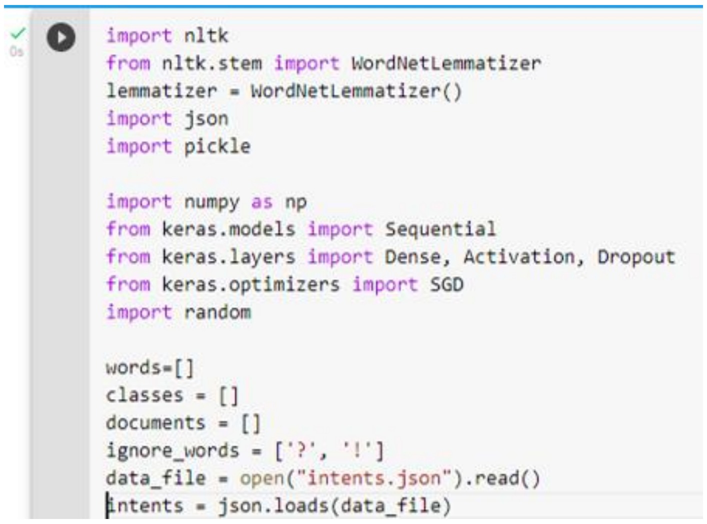
The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP).

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

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

First, make a file name as train_chatbot.py. We import the necessary packages for our chatbot and initialize the variables we will use in our Python project.

The data file is in JSON format so we used the json package to parse the JSON file into Python.



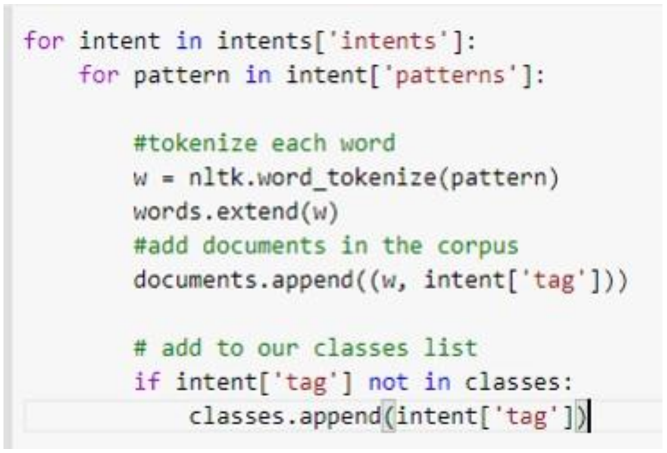
```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open("intents.json").read()
intents = json.loads(data_file)
```

2. Preprocess data

When working with text data, we need to perform various preprocessing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to preprocess the data.

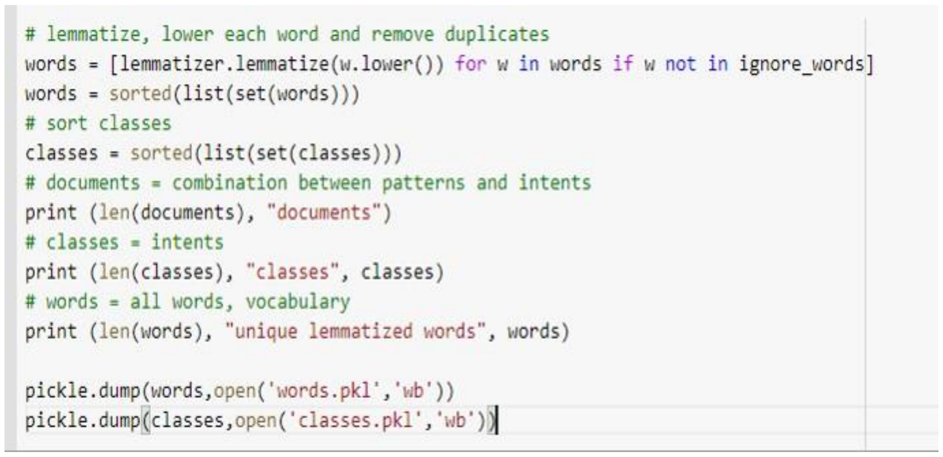
Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.

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'])
```

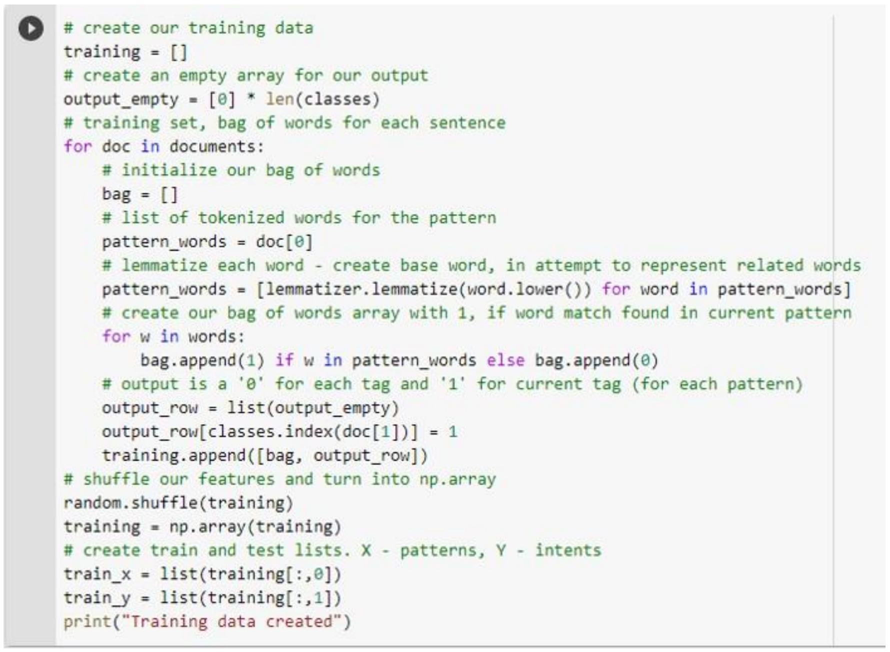
Now we will lemmatize each word and remove duplicate words from the list. Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.



```
# 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'))
```

3. Create training and testing data

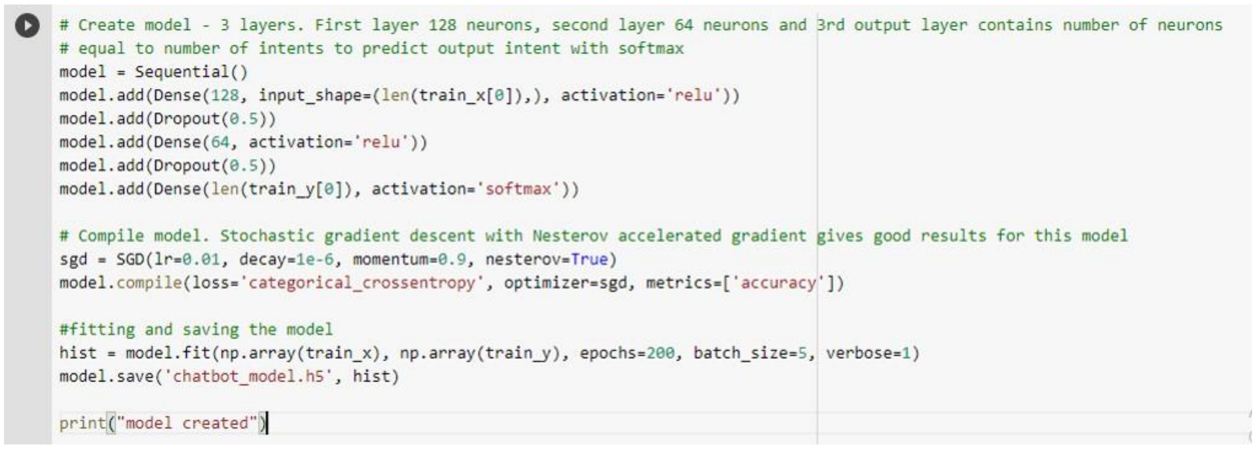
Now, we will create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn't understand text so we will convert text into numbers.



```
# create our training data
training = []
# create an empty array for our output
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

We have our training data ready, now we will 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. Let us save the model as 'chatbot_model.h5'.



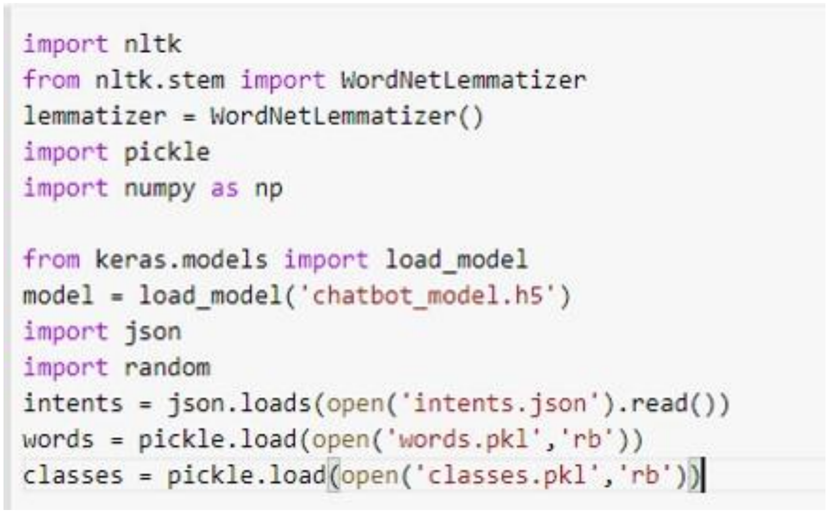
```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
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 to let us create a new file 'chatapp.py'.

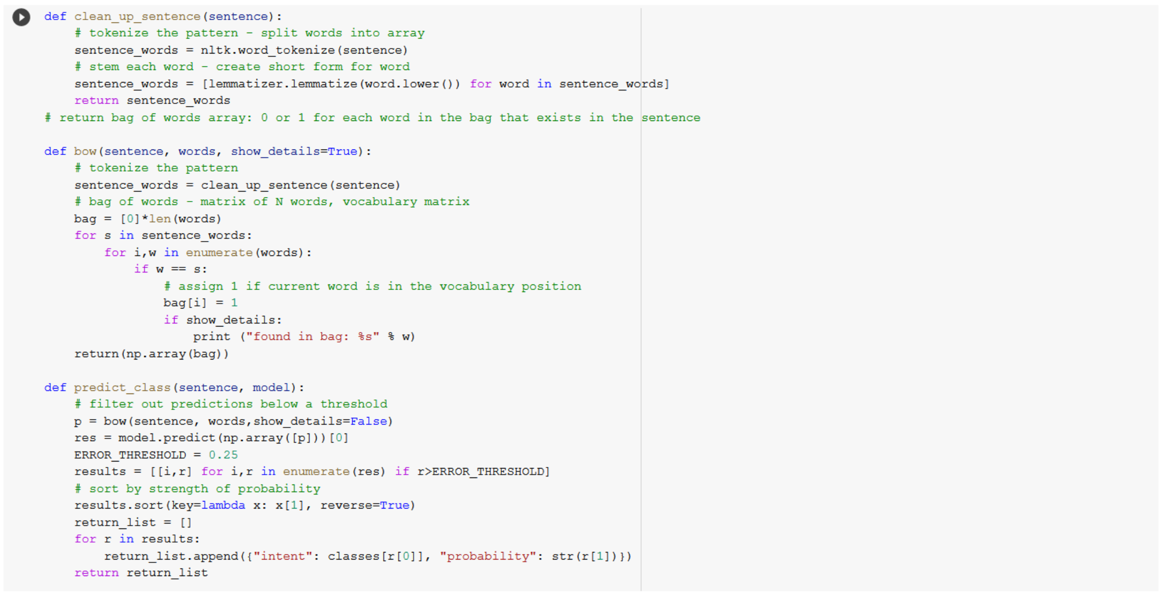
We will load the trained model and then use a graphical user interface that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

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:



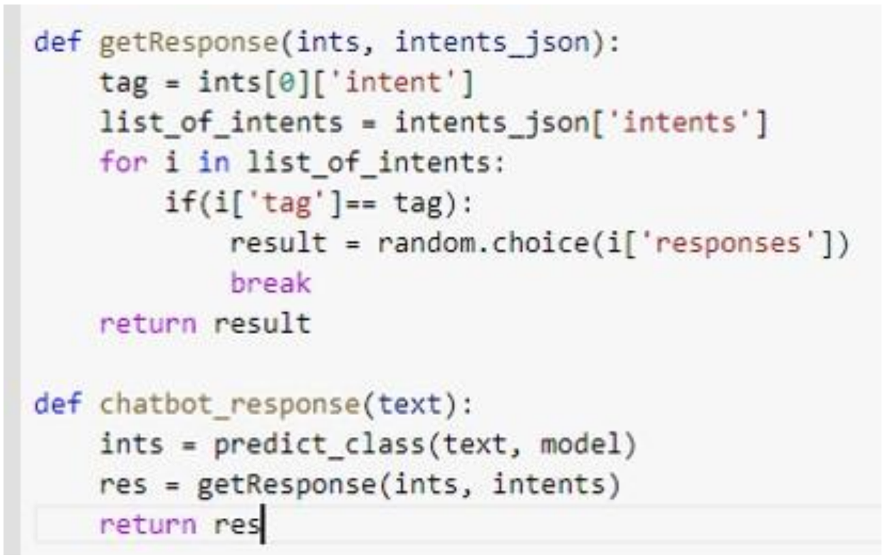
```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

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.



```python
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
```

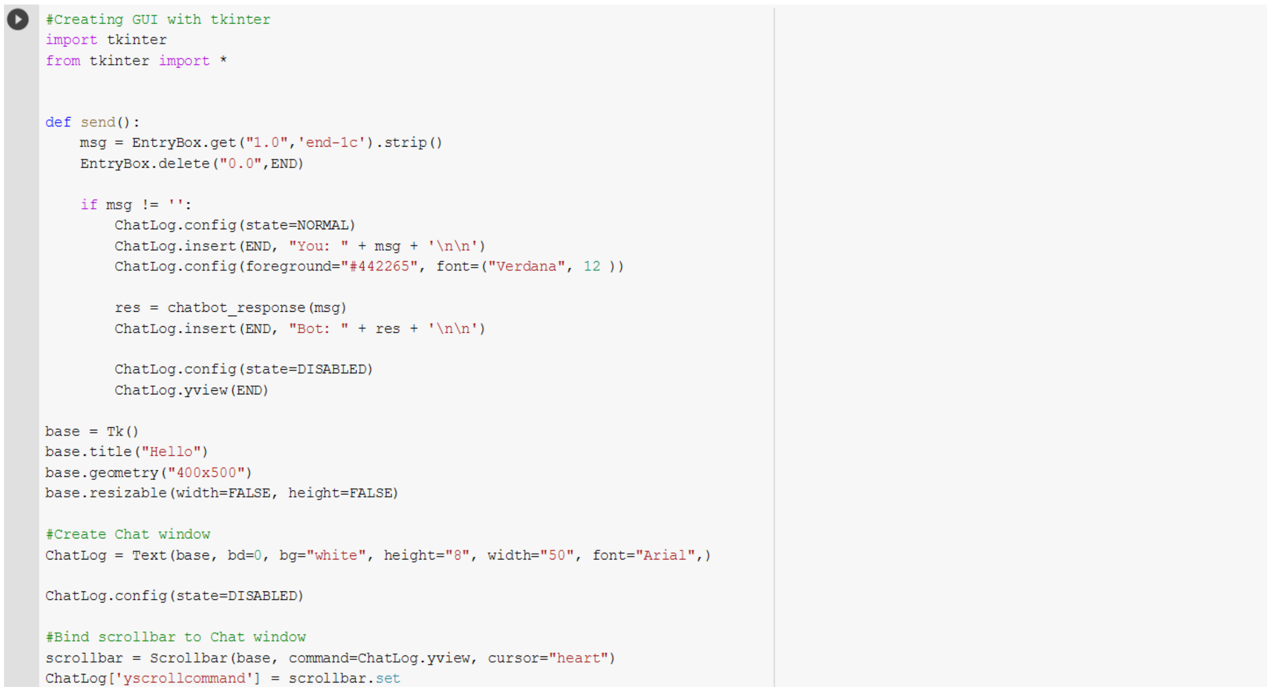
After predicting the class, we will get a random response from the list of intents.



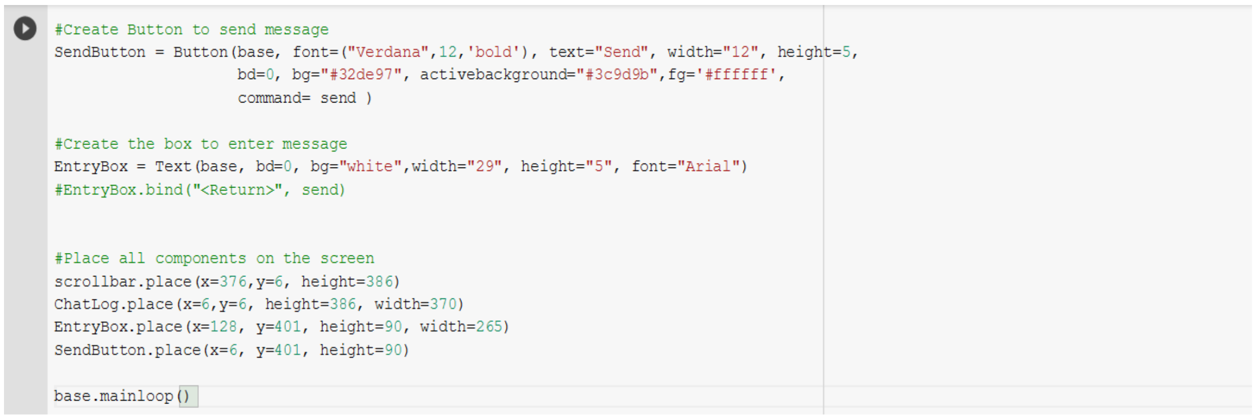
```python
def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(text):
    ints = predict_class(text, model)
    res = getResponse(ints, intents)
    return res
```

Testing GUI code

Now we will develop a graphical user interface. Let's use Tkinter library which is shipped with tons of useful libraries for GUI. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI. Here is the full source code for the GUI.

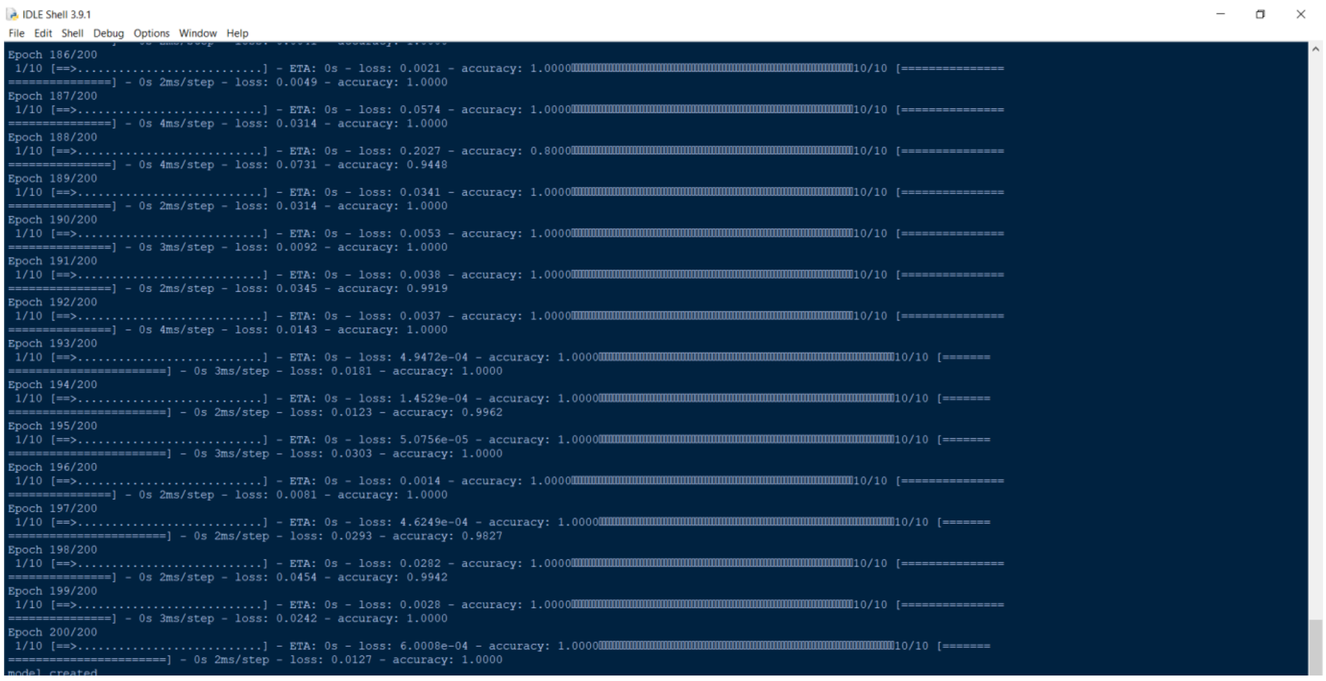


```python
#Creating GUI with tkinter
import tkinter
from tkinter import *


def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        res = chatbot_response(msg)
        ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)

base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set
```

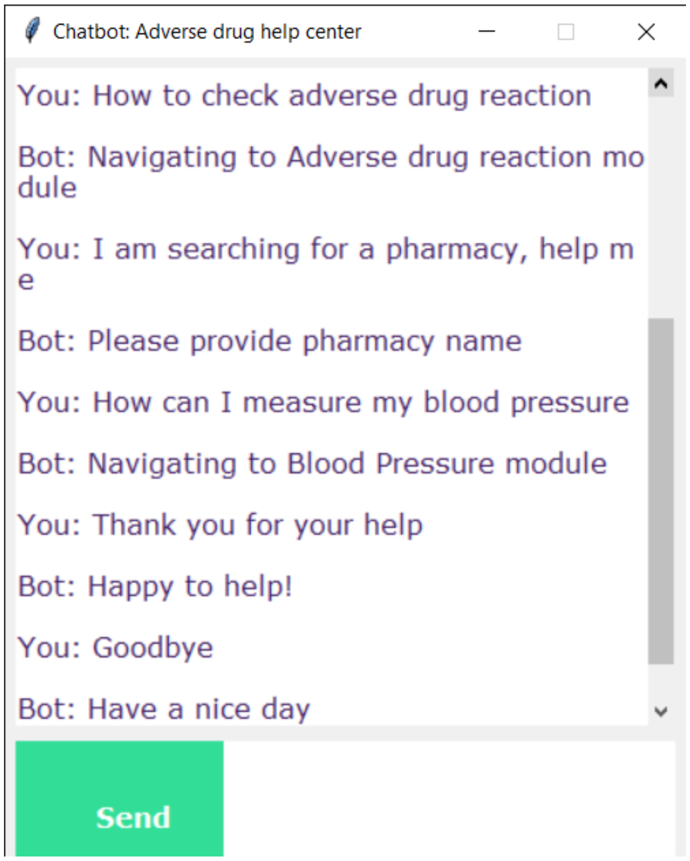


```python
#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

Results & Discussion



```
Epoch 186/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0021 - accuracy: 1.000010/10 [===============
===============] - 0s 2ms/step - loss: 0.0049 - accuracy: 1.0000
Epoch 187/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0574 - accuracy: 1.000010/10 [===============
===============] - 0s 4ms/step - loss: 0.0314 - accuracy: 1.0000
Epoch 188/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.2027 - accuracy: 0.800010/10 [===============
===============] - 0s 4ms/step - loss: 0.0731 - accuracy: 0.9448
Epoch 189/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0341 - accuracy: 1.000010/10 [===============
===============] - 0s 2ms/step - loss: 0.0314 - accuracy: 1.0000
Epoch 190/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0053 - accuracy: 1.000010/10 [===============
===============] - 0s 3ms/step - loss: 0.0092 - accuracy: 1.0000
Epoch 191/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0038 - accuracy: 1.000010/10 [===============
===============] - 0s 2ms/step - loss: 0.0345 - accuracy: 0.9919
Epoch 192/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0037 - accuracy: 1.000010/10 [===============
===============] - 0s 4ms/step - loss: 0.0143 - accuracy: 1.0000
Epoch 193/200
 1/10 [==>...........................] - ETA: 0s - loss: 4.9472e-04 - accuracy: 1.000010/10 [=======
=======================] - 0s 3ms/step - loss: 0.0181 - accuracy: 1.0000
Epoch 194/200
 1/10 [==>...........................] - ETA: 0s - loss: 1.4529e-04 - accuracy: 1.000010/10 [=======
=======================] - 0s 2ms/step - loss: 0.0123 - accuracy: 0.9962
Epoch 195/200
 1/10 [==>...........................] - ETA: 0s - loss: 5.0756e-05 - accuracy: 1.000010/10 [=======
=======================] - 0s 3ms/step - loss: 0.0303 - accuracy: 1.0000
Epoch 196/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0014 - accuracy: 1.000010/10 [===============
===============] - 0s 2ms/step - loss: 0.0081 - accuracy: 1.0000
Epoch 197/200
 1/10 [==>...........................] - ETA: 0s - loss: 4.6249e-04 - accuracy: 1.000010/10 [=======
=======================] - 0s 2ms/step - loss: 0.0293 - accuracy: 0.9827
Epoch 198/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0282 - accuracy: 1.000010/10 [===============
===============] - 0s 2ms/step - loss: 0.0454 - accuracy: 0.9942
Epoch 199/200
 1/10 [==>...........................] - ETA: 0s - loss: 0.0028 - accuracy: 1.000010/10 [===============
===============] - 0s 3ms/step - loss: 0.0242 - accuracy: 1.0000
Epoch 200/200
 1/10 [==>...........................] - ETA: 0s - loss: 6.0008e-04 - accuracy: 1.000010/10 [=======
=======================] - 0s 2ms/step - loss: 0.0127 - accuracy: 1.0000
model created
```



You: How to check adverse drug reaction

Bot: Navigating to Adverse drug reaction mo
dule

You: I am searching for a pharmacy, help m
e

Bot: Please provide pharmacy name

You: How can I measure my blood pressure

Bot: Navigating to Blood Pressure module

You: Thank you for your help

Bot: Happy to help!

You: Goodbye

Bot: Have a nice day

As we can see from results above, the accuracy was around 100% after training our model with 200 epochs with a low learning rate to achieve the better accuracy and the minimum loss. While using the GUI, and after talking with the bot, we can see that the bot can recognize some new sentences after get trained and can give the right answer to every question asked.

Conclusion

In this Python data science project, we understood about chatbots and implemented a deep learning version of a chatbot in Python which is accurate. You can customize the data according to business requirements and train the chatbot with great accuracy. Chatbots are used everywhere and all businesses are looking forward to implementing bot in their workflow. Bots will help humans in future a lot to make their life easier and easier.

References

- https://data-flair.training/blogs/python-chatbot-project/
- https://searchenterpriseai.techtarget.com/definition/recurrent-neural-networks
- https://towardsdatascience.com/machine-learning-recurrent-neural-networks-and-long-short-term-memory-lstm-python-keras-example-86001ceaaebc
- https://sendpulse.com/support/glossary/chatbot
- http://ceur-ws.org/Vol-2101/paper8.pdf
- https://research.aimultiple.com/chatbot-applications/
- https://www.mygreatlearning.com/blog/basics-of-building-an-artificial-intelligence-chatbot/
- https://www.intellectsoft.net/blog/how-to-build-a-chatbot/