ReadMe_Developer

Strategy of Implementation

To implement this admission chatbot follow these steps as mentioned below:

1. Prepare the Dependencies:

It is best if you create and use a new Python virtual environment for the installation. To do so, you have to write and execute this command in your Python terminal:

```

$ (venv) pip install flask

$ (venv) pip install torch torchvision

$ (venv) pip install nltk flask-cors

```

2. Import Classes:

Importing classes is the second step in the Python chatbot creation process. All you need to do is import two classes. To do this, you can execute the following command:

Install nltk package

```

$ (venv) python

>>> import nltk

>>> nltk.download('punkt')

```

3. Create and Train the Chatbot:

This is the third step on creating a chatbot in python. The chatbot you are creating will be an instance of the class. Training ensures that the bot has enough knowledge to get started with specific responses to specific inputs.

To provide a list of responses, you can do it by specifying the lists of strings that can be later used to train your Python chatbot, and find the best match for each query.

Here are the 6 steps to create a admission chatbot:

1. Import and load the data file
2. Preprocess data
3. Create training and testing data
4. Build the model
5. Predict the response
6. Communicate with the Python Chatbot

1. Import and load the data file

First, make a file name as train_chatbot.py. We import the necessary packages for our chatbot and initialise the variables we will use in our Python project.

The data file is in JSON format so we used the json package to parse the JSON file into Python. (mentioned in chat.py on page no. 46)

```
import nltk

import random

import json

import torch

from model import NeuralNet

from nltk_utils import bag_of_words, tokenize

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

with open('intents.json', 'r') as json_data:
```

```
intents = json.load(json_data)
```

2. Preprocess data

When working with text data, we need to perform various preprocessing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to preprocess the data.

Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.

Here we iterate through the patterns and tokenize the sentence using nltk.word_tokenize() function and append each word in the words list. We also create a list of classes for our tags.(mentioned in train.py on page no.53)

```
for intent in intents['intents']:
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))
```

Now we will stem from each word. Stemming is a technique used to extract the base form of the words by removing affixes from them.

```
# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))
```

3. Create training and testing data

Now, we will create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn't understand text so we will convert text into numbers.(mentioned in train.py on page no. 53)

```
# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    # y: PyTorch CrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
    y_train.append(label)
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
# Hyper-parameters
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)
```

4. 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 'chatbot_model.h5'.

```
# Create modelimport torch
import torch.nn as nn
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()
```

5. Predict the response

To predict the sentences and get a response from the user to let us create a new file ‘chat.py’. (mentioned on page no.48)

The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

Again we import the necessary packages and load the ‘words.pkl’ and ‘classes.pkl’ pickle files which we have created when we trained our model:

```
import numpy as np

import nltk

# nltk.download('punkt')

from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()


def tokenize(sentence):

    """

    split sentence into array of words/tokens

    a token can be a word or punctuation character, or number

    """

    return nltk.word_tokenize(sentence)
```

To predict the class, we will need to provide input in the same way as we did while training.

So we will create some functions that will perform text preprocessing and then predict the class.

```
def get_response(msg):

    sentence = tokenize(msg)
```

```
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                return random.choice(intent['responses'])
    return "I do not understand..."

if __name__ == "__main__":
    print("Let's chat! (type 'quit' to exit)")
    while True:
        # sentence = "do you use credit cards?"
        sentence = input("You: ")
        if sentence == "quit":
            break
```

```
resp = get_response(sentence)

print(resp)
```

6. Communicate with the Python Chatbot:

To run the chatbot, we have two main files; train_chatbot.py and chatapp.py.

First, we train the model using the command in the terminal:

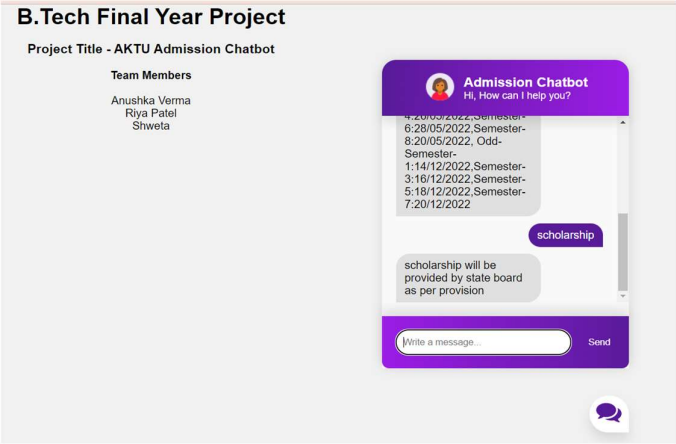
python app.py:

If we don't see any error during training, we have successfully created the model. Then to run the app,we run html file

base.html:

On the browser you will see an interface to contact with chatbot .

To interact with your Python chatbot by using responses. This is how it should look while communicating:



However, it is essential to understand that the chatbot using python might not know how to answer all your questions. Since its knowledge and training is still very limited, you have to give it time and provide more training data to train it further.