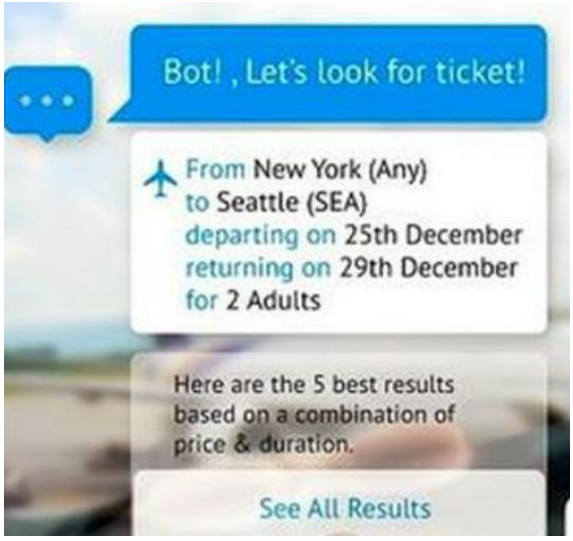
Advanced Machine Learning Project - Chatbot

Kent State University
Course: MIS-64061
Professor: Dr. Murali Shanker
Student: Tanmoy Kanti Kumar

The objective of this assignment is to research on a deep learning model and define, build a protype.

Definition:

Our aim is to research on Chatbots and try to build a simple one. This will be a stepping stone to build a more advanced one.



A chatbot is an artificial intelligence-powered piece of software in a device (Siri, Alexa, Google Assistant etc), application, website or other networks that try to gauge consumer's needs and then assist them to perform a particular task like a commercial transaction, hotel booking, form submission etc. Today almost every company has a chatbot deployed to engage with the users. Some of the ways in which companies are using chatbots are:

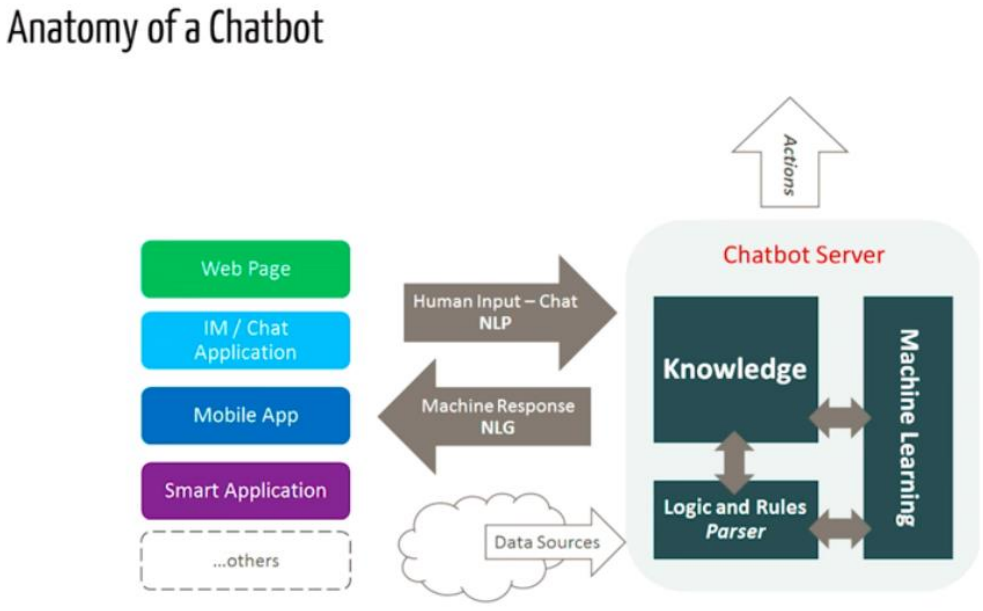
- To deliver flight information
- to connect customers and their finances
- As customer support
- The possibilities are (almost) limitless.

There are broadly two variants of chatbots: **Rule-Based** and **Self-learning**.

- In a Rule-based approach, a bot answers questions based on some rules on which it is trained on. The rules defined can be very simple to very complex. The bots can handle simple queries but fail to manage complex ones.
- Self-learning bots are the ones that use some Machine Learning-based approaches and are definitely more efficient than rule-based bots. These bots can be of further two types: Retrieval Based or Generative.
    - In retrieval-based models, a chatbot uses some heuristic to select a response from a library of predefined responses. The chatbot uses the message and context of the conversation for selecting the best response from a predefined list of bot messages. The context can include a current position in the dialogue tree, all previous messages in the conversation, previously saved variables (e.g. username). Heuristics for selecting a response can be engineered in many different ways, from rule-based if-else conditional logic to machine learning classifiers.
    - Generative bots can generate the answers and not always replies with one of the answers from a set of answers. This makes them more intelligent as they take word by word from the query and generates the answers.

Concepts:

At first, Chatbot can look like a normal app. There is an application layer, a database and APIs to call external services. In a case of the chatbot, UI is replaced with chat interface. While Chatbots are easy to use for users, it adds complexity for the app to handle.



For a chatbot, the biggest challenge is with the input text data. However, Machine learning algorithms need some sort of numerical feature vector in order to perform the task. So, before we start with any NLP project, we need to pre-process it to make it ideal for work. Natural Language processing (NLP) Chatbot takes some combination of steps to convert the customer's text or speech into structured data that is used to select the related answer. Some of the Natural Language Processing steps are:

- Sentiment Analysis: Tries to learn if the user is having a good experience or if the after some point the chat should be forwarded to the human.
- Tokenization: The NLP divides a string of words into pieces or tokens that are linguistically symbolic or are differently useful for the application.
- Named Entity Recognition: The chatbot program model looks for categories of words, like the name of the product, the user's name or address, whichever data is required.
- Normalization: The Chatbot program model processes the text in an effort to find common spelling mistakes or typographical errors that might the user intent to convey. This gives more human like effect of the Chatbot to the users.
- Dependency Parsing: The Chatbot looks for the objects and subjects- verbs, nouns and common phrases in the user's text to find dependent and related phrases that users might be trying to convey.

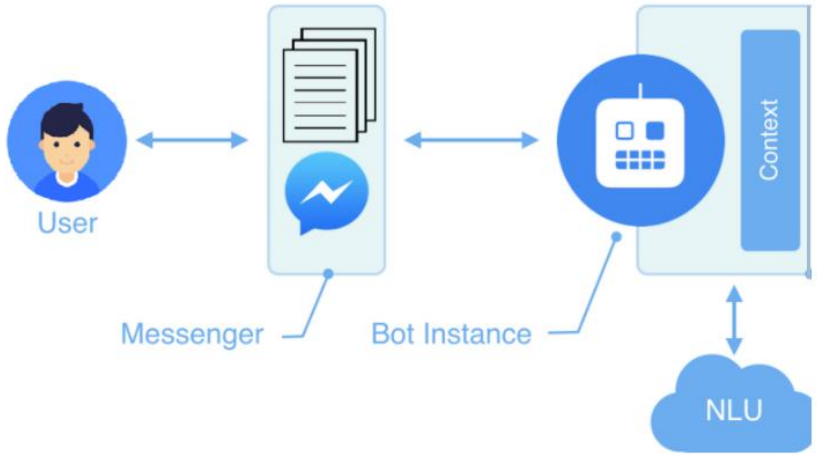
After the initial preprocessing phase, we need to transform the text into a meaningful vector/array of numbers. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

Model Solution Method:

In attempt to build a chatbot from scratch I have focused on building a "**Retrieval Based**" chatbot model. After successful completion of this simple chatbot, will work on more advanced ones.

Overall diagram of a retrieval based chatbot.



Followed below steps to build a chatbot:

- Libraries & Data
- Initializing Chatbot Training
- Building the Deep Learning Model
- Building Chatbot GUI
- Running Chatbot

Libraries & Data

Started by building individual components and then integrating them for a working chatbot. Here's a quick breakdown of the components:

- train_chatbot.py — The code for reading in the natural language data into a training set and using a Keras sequential neural network to create a model.
- chatgui.py — The code for cleaning up the responses based on the predictions from the model and creating a graphical interface for interacting with the chatbot.
- classes.pkl — A list of different types of classes of responses.
- words.pkl — A list of different words that could be used for pattern recognition.
- intents.json — A bunch of JavaScript objects that lists different tags that correspond to different types of word patterns.
- chatbot_model.h5 — The actual model created by train_chatbot.py and used by chatgui.py

Now let us begin by importing the necessary libraries by running the python files on the terminal. I use pip3 to install the packages.

```
In [13]:
import nltk
nltk.download('punkt')
nltk.download('wordnet')
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
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\kumar\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\kumar\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

We have libraries like nltk (Natural Language Toolkit), which contains a tools for cleaning up text and preparing it for deep learning algorithms, json, which loads json files directly into Python, pickle, which loads pickle files, numpy, which can perform linear algebra operations very efficiently, and keras, which is the deep learning framework we'll be using.

Initializing Chatbot Training

```
In [14]:
words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read()
intents = json.loads(data_file)
```

At first I did initialize all of the lists where we'll store our natural language data. I have the json file which contains the "intents". Here's a snippet of what the json file actually looks like.

```
{"intents": [
        {"tag": "greeting",
         "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],
         "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],
         "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": [""]
        },
```

We use the json module to load in the file and save it as the variable intents.

In [15]:
```
for intent in intents['intents']:
    for pattern in intent['patterns']:

        # take each word and tokenize it
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        # adding documents
        documents.append((w, intent['tag']))

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

If you look carefully at the json file, you can see that there are sub-objects within objects. For example, “patterns” is an attribute within “intents”. So, we will use a nested for loop to extract all of the words within “patterns” and add them to our words list. Then I added to our documents list each pair of patterns within their corresponding tag. Also added the tags into our classes list, and we use a simple conditional statement to prevent repeats.

In [16]:
```
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

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

print (len(documents), "documents")

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

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


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

```
47 documents
9 classes ['adverse_drug', 'blood_pressure', 'blood_pressure_search', 'goodbye', 'greeting', 'hospital_search', 'options',
'pharmacy_search', 'thanks']
88 unique lemmatized words ["'s", ',', 'a', 'adverse', 'all', 'anyone', 'are', 'awesome', 'be', 'behavior', 'blood', 'by',
'bye', 'can', 'causing', 'chatting', 'check', 'could', 'data', 'day', 'detail', 'do', 'dont', 'drug', 'entry', 'find', 'fo
r', 'give', 'good', 'goodbye', 'have', 'hello', 'help', 'helpful', 'helping', 'hey', 'hi', 'history', 'hola', 'hospital', 'h
ow', 'i', 'id', 'is', 'later', 'list', 'load', 'locate', 'log', 'looking', 'lookup', 'management', 'me', 'module', 'nearby',
'next', 'nice', 'of', 'offered', 'open', 'patient', 'pharmacy', 'pressure', 'provide', 'reaction', 'related', 'result', 'sea
rch', 'searching', 'see', 'show', 'suitable', 'support', 'task', 'thank', 'thanks', 'that', 'there', 'till', 'time', 'to',
'transfer', 'up', 'want', 'what', 'which', 'with', 'you']
```

Next, we will take the words list and lemmatize and lowercase all the words inside. The purpose of lemmatizing our words is to narrow everything down to the simplest level it can be. It will save us a lot of time and unnecessary error when we actually process these words for machine learning. This is also similar to stemming, which is to reduce an inflected word down to its base or root form.

Next, we sort our lists and print out the results.

Building the Deep Learning Model

In [17]:
```
# initializing training data
training = []
output_empty = [0] * len(classes)
for doc in documents:
    # initializing 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
```

```
<ipython-input-17-1a891a7a8859>:22: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a
list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you m
ust specify 'dtype=object' when creating the ndarray
  training = np.array(training)
```

Initializing the training data with a variable training. We are creating a giant nested list which contains bags of words for each of our documents. We have a feature called output_row which simply acts as a key for the list. We then shuffle our training set and do a train-test-split, with the patterns being the X variable and the intents being the Y variable.

In [18]:
```
# 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")
```

```
Epoch 1/200
10/10 [==============================] - 0s 846us/step - loss: 2.2631 - accuracy: 0.0314
Epoch 2/200
10/10 [==============================] - 0s 778us/step - loss: 2.1226 - accuracy: 0.2412
Epoch 3/200
10/10 [==============================] - 0s 625us/step - loss: 2.0212 - accuracy: 0.2136
Epoch 4/200
10/10 [==============================] - 0s 667us/step - loss: 1.9699 - accuracy: 0.3710
Epoch 5/200
10/10 [==============================] - 0s 784us/step - loss: 1.8018 - accuracy: 0.3770
Epoch 6/200
10/10 [==============================] - 0s 667us/step - loss: 1.7077 - accuracy: 0.5977
Epoch 7/200
10/10 [==============================] - 0s 1ms/step - loss: 1.7117 - accuracy: 0.5716
Epoch 8/200
10/10 [==============================] - 0s 535us/step - loss: 1.4272 - accuracy: 0.5772
Epoch 9/200
10/10 [==============================] - 0s 777us/step - loss: 1.5030 - accuracy: 0.4951
```

Now that we have our training and test data ready, we will now use a deep learning model from keras called Sequential.

The Sequential model in keras is the simplest neural networks, a multi-layer perceptron.

This network has 3 layers, with the first one having 128 neurons, the second one having 64 neurons, and the third one having the number of intents as the number of neurons. Intention of this network is to be able to predict which intent to choose given some data.

The model will be trained with stochastic gradient descent. Stochastic gradient descent is more efficient than normal gradient descent, that's all you need to know.

After the model is trained, the whole thing is turned into a numpy array and saved as chatbot_model.h5. We will use this model to form our chatbot interface.

Building Chatbot GUI

In [19]:
```
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'))
```

Once again, we need to extract the information from our files.

In [20]:
```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    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
```

```
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(msg):
    ints = predict_class(msg, model)
    res = getResponse(ints, intents)
    return res
```

Below functions contains all of the necessary processes for running the GUI and encapsulates them into units. We have the clean_up_sentence() function which cleans up any sentences that are inputted. This function is used in the bow() function, which takes the sentences that are cleaned up and creates a bag of words that are used for predicting classes (which are based off the results we got from training our model earlier).

In our predict_class() function, we use an error threshold of 0.25 to avoid too much overfitting. This function will output a list of intents and the probabilities, their likelihood of matching the correct intent. The function getResponse() takes the list outputted and checks the json file and outputs the most response with the highest probability.

Finally our chatbot_response() takes in a message (which will be inputted through our chatbot GUI), predicts the class with our predict_class() function, puts the output list into getResponse(), then outputs the response. What we get is the foundation of our chatbot. We can now tell the bot something, and it will then respond back.

In [23]:
```
#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

#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()
```

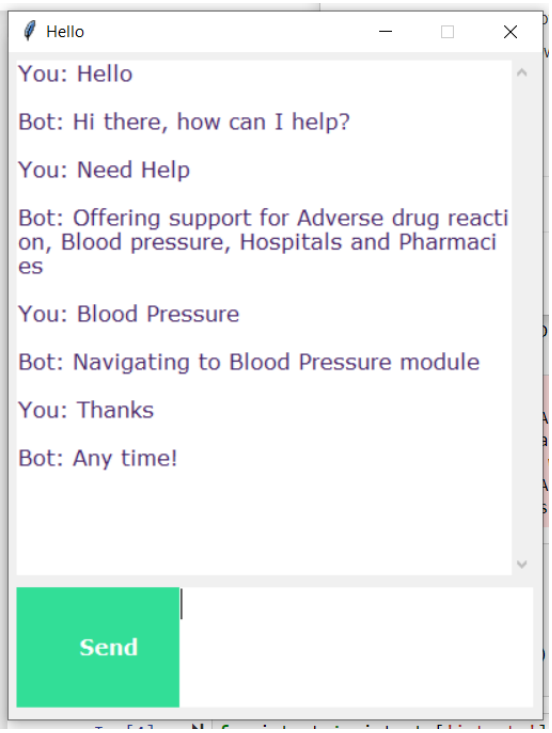
I have used tkinter to create the GUI. Tinker is a Python library that allows us to create custom interfaces.

We create a function called send() which sets up the basic functionality of our chatbot. If the message that we input into the chatbot is not an empty string, the bot will output a response based on our chatbot_response() function.

After this, we build our chat window, our scrollbar, our button for sending messages, and our textbox to create our message. We place all the components on our screen with simple coordinates and heights.

Running Chatbot

After successful execution of the program, we will get an chat window popped up which will respond against our inquiries.



Summary:

Building a simple chatbot exposed me to a variety of useful skills for data science and general programming. I feel that the this has helped me to understand applications of NLP and how chatbots can be implemented for variety of interactive user applications. I will further deep drive into this area and will advance the usage and application of the chatbot application.

References:

1. https://marutitech.com/chatbots-work-guide-chatbot-architecture/
2. https://bigdata-madesimple.com/how-do-chatbots-work-an-overview-of-the-architecture-of-a-chatbot/#:~:text=A%20chatbot%20is%20programmed%20to,scripts%20and%20machine%20learning%20applications.
3. https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1
4. https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32#:~:text=Natural%20Language%20Processing%2C%20usually%20shortened,a%20manner%20that%20is%20valuable.
5. https://medium.com/analytics-vidhya/building-a-simple-chatbot-in-python-using-nltk-7c8c8215ac6e
6. https://towardsdatascience.com/how-to-create-a-chatbot-with-python-deep-learning-in-less-than-an-hour-56a063bdfc44
7. https://www.datacamp.com/community/tutorials/building-a-chatbot-using-chatterbot