4222-SURYA GROUP OF INSTITUTION

VIKIRAVANDI-605 652.

NAAN MUDHALVAN PROJECT

CREATE CHATBOT IN PYTHON

PHASE 4: Development Part 2

Presented by:

R.VINISHA

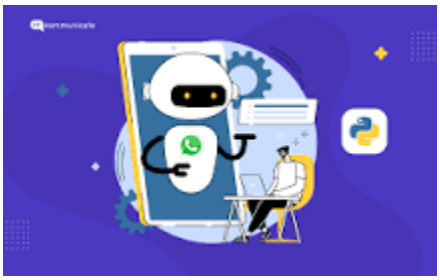
REG NO: 422221106018

ECE DEPARTMENT

Flask_Phase 4: Development part 2

What is flask chatbot

Chatbots are software tools created to interact with humans through chat. The first chatbots could create simple conversations based on a complex system of rules. You can build intelligent chatbots for WhatsApp using the Python Framework Flask and the Kompose Bot builder.



The Flask is a Python micro-framework used to create small web applications and websites using Python. Flask works on a popular templating engine called Jinja2, a web templating system combined with data sources to the dynamic web pages.

How do I create a chatbot and integrate with my website

How to Build a Chatbot for Your Website (Step-by-Step)

1. Decide what type of chatbot is best for your business. ...
2. Determine your chatbot KPIs. ...
3. Understand chatbot user needs. ...

4. Give your chatbot a personality. ...
5. Create your chatbot conversation flow. ...
6. Design your bot. ...
7. Preview and test. ...
8. Target your chatbots.

How to create chatbot using Flask Python

- Step 1: Import necessary methods of Flask and ChatterBot.
- Step 2: Then, we will initialize the Flask app by adding the below code. Flask(name) is used to create the Flask class object so that Python code can initialize the Flask server.
- Step 3: Now, we will give the name to our chatbot.

What are the features of Flask app

Advantages of using Flask:

- There is a built-in development server and a fast debugger provided.
- Lightweight.
- Support Secure cookies.
- Templating using Jinja2.
- Request dispatching using REST.
- Support for unit testing is built-in.

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 ‘model.h5’.

```
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('data.json').read()

intents = json.loads(data_file)
```

```
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'])
# lemmaztize 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)

pickle.dump(words,open('texts.pkl','wb'))

pickle.dump(classes,open('labels.pkl','wb'))

# 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")

# 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('model.h5', hist)

print("model created")
```

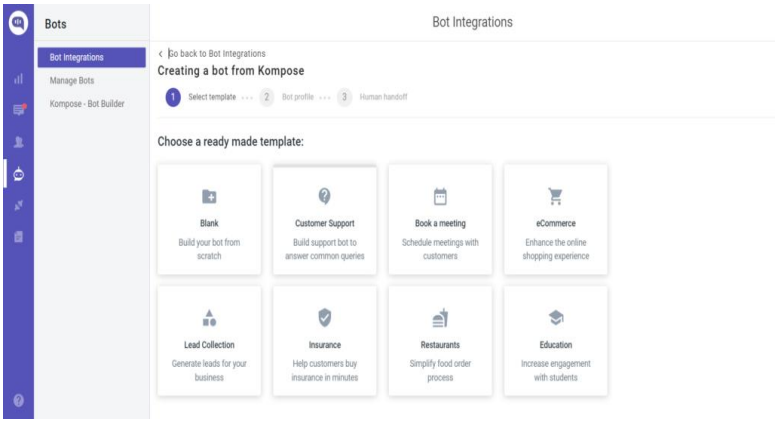
Fwwdback of chatbot using flask

We will make a Flask chatbot. Flask is a microframework used for web development. We will follow the process given below:

1. Make a web app using the flask.
2. Make a directory for the templates.
3. Train the bot.
4. Make conversation with the bot.

Flask is a Python web framework that is widely used for building web applications. It is a lightweight framework that is easy to learn and has a simple

syntax. In this tutorial, you will learn how to use Flask to create a back-end that communicates with a third-party API and renders the response in dynamic HTML using Jinja.



- **Step 1**: Installing Python 3.7 with pip3.7 along with Python3.10

This step is optional if you already have Python Version *<=3.8 or >=3.4*. Please check your python using this command – python -V or python3 -V

```
# Install Python 3.7 Version

sudo apt update

sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libsqlite3-dev libreadline-dev libffi-dev wget libbz2-dev

sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt install python3.7

python3.7 --version

# Install Pip

python3.7 -m pip install pip
```

```
# if you are getting distutils error

sudo apt-get install python3.7-distutilsCopy Code
```

```
from flask import Flask, render_template, request, jsonify
import os,sys,requests, json
from random import randintapp = Flask(__name__)@app.route('/')
def home():
  return render_template('index.html')@app.route('/parse',methods=['POST', 'GET'] )
def extract():
  text=str(request.form.get('value1'))
  payload = json.dumps({"sender": "Rasa","message": text})
  headers = {'Content-type': 'application/json', 'Accept':     'text/plain'}
  response = requests.request("POST",   url="http://localhost:5005/webhooks/rest/webhook",
headers=headers, data=payload)
  response=response.json()
  resp=[]
  for i in range(len(response)):
    try:
      resp.append(response[i]['text'])
    except:
      continue
  result=resp
  return render_template('index.html', result=result,text=text)if __name__ == "__main__":
  app.run(debug=True)
```

- **Step 2:** Create and Install Virtualenv for Python Version <=3.8 or >=3.4

As we will be using virtual environment for this project. Please read our blog on Using Virtual Environments for Python Projects, if you are not familar with virtualenv in Python.

Let’s install virtualenv using Python3.7

> pip install virtualenvCopy Code

Create virtualenv named venv and activate it.

> python3.7 -m virtualenv venv
>
> source venv/bin/activateCopy Code

Please Note: When you have virtualenv activated, you will see python version 3.7. Outside virtualenv it will be different.

```
#imports
from flask import Flask, render_template, request
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer

app = Flask(__name__)
#create chatbot
englishBot = ChatBot("Chatterbot", storage_adapter="chatterbot.storage.SQLStorageAdapter")
trainer = ChatterBotCorpusTrainer(englishBot)
trainer.train("chatterbot.corpus.english") #train the chatter bot for english

#define app routes
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/get")
#function for the bot response
def get_bot_response():
    userText = request.args.get('msg')
    return str(englishBot.get_response(userText))

if __name__ == "__main__":
    app.run()
```

- **Step 3**: Installing Required Libraries

```
pip install Flask

pip install ChatterBot==1.0.8

pip install chatterbot-corpus==1.2.0

pip install spacy==2.1.9

pip install nltk==3.8.1Copy Code
```

- **Step 4**: Initializing Flask App

Create a new Python file (app.py) and add the following code:

```
from chatbot import chatbot

from flask import Flask, render_template, request

app = Flask(__name__)

app.static_folder = 'static'



@app.route("/")

def home():

    return render_template("index.html")

@app.route("/get")
```

```
def get_bot_response():

    userText = request.args.get('msg')

    return str(chatbot.get_response(userText))

if __name__ == "__main__":

    app.run()Copy Code
```

```
# chat initialization
model = load_model("chatbot_model.h5")
intents = json.loads(open("intents.json").read())
words = pickle.load(open("words.pkl", "rb"))
classes = pickle.load(open("classes.pkl", "rb"))

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("index.html")


@app.route("/get", methods=["POST"])
def chatbot_response():
    msg = request.form["msg"]
    #checks is a user has given a name, in order to give a personalized feedback
    if msg.startswith('my name is'):
        name = msg[11:]
        ints = predict_class(msg, model)
        res1 = getResponse(ints, intents)
        res =res1.replace("{n}",name)
    elif msg.startswith('hi my name is'):
        name = msg[14:]
        ints = predict_class(msg, model)
        res1 = getResponse(ints, intents)
        res =res1.replace("{n}",name)
    #if no name is passed execute normally
```

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


# chat functionalities
def clean_up_sentence(s
```

- **Step 5**: Interact with Chatterbot Machine Learning Model

Create a new Python file (chatbot.py) and and add the following code:

```
from chatterbot import ChatBot

import spacy

spacy.cli.download("en_core_web_sm")

spacy.cli.download("en")



nlp = spacy.load('en_core_web_sm')

chatbot = ChatBot(

    'CoronaBot',

    storage_adapter='chatterbot.storage.SQLStorageAdapter',

    logic_adapters=[

        'chatterbot.logic.MathematicalEvaluation',
```

```
        'chatterbot.logic.TimeLogicAdapter',
        'chatterbot.logic.BestMatch',
        {
            'import_path': 'chatterbot.logic.BestMatch',
            'default_response': 'I am sorry, but I do not understand. I am still learning.',
            'maximum_similarity_threshold': 0.90
        }
    ],
    database_uri='sqlite:///database.sqlite3'
)
# Training With Own Questions
from chatterbot.trainers import ListTrainer
trainer = ListTrainer(chatbot)
training_data_quesans = open('training_data/ques_ans.txt').read().splitlines()
training_data_personal = open('training_data/personal_ques.txt').read().splitlines()
training_data = training_data_quesans + training_data_personal
trainer.train(training_data)
```

```
# Training With Corpus

from chatterbot.trainers import ChatterBotCorpusTrainer

trainer_corpus = ChatterBotCorpusTrainer(chatbot)

trainer_corpus.train(

   'chatterbot.corpus.english'

)Copy Code
```

In above example, we are loading data from training_data/ques_ans.txt and training_data/personal_ques.txt. You can download the files from here.

- **Step 6:** Creating Chatbot UI

    - HTML Template (index.html) – Create a new HTML file index.html inside the “templates” folder. Please copy the html content from here.
    - CSS File (style.css) – Create a new CSS file inside the “static” folder to style the chat interface. Please copy the css content from here.

Bellow is the folder structure your project should look like:

```
chatbot_project/

├── app.py

├── chatbot.py

├── venv
```

```
├── database.sqlite3.py
├── templates/
│   └── index.html
├── training_data/
│   ├── ques_ans.txt
│   └──personal_ques.txt
└── static/
    └── style.cssCopy Code
```

- **Step 7**: Run the Flask App

Open your terminal, navigate to the chatbot_project directory, and execute the following command:

```
python3.7 app.pyCopy Code
```

Step 8: Interact with the Chatbot

Open your web browser and go to http://localhost:5000 to access the chat interface. Now, you can type messages in the input field, press “Send,” and watch your chatbot respond!

Note – If you are not getting right answer for questions, delete the .sqlite3 database file from your folder and re-run the flask app.

```
from flask import Flask, render_template, request, jsonify
import openai
app = Flask(__name__)
# OpenAI API Key
openai.api_key = 'YOUR_API_KEY'
def get_completion(prompt):
        print(prompt)
        query = openai.Completion.create(
                engine="text-davinci-003",
                prompt=prompt,
                max_tokens=1024,
                n=1,
                stop=None,
                temperature=0.5,
        )
        response = query.choices[0].text
        return response
@app.route("/", methods=['POST', 'GET'])
```

```
def query_view():
        if request.method == 'POST':
                print('step1')
                prompt = request.form['prompt']
                response = get_completion(prompt)
                print(response)
                return jsonify({'response': response})
        return render_template('index.html')
if __name__ == "__main__":
        app.run(debug=True)
```

Conclusion

A chatbot is one of the simple ways to transport data from a computer without having to think for proper keywords to look up in a search or browse several web pages to collect information; users can easily type their query in natural language and retrieve information.