



# NLP and Speech Recognition Chatbot

Course-End Certification Project



Get Certified. Get Ahead.

---

## NLP and Speech Recognition Chatbot

### Problem Statement:

A company holds an event that has been given the deserved promotion through marketing in hopes of attracting as big an audience as possible. Now, it's up to the customer support team to guide the audience and answer any queries. Providing high-quality support and guidance is the challenge. The chatbot is very helpful for its 24/7 presence and ability to reply instantly.

**Objective:** To develop a real-time chatbot to engage with the customers in order to boost their business growth by using NLP and Speech Recognition.

**Domain:** Customer Support

**Analysis to be done:** Create a set of prebuilt commands or inputs as a dataset. Here, we use command .json as Dataset that contains the patterns we need to find and the responses we want to return to the user.

### Steps to perform:

#### Tasks -

1. install all of the required modules using pip

Before you proceed, you need to install before python packages.

- ✓ nltk toolkit, for all the text preprocessing and combinations of required words that add value to sentence in order to do a required search
- ✓ TensorFlow, keras, pickle – Backend to create a required model
- ✓ speech\_recognition – for speech recognition and then converting that into text

Commands to install the packages

```
pip install nltk
```

```
pip install keras
```

```
pip install SpeechRecognition
```

```
pip install tensorflow
```

```
pip install pickle
```

2. You can get started with code. first will show the files are being used -

- ✓ **commands.json** – The data file with predefined patterns and responses
- ✓ **Create\_ModelChatbot.ipynb** – Notebook file to build a model and train with the dataset
- ✓ **Words.pkl** – Pickle file that stores the words Python object that contains a list of our vocabulary
- ✓ **Classes.pkl** – Pickle file contains the list of categories

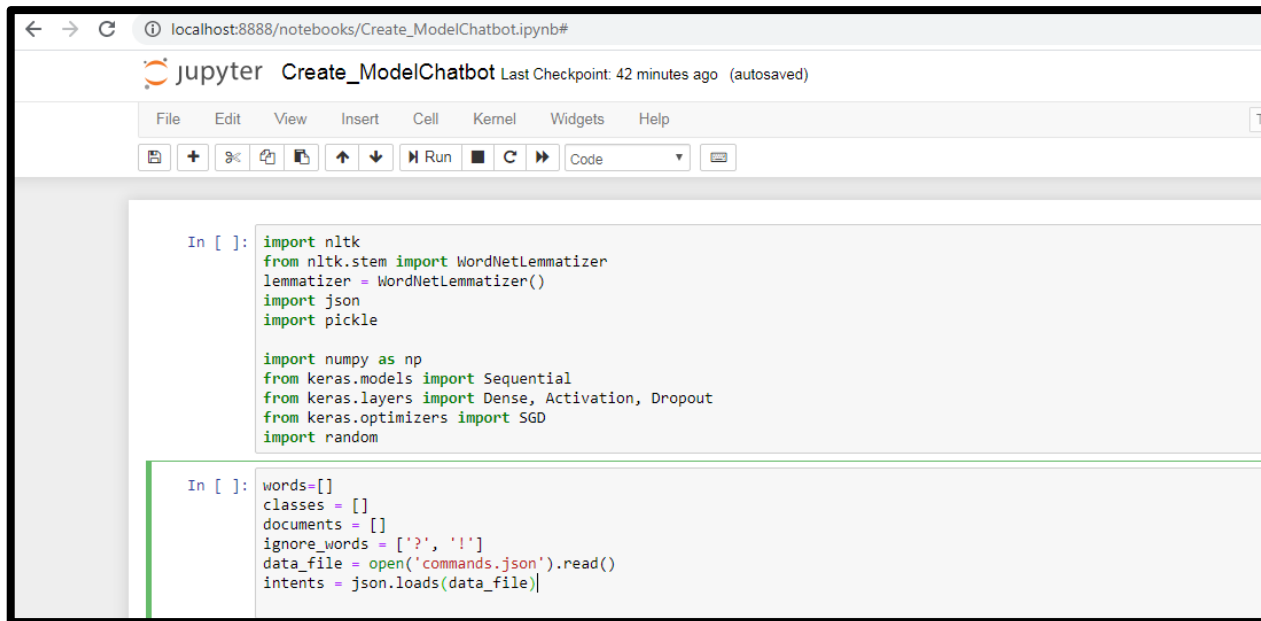
- ✓ **Chatbot\_model.h5** – This is a trained model that has been developed using a deep learning approach
- ✓ **ChatBotScript.ipynb** – This is the main Notebook which is intended to build the functions that have both Speech recognition to listen to user commands and function which returns the response

3. Here's the main function with readCommand() and talk function.

- ✓ **Import and load the data file** Load all the necessary packages and the json file which you used as a dataset

```
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('commands.json').read()
intents = json.loads(data_file)
```



```
In [ ]: import nltk
        from nltk.stem import WordNetLemmatizer
        lemmatizer = WordNetLemmatizer()
        import json
        import pickle

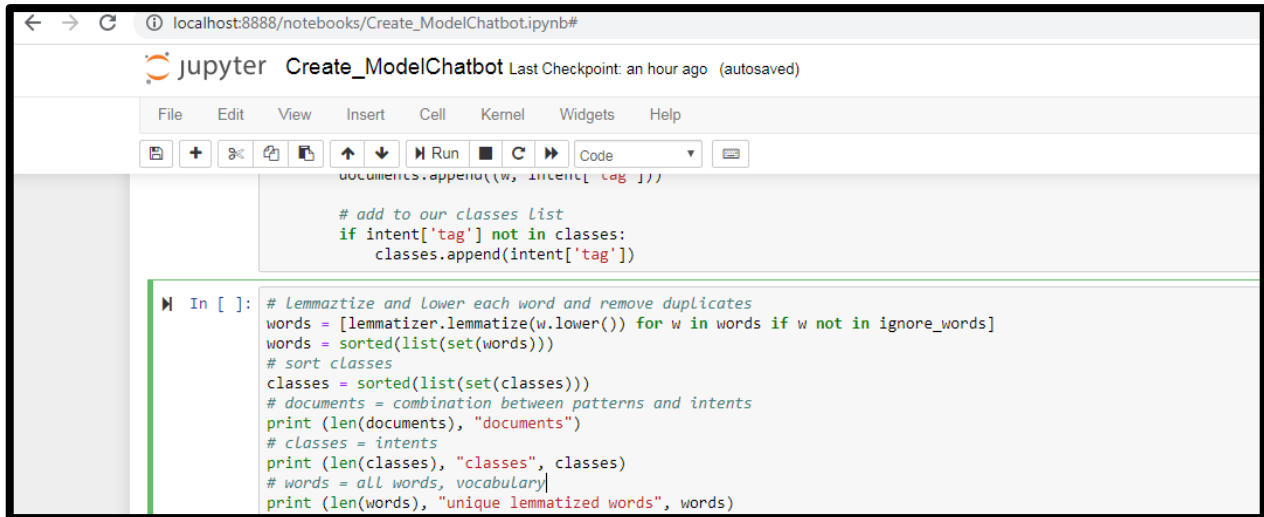
        import numpy as np
        from keras.models import Sequential
        from keras.layers import Dense, Activation, Dropout
        from keras.optimizers import SGD
        import random

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

### ✓ Preprocess Data :

When you use text data as your dataset you need to perform certain preprocessing rules before building any machine learning or deep learning model. Tokenizing is the very first step you can use for text preprocessing. Tokenizing is the process of breaking the whole text into small parts like words. Here you iterate through the patterns and tokenize the sentence using `nltk.word_tokenize()` function and append each word in the words list. You also create a list of classes for your tags.

```
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)
```



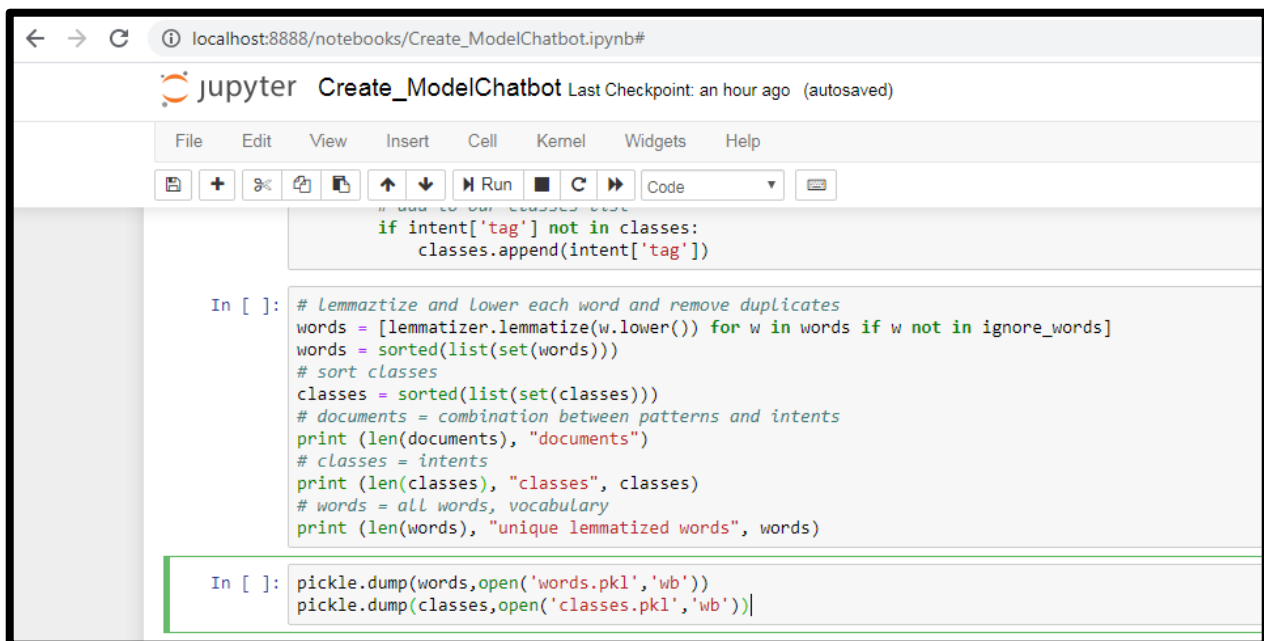
The screenshot shows a Jupyter Notebook titled 'Create\_ModelChatbot' with a last checkpoint of 'an hour ago (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The code in the notebook is as follows:

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

# add to our classes list
if intent['tag'] not in classes:
    classes.append(intent['tag'])

In [ ]: # Lemmatize and 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)
```

Now lemmatize each word and remove duplicate words from the list. Then, create a pickle file which will be used as predictions.



The screenshot shows the same Jupyter Notebook interface. The code in the notebook is as follows:

```
# add to our classes list
if intent['tag'] not in classes:
    classes.append(intent['tag'])

In [ ]: # Lemmatize and 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)

In [ ]: pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

- ✓ Creating Training and Testing Dataset: Create the training data in which you will provide the input and the output. You will use the pattern as the input and class your input pattern belongs to will be our output. Then we will convert this text into numbers.

# Training Data Creation

training = []

# Define Output Array

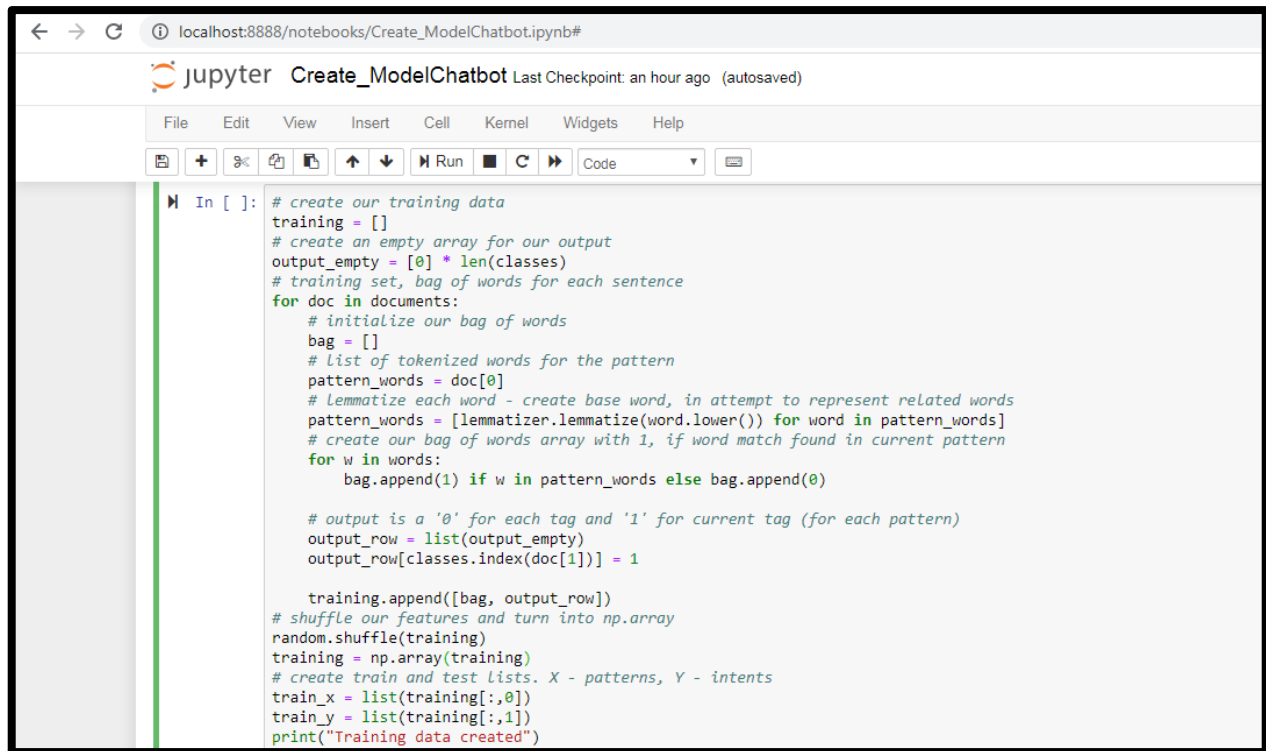
output\_empty = [0] \* len(classes)

---

```
# Training data as Bag of words for respective sentence
for doc in documents:
    # Bag of words Initialization
    bag = []
    # tokenized word list as pattern
    pattern_words = doc[0]
    # lemmatize the words and define the meaning
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # Creation of bag of words array with 1, if it matches in current pattern
    for w in words:
        bag.append(1 if w in pattern_words else 0)

    # output is a '0' for each tag and '1' for each pattern
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
# Conversion to np.array
random.shuffle(training)
training = np.array(training)
# create train and test dataset
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```



```
In [ ]: # 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")
```

✓ **Build the Model** : Since your training data is ready, build the deep neural network model that has 3 layers for which you will use Keras Sequential API.

# Create model - 3 layers neural network

```
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.

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

```
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

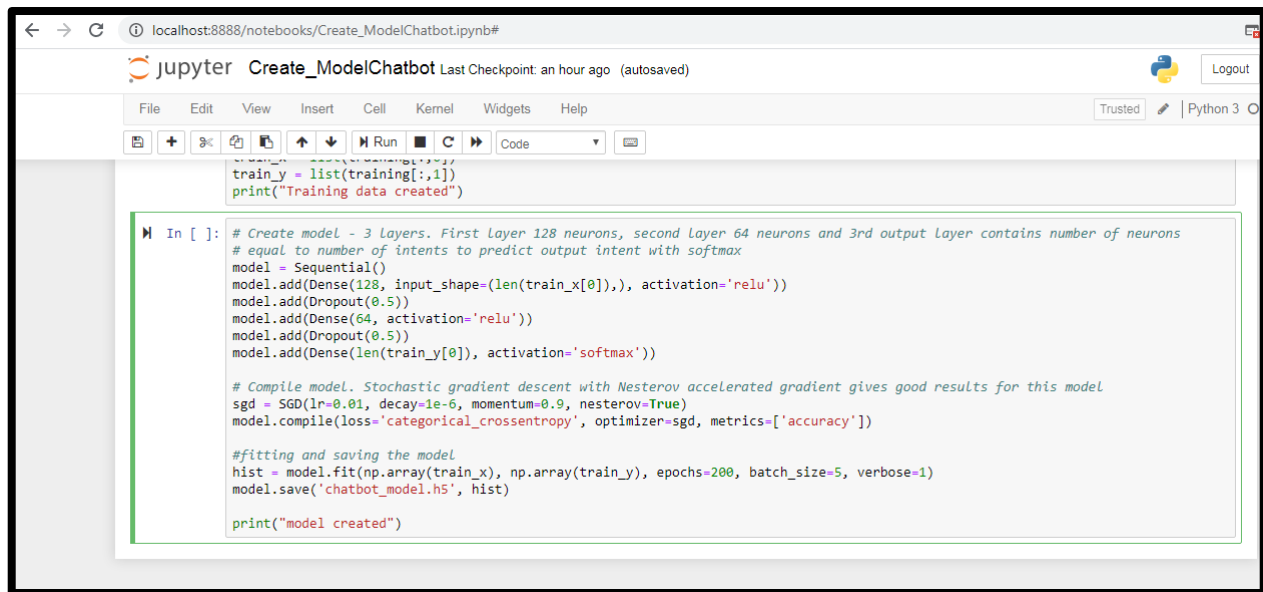
#fitting and saving the model

```
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")
```





```
train_y = list(training[:,1])
print("Training data created")

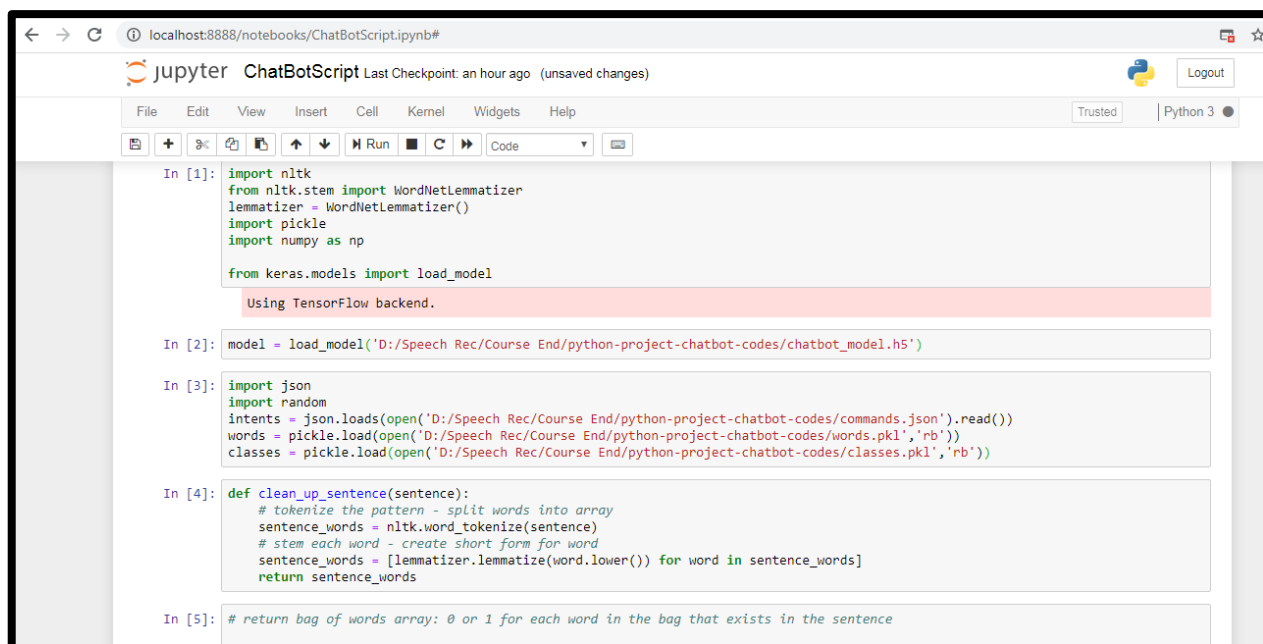
In [ ]: # Create model - 3 Layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
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")
```

✓ **Predict the Responses (ChatBotScript.ipynb):** Since the model is ready, you are going to load the trained model and use it for predictions for all the commands that have been provided by the user.



```
In [1]: import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model

Using TensorFlow backend.

In [2]: model = load_model('D:/Speech Rec/Course End/python-project-chatbot-codes/chatbot_model.h5')

In [3]: import json
import random
intents = json.loads(open('D:/Speech Rec/Course End/python-project-chatbot-codes/commands.json').read())
words = pickle.load(open('D:/Speech Rec/Course End/python-project-chatbot-codes/words.pkl', 'rb'))
classes = pickle.load(open('D:/Speech Rec/Course End/python-project-chatbot-codes/classes.pkl', 'rb'))

In [4]: 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

In [5]: # return bag of words array: 0 or 1 for each word in the bag that exists in the sentence
```