Build Your First Python Chatbot Project

In this article, see a tutorial on how to build a Python chatbot project.

by Shivashish Thkaur </>CORE · May. 04, 20 · AI Zone · Tutorial

Introduction

Chatbots are extremely helpful for business organizations and also the customers. The majority of people prefer to talk directly from a chatbox instead of calling service centers. Facebook released data that proved the value of bots. More than 2 billion messages are sent between people and companies monthly. The HubSpot research tells us that 71% of people want to get customer support from messaging apps. It is a quick way to get their problems solved so chatbots have a bright future in organizations.

Today we are going to build an exciting project on Chatbot. We will implement a chatbot from scratch that will be able to understand what the user is talking about and give an appropriate response.

Prerequisites

To implement the chatbot, we will be using Keras, which is a Deep Learning library, NLTK, which is a Natural Language Processing toolkit, and some helpful libraries. Run the below command to make sure all the libraries are installed:

```
pip install tensorflow keras pickle nltk
```

If you want to learn Python for free, then here is *the Master guide to learn Python for free*.

You may also like: Build It Yourself: Chatbot API With Keras/TensorFlow Model

How do Chatbots Work?

Chatbots are nothing but an intelligent piece of software that can interact and communicate with people just like humans. Interesting, isn't it? So now let's see how they actually work.

All chatbots come under the NLP (Natural Language Processing) concepts. NLP is composed of two things:

- **NLU** (Natural Language Understanding): The ability of machines to understand human language like English.
- **NLG** (Natural Language Generation): The ability of a machine to generate text similar to human written sentences.

Imagine a user asking a question to a chatbot: "Hey, what's on the news today?"

The chatbot will break down the user sentence into two things: intent and an entity. The intent for this sentence could be get_news as it refers to an action the user wants to perform. The entity tells specific details about the intent, so "today" will be the entity. So this way, a machine learning model is used to recognize the intents and entities of the chat.

Project File Structure

Project File Structure

After the project is complete, you will be left with all these files. Lets quickly go through each of them. It will give you an idea of how the project will be implemented.

- **Train_chatbot.py —** In this file, we will build and train the deep learning model that can classify and identify what the user is asking to the bot.
- **Gui_Chatbot.py —** This file is where we will build a graphical user interface to chat with our trained chatbot.
- **Intents.json —** The intents file has all the data that we will use to train the model. It contains a collection of tags with their corresponding patterns and responses.
- **Chatbot_model.h5 —** This is a hierarchical data format file in which we have stored the weights and the architecture of our trained model.
- **Classes.pkl —** The pickle file can be used to store all the tag names to classify when we are predicting the message.
- **Words.pkl —** The words.pkl pickle file contains all the unique words that are the vocabulary of our model.

Download the source code and the dataset: https://drive.google.com/drive/folders/1r6MrrdE8V0bWBxndGfJxJ4Om62dJ2OMP?usp=sharing

How to Build Your Own Chatbot

I've simplified the building of this chatbot in 5 steps:

Step 1. Import Libraries and Load the Data

Create a new python file and name it as train_chatbot and then we are going to import all the required modules. After that, we will read the JSON data file in our Python program.

Python

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

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

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

Step 2. Preprocessing the Data

The model cannot take the raw data. It has to go through a lot of pre-processing for the machine to easily understand. For textual data, there are many preprocessing techniques available. The first technique is tokenizing, in which we break the sentences into words.

By observing the intents file, we can see that each tag contains a list of patterns and responses. We tokenize each pattern and add the words in a list. Also, we create a list of classes and documents to add all the intents associated with patterns.

Python

```
words=[]
classes = []
documents = []
ignore_letters = ['!', '?', ',', '.']

for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        word = nltk.word_tokenize(pattern)
        words.extend(word)
        #add documents in the corpus
        documents.append((word, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

print(documents)
```

Another technique is Lemmatization. We can convert words into the lemma form so that we can reduce all the canonical words. For example, the words play, playing, plays, played, etc. will all be replaced with play. This way, we can reduce the number of total words in our vocabulary. So now we lemmatize each word and remove the duplicate words.

Python

```
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_letters]
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'))
```

In the end, the words contain the vocabulary of our project and classes contain the total entities to classify. To save the python object in a file, we used the pickle.dump() method. These files will be helpful after the training is done and we predict the chats.

Step 3. Create Training and Testing Data

To train the model, we will convert each input pattern into numbers. First, we will lemmatize each word of the pattern and create a list of zeroes of the same length as the total number of words. We will set value 1 to only those indexes that contain the word in the patterns. In the same way, we will create the output by setting 1 to the class input the pattern belongs to.

Python

```
# create the training data
training = []
# create empty array for the output
output_empty = [0] * len(classes)
# training set, bag of words for every sentence
for doc in documents:
    # initializing bag of words
    bag = []
    # list of tokenized words for the pattern
    word_patterns = doc[0]
    # lemmatize each word - create base word, in attempt to represent related words
```

```
    word_patterns = [lemmatizer.lemmatize(word.lower()) for word in word_patterns]
    # create the bag of words array with 1, if word is found in current pattern
    for word in words:
        bag.append(1) if word in word_patterns 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 the features and make numpy array
random.shuffle(training)
training = np.array(training)
# create training and testing lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data is created")
```

Step 4. Training the Model

The architecture of our model will be a neural network consisting of 3 dense layers. The first layer has 128 neurons, the second one has 64 and the last layer will have the same neurons as the number of classes. The dropout layers are introduced to reduce overfitting of the model. We have used the SGD optimizer and fit the data to start the training of the model. After the training of 200 epochs is completed, we then save the trained model using the Keras model.save("chatbot_model.h5") function.

Python

```
# deep neural networds model
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'))

# Compiling model. SGD 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'])

#Training 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 is created")
```

Step 5. Interacting With the Chatbot

Our model is ready to chat, so now let's create a nice graphical user interface for our chatbot in a new file. You can name the file as gui_chatbot.py

In our GUI file, we will be using the Tkinter module to build the structure of the desktop application and then we will capture the user message and again perform some preprocessing before we input the message into our trained model.

The model will then predict the tag of the user's message, and we will randomly select the response from the list of responses in our intents file.

Here's the full source code for the GUI file.

Python

```
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'))

def clean_up_sentence(sentence):
    # tokenize the pattern - splitting words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stemming every word - reducing to base form
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words
# return bag of words array: 0 or 1 for words that exist in sentence

def bag_of_words(sentence, words, show_details=True):
    # tokenizing patterns
    sentence_words = clean_up_sentence(sentence)
    # bag of words - vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,word in enumerate(words):
            if word == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % word)
    return(np.array(bag))

def predict_class(sentence):
    # filter below  threshold predictions
    p = bag_of_words(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]
    # sorting strength 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

#Creating tkinter GUI
import tkinter
from tkinter import *

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

    if msg != '':
        ChatBox.config(state=NORMAL)
        ChatBox.insert(END, "You: " + msg + '\n\n')
        ChatBox.config(foreground="#446665", font=("Verdana", 12 ))

        ints = predict_class(msg)
```

```
        ints = predict_class(msg)
        res = getResponse(ints, intents)

        ChatBox.insert(END, "Bot: " + res + '\n\n')

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

root = Tk()
root.title("Chatbot")
root.geometry("400x500")
root.resizable(width=FALSE, height=FALSE)

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

ChatBox.config(state=DISABLED)

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

#Create Button to send message
SendButton = Button(root, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#f9a602", activebackground="#3c9d9b",fg='#000000',
                    command= send )

#Create the box to enter message
EntryBox = Text(root, 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)
ChatBox.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)

root.mainloop()
```

Explore more *Python Projects with Source Code*.

Running the Chatbot

Now we have two separate files, one is the train_chatbot.py, which we will use first to train the model.

```
python train_chatbot.py
```

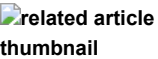
In the end, I want to say at this time of coronavirus please be safe and follow these **12 healthy lifestyle tips**.

Further Reading

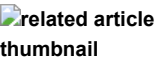
4 Business Applications for Natural Language Processing

Building a Chatbot With an Expert System

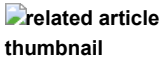
Like This Article? Read More From DZone



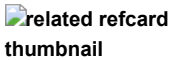
DZone Article
15 Most Used Machine Learning Tools By Experts



DZone Article
How Do You Measure If Your Customer Churn Predictive Model Is Good?



DZone Article
Predicting Wine Quality With Several Classification Techniques



Free DZone Refcard
Introduction to TensorFlow

Topics: ARTIFICAL INTELLIGENCE, CHATBOT DEVELOPMENT, DATA SCIENCE, LEARNING PYTHON, MACHINE LEARNING, PYTHON PROGRAMMING, PYTHON PROJECTS

Opinions expressed by DZone contributors are their own.