**CHATBOT USING PYTHON**

**Phase 3 - Project Development Part 1**

**Objective:**

The main goal of this code is to build a conversational agent that can interpret user messages and identify the intent behind those messages. It's a fundamental component of many chatbots and virtual assistants, enabling them to respond appropriately to user queries.

The code is written in Python and is used to build a machine learning model for natural language processing (NLP) tasks, particularly intent recognition. Here's a breakdown of the environment, basic interactions, and installed libraries used in the code:

**Environment:**

1. **Programming Language:** The code is written in Python, a widely used language for machine learning and NLP tasks.

2. **Machine Learning Framework**: Keras is used to build and train a neural network model. Keras is typically run on top of a deep learning backend, such as TensorFlow or Theano.

**Basic Interactions:**

The code can be interacted with in the following ways:

1. **Data Preparation:** The code reads and processes data from a JSON file ('data.json') that contains intents and associated patterns.

2. **Training:** The model is trained using the processed data to learn the relationships between patterns and intents. Training includes specifying the model architecture, compiling the model, and fitting it to the training data.

3. **Saving:** The trained model is saved to a file named 'model.h5' for later use.

4. **Output:** The code may produce output during the training process, indicating the progress of the training, including loss and accuracy metrics.

**Installed Libraries:**

The code relies on several external libraries and modules, which need to be installed for it to run successfully. Here are the key libraries used:

1. **nltk (Natural Language Toolkit):** This library is used for natural language processing tasks. In the code, it's employed for tokenization and lemmatization of words.

2. **Keras:** Keras is a high-level deep learning library that sits on top of other deep learning backends (such as TensorFlow or Theano). It's used for defining and training neural network models in this code. Specifically, it imports modules like `Sequential`, `Dense`, `Activation`, `Dropout`, and `SGD`.

3. **numpy:** NumPy is used for numerical operations and data manipulation. It is commonly used in machine learning and scientific computing.

4. **json:** The code uses the `json` library to read and parse data from the 'data.json' file, which contains the intent and pattern information.

5. **pickle:** The `pickle` library is used to serialize (save) Python objects, specifically the 'words' and 'classes' lists are serialized and saved to 'texts.pkl' and 'labels.pkl', respectively.

--------------------------------------------------------------------------------------------------------------------------------------

**CODE:**

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.legacy 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)))

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:

if w in pattern\_words:

bag.append(1)

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,dtype=object)

# 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")

--------------------------------------------------------------------------------------------------------------------------------------

**TEAM MEMBERS:**

Reshmi M J

Suvetha R

Pavithra A

Mohammed Nawfal M S

Nidhish babu T L S