CARE COLLEGE OF ENGINEERING

Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai
Accredited by NAAC with 'A' Grade
#27, Thayanur, Tiruchirappalli - 620009

Cloud Application Development - Group 5

Project Title: Chatbot using Watson assistant

Table of contents

Fields of NLP

Below are the core fields of NLP:

1. **Natural language generation(NLG)**: NLG is a specialty in artificial intelligence. It is a software technology that automatically transforms data into simple English.
2. **Natural language understanding(NLU)**: Is a branch of natural language processing (NLP) that helps computers understand and interpret human language by breaking speech into its constituent parts. However, NLU goes further than speech recognition to understand what the user is trying to communicate with their words.
3. **Natural language interaction(NLI)**: To engage with any connected device or service in a human-like manner, NLI brings together a varied collection of natural language principles.

Types of bots

- **Scraper Bots:** They are designed to read and save data from the Internet.
- **Spam Bots:** Designed to gather email addresses from spam mailing lists.
- **Spider Bots:** Searches the web and downloads pages before indexing the material on each page it comes across.
- **Transnational Bots:** Designed to be used in transactions.
- **Social Media Bot:** Automatically responds to questions on social media platforms.

Building an NLP chatbot

Follow the steps below to build a conversational interface for our chatbot successfully.

Step one: Importing libraries

Imports are critical for successfully organizing your Python code. Correctly importing code will increase your productivity by allowing you to reuse code while also maintaining the maintainability of your projects.

The necessary libraries include:

- JSON: It is possible to utilize it to work with JSON data.
- String: Provides access to several potentially valuable constants.
- Random: For various distributions, this module implements pseudo-random number generators.
- WordNetLemmatizer: It can lemmatize.
- Tensorflow: A multidimensional array of elements is represented by this symbol.
- Sequential: Sequential groups a linear stack of layers into a tf.keras.Model.

The code below shows how we import the libraries:

```
import json
import string
import random
import nltk
import numpy as num
from nltk.stem import WordNetLemmatizer # It has the ability to lemmatize.
import tensorflow as tensorF # A multidimensional array of elements is represented
by this symbol.
from tensorflow.keras import Sequential # Sequential groups a linear stack of layers
into a tf.keras.Model
from tensorflow.keras.layers import Dense, Dropout

nltk.download("punkt")# required package for tokenization
nltk.download("wordnet")# word database
```

Step two: Creating a JSON file

This step will create an intents JSON file that lists all the possible outcomes of user interactions with our chatbot. We first need a set of tags that users can use to categorize their queries. These tags include name, age, and many others. Every new tag would require a unique pattern.

Identifying these trends can help the chatbot train itself on how people query about our chatbot’s name, allowing it to be more responsive. The chatbot will return pre-programmed responses to answer questions:

```
ourData = {"ourIntents": [

          {"tag": "age",
           "patterns": ["how old are you?"],
           "responses": ["I am 2 years old and my birthday was yesterday"]
          },
           {"tag": "greeting",
           "patterns": [ "Hi", "Hello", "Hey"],
           "responses": ["Hi there", "Hello", "Hi :)"],
          },
           {"tag": "goodbye",
           "patterns": [ "bye", "later"],
           "responses": ["Bye", "take care"]
          },
          {"tag": "name",
           "patterns": ["what's your name?", "who are you?"],
           "responses": ["I have no name yet," "You can give me one, and I will
appreciate it"]
          }

]}
```

Step three: Processing data

In this section, vocabulary of all the terms used in the patterns, list of tag classes, list of all the patterns in the intents file, and all the related tags for each pattern will be created before creating our training data:

```
lm = WordNetLemmatizer() #for getting words
# lists
ourClasses = []
newWords = []
documentX = []
documentY = []
# Each intent is tokenized into words and the patterns and their associated tags are
added to their respective lists.
for intent in data["intents"]:
    for pattern in intent["patterns"]:
        ournewTkns = nltk.word_tokenize(pattern)# tokenize the patterns
        newWords.extend(ournewTkns)# extends the tokens
        documentX.append(pattern)
        documentY.append(intent["tag"])


    if intent["tag"] not in ourClasses:# add unexisting tags to their respective classes
        ourClasses.append(intent["tag"])

newWords = [lm.lemmatize(word.lower()) for word in newWords if word not in
string.punctuation] # set words to lowercase if not in punctuation
newWords = sorted(set(newWords))# sorting words
ourClasses = sorted(set(ourClasses))# sorting classes
```

Step four: Designing a neural network model

Because neural networks can only understand numerical values, we must first process our data so that a neural network can understand what we are doing.

The code below is used to turn our data into numerical values using bag of words (BoW) encoding system:

```
trainingData = [] # training list array
outEmpty = [0] * len(ourClasses)
# bow model
for idx, doc in enumerate(documentX):
   bagOfwords = []
   text = lm.lemmatize(doc.lower())
   for word in newWords:
      bagOfwords.append(1) if word in text else bagOfwords.append(0)

   outputRow = list(outEmpty)
   outputRow[ourClasses.index(documentY[idx])] = 1
   trainingData.append([bagOfwords, outputRow])

random.shuffle(trainingData)
trainingData = num.array(trainingData, dtype=object)# coverting our data into an
array afterv shuffling

x = num.array(list(trainingData[:, 0]))# first trainig phase
y = num.array(list(trainingData[:, 1]))# second training phase
```

After converting our data to a numerical representation, we can now design a neural network model which we will feed our training data. The model will select an appropriate response from the tag associated with a given feature:

```
iShape = (len(x[0]),)
oShape = len(y[0])
# parameter definition
ourNewModel = Sequential()
# In the case of a simple stack of layers, a Sequential model is appropriate

# Dense function adds an output layer
ourNewModel.add(Dense(128, input_shape=iShape, activation="relu"))
# The activation function in a neural network is in charge of converting the node's
```

```
summed weighted input into activation of the node or output for the input in
question
ourNewModel.add(Dropout(0.5))
# Dropout is used to enhance visual perception of input neurons
ourNewModel.add(Dense(64, activation="relu"))
ourNewModel.add(Dropout(0.3))
ourNewModel.add(Dense(oShape, activation = "softmax"))
# below is a callable that returns the value to be used with no arguments
md = tensorF.keras.optimizers.Adam(learning_rate=0.01, decay=1e-6)
# Below line improves the numerical stability and pushes the computation of the
probability distribution into the categorical crossentropy loss function.
ourNewModel.compile(loss='categorical_crossentropy',
            optimizer=md,
            metrics=["accuracy"])
# Output the model in summary
print(ourNewModel.summary())
# Whilst training your Nural Network, you have the option of making the output
verbose or simple.
ourNewModel.fit(x, y, epochs=200, verbose=1)
# By epochs, we mean the number of times you repeat a training set.
```

Step five: Building useful features

In order to make use of our model in a chatbot, we must first implement the necessary functionality, which will be made easier by building a library of utility functions will help:

```
def ourText(text):
  newtkns = nltk.word_tokenize(text)
  newtkns = [lm.lemmatize(word) for word in newtkns]
  return newtkns

def wordBag(text, vocab):
  newtkns = ourText(text)
  bagOwords = [0] * len(vocab)
  for w in newtkns:
    for idx, word in enumerate(vocab):
      if word == w:
        bagOwords[idx] = 1
  return num.array(bagOwords)

def Pclass(text, vocab, labels):
  bagOwords = wordBag(text, vocab)
  ourResult = ourNewModel.predict(num.array([bagOwords]))[0]
  newThresh = 0.2
  yp = [[idx, res] for idx, res in enumerate(ourResult) if res > newThresh]

  yp.sort(key=lambda x: x[1], reverse=True)
  newList = []
  for r in yp:
    newList.append(labels[r[0]])
  return newList

def getRes(firstlist, fJson):
  tag = firstlist[0]
  listOfIntents = fJson["intents"]
  for i in listOfIntents:
    if i["tag"] == tag:
      ourResult = random.choice(i["responses"])
      break
  return ourResult
```

The user will be able to enter a query in a while loop, which will then be cleaned. Next, we use our bag of words model to convert our text into numerical values and make a prediction about which tag in our intent the features most closely represent us:

```
while True:
   newMessage = input("")
   intents = Pclass(newMessage, newWords, ourClasses)
   ourResult = getRes(intents, ourData)
   print(ourResult)
```