Importing and loading the data file

In [1]:

```
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 tensorflow.keras.optimizers import SGD
import random

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

Data Preprocessing

In [2]:

```
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'])
```

In [3]:

```
# 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'))
```

```
122 documents
44 classes ['Bill', 'HR_related_problem', 'Location', 'Weather', 'about',
'appointment status', 'cabin', 'check_leave', 'college', 'commission', 'co
nfiguration', 'connect_people', 'customer_satisfaction', 'domain', 'email_
id', 'factors_impacting_sale', 'forgot_password', 'gadgets', 'goodbye', 'g
reeting', 'highest_grossing', 'hours', 'invalid', 'key_customers', 'leav
e', 'maintainence', 'manufacturing_problems', 'meeting_timings', 'missing_
id', 'name', 'noans', 'options', 'order_components', 'order_tracking', 'pr
edict_delay', 'predict_performance', 'project_handling_queries', 'search_d
epartment', 'search_person_by_id', 'solve_problems', 'supplier_info', 'tha
nks', 'turnover', 'version_update']
239 unique lemmatized words ["'s", ',', '.', '23a12', '23a31', '32712', '3
45a23', '431b67', '561a24', '562b78', '@', 'a', 'abc', 'abx', 'accident',
'ai', 'am', 'an', 'analysis', 'and', 'anyone', 'appointment', 'appoitmen
t', 'are', 'at', 'awesome', 'based', 'be', 'been', 'benefit', 'bhatt', 'bi
ll', 'bored', 'bye', 'cabin', 'cafeteria', 'call', 'can', 'canteen', 'cha
t', 'chatting', 'clarity', 'clear', 'college', 'commission', 'company', 'c
ompensation', 'complaint', 'component', 'comprises', 'computer', 'configur
ation', 'configure', 'conflict', 'could', 'customer', 'date', 'day', 'dela
yed', 'delivery', 'demand', 'department', 'design', 'desktop', 'developmen
t', 'do', 'doe', 'doing', 'domain', 'duedate', 'each', 'electricity', 'emp
loyee', 'factor', 'feedback', 'find', 'fixed', 'for', 'forgets', 'forgot',
'from', 'gadget', 'gmail.com', 'good', 'goodbye', 'grossing', 'guide', 'h
a', 'handled', 'happy', 'have', 'head', 'hello', 'help', 'helpful', 'helpi
ng', 'hey', 'hi', 'highest', 'hola', 'hour', 'how', 'hr/it/projects', 'i',
'id', 'impact', 'impacting', 'improve', 'improved', 'improving', 'in', 'in
formation', 'insufficient', 'is', 'issue', 'it', 'job', 'key', 'kiit.ac.i
n', 'know', 'knowledge', 'kumar', 'lack', 'laptop', 'last', 'later', 'leav
e', 'legal', 'like', 'list', 'locate', 'location', 'login', 'love', 'maint
ainence', 'management', 'manish', 'manoj', 'manufacturer', 'marry', 'me',
'meet', 'meeting', 'member', 'michel', 'miscommunication', 'my', 'name',
'nandi', 'nearby', 'need', 'needed', 'next', 'nice', 'not', 'occured', 'o
f', 'office', 'on', 'open', 'opening', 'order', 'our', 'out', 'password',
'planning', 'present', 'pricipal', 'problem', 'product', 'profit', 'projec
t', 'provide', 'query', 'raised', 'rakesh', 'rate', 'recorded', 'related',
'resolved', 'response', 'risk', 'roy', 'sale', 'see', 'set', 'shantanu',
'shared', 'sharma', 'should', 'siddhart', 'skill', 'skilled', 'software',
'someone', 'specification', 'step', 'stock', 'sujata', 'supplier', 'targe
t', 'team', 'thank', 'thanks', 'that', 'the', 'there', 'this', 'till', 'ti
me', 'to', 'today', 'track', 'training', 'turnover', 'updated', 'updatio
n', 'urgently', 'user', 'various', 'version', 'vp', 'wa', 'want', 'we', 'w
eather', 'what', 'when', 'where', 'which', 'who', 'why', 'wifi', 'with',
'workforce', 'working', 'year', 'you', 'your']
```

Creating training and testing data

In [4]:

```
# 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")
```

```
Training data created

C:\Users\PRATHI~1\AppData\Local\Temp/ipykernel_8916/2748295590.py:24: Visi
bleDeprecationWarning: Creating an ndarray from ragged nested sequences (w
hich is a list-or-tuple of lists-or-tuples-or ndarrays with different leng
ths or shapes) is deprecated. If you meant to do this, you must specify 'd
type=object' when creating the ndarray.
  training = np.array(training)
```

Building the model

In [5]:

```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd outpu
# 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 goo
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
model.save('chatbot_model.h5', hist)

print("model created")
```

```
Epoch 128/200
25/25 [==============================] - 0s 2ms/step - loss: 0.3274 - a
ccuracy: 0.8689
Epoch 129/200
25/25 [==============================] - 0s 2ms/step - loss: 0.2292 - a
ccuracy: 0.9180
Epoch 130/200
25/25 [==============================] - 0s 2ms/step - loss: 0.1738 - a
ccuracy: 0.9590
Epoch 131/200
25/25 [==============================] - 0s 2ms/step - loss: 0.2859 - a
ccuracy: 0.9016
Epoch 132/200
25/25 [==============================] - 0s 2ms/step - loss: 0.2454 - a
ccuracy: 0.9344
Epoch 133/200
25/25 [==============================] - 0s 2ms/step - loss: 0.2182 - a
ccuracy: 0.9508
Epoch 134/200
25/25 [==============================] - 0s 2ms/step - loss: 0.2168 - a
```

Predicting the response (GUI)

In [6]:

```
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'))
```

In [7]:

```
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
```

In [8]:

```
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
```

In [9]:

```
#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("ChatBotHero")
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)
ChatLog['yscrollcommand'] = scrollbar.set

#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=130, y=401, height=40, width=265)
SendButton.place(x=6, y=401, height=40)

base.mainloop()
```

```
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
```

In [ ]:

```

```

In [ ]:

```

```