**CODE**

import json

import random

import torch

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from transformers import BertTokenizer, BertForSequenceClassification

from torch.utils.data import DataLoader, TensorDataset

from sklearn.metrics import accuracy\_score

# Load JSON data

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

data = json.load(file)

# Prepare data for training

patterns = []

tags = []

for intent in data['intents']:

for pattern in intent['patterns']:

patterns.append(pattern)

tags.append(intent['tag'])

# Convert to DataFrame

df = pd.DataFrame({"pattern": patterns, "tag": tags})

# Encode labels

label\_encoder = LabelEncoder()

df['tag\_encoded'] = label\_encoder.fit\_transform(df['tag'])

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

df['pattern'], df['tag\_encoded'], test\_size=0.2, random\_state=42

)

# Tokenize the data using BERT tokenizer

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

def tokenize\_data(texts):

return tokenizer(texts, padding=True, truncation=True, max\_length=32, return\_tensors="pt")

# Tokenize training and test data

train\_encodings = tokenize\_data(X\_train.tolist())

test\_encodings = tokenize\_data(X\_test.tolist())

# Convert labels to tensors

train\_labels = torch.tensor(y\_train.tolist())

test\_labels = torch.tensor(y\_test.tolist())

# Create DataLoader

train\_data = TensorDataset(train\_encodings['input\_ids'], train\_encodings['attention\_mask'], train\_labels)

test\_data = TensorDataset(test\_encodings['input\_ids'], test\_encodings['attention\_mask'], test\_labels)

train\_loader = DataLoader(train\_data, batch\_size=16, shuffle=True)

test\_loader = DataLoader(test\_data, batch\_size=16)

# Define the BERT model for classification

class BertModel(torch.nn.Module):

def \_\_init\_\_(self, model\_name='bert-base-uncased', num\_labels=len(label\_encoder.classes\_)):

super(BertModel, self).\_\_init\_\_()

self.bert = BertForSequenceClassification.from\_pretrained(model\_name, num\_labels=num\_labels)

def forward(self, input\_ids, attention\_mask):

outputs = self.bert(input\_ids=input\_ids, attention\_mask=attention\_mask)

return outputs.logits

# Initialize the model

model = BertModel()

# Set up the optimizer and loss function

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)

# Train the model

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)

def train\_model():

model.train()

for batch in train\_loader:

input\_ids, attention\_mask, labels = [item.to(device) for item in batch]

optimizer.zero\_grad()

logits = model(input\_ids, attention\_mask)

loss = torch.nn.CrossEntropyLoss()(logits, labels)

loss.backward()

optimizer.step()

def evaluate\_model():

model.eval()

all\_preds = []

all\_labels = []

with torch.no\_grad():

for batch in test\_loader:

input\_ids, attention\_mask, labels = [item.to(device) for item in batch]

logits = model(input\_ids, attention\_mask)

preds = torch.argmax(logits, dim=-1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

accuracy = accuracy\_score(all\_labels, all\_preds)

print(f"Accuracy: {accuracy \* 100:.2f}%")

# Train for a few epochs and evaluate every 10 epochs

for epoch in range(100):

train\_model()

if (epoch + 1) % 10 == 0: # Print every 10 epochs

print(f"Epoch {epoch + 1}")

evaluate\_model()

# Define function to get response from the bot

def get\_response(user\_input):

# Tokenize user input

user\_input\_encodings = tokenize\_data([user\_input])

input\_ids = user\_input\_encodings['input\_ids'].to(device)

attention\_mask = user\_input\_encodings['attention\_mask'].to(device)

# Predict intent using BERT

with torch.no\_grad():

logits = model(input\_ids, attention\_mask)

preds = torch.argmax(logits, dim=-1)

# Get the predicted intent

intent\_encoded = preds.item()

intent = label\_encoder.inverse\_transform([intent\_encoded])[0]

# Fetch a random response for the predicted intent

for intent\_data in data['intents']:

if intent\_data['tag'] == intent:

response = random.choice(intent\_data['responses'])

return response

# Test the bot

user\_input = "who are faculty of cse department?"

response = get\_response(user\_input)

print("Bot:", response)

OUTPUT :

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Epoch 10

Accuracy: 60.41%

Epoch 20

Accuracy: 73.60%

Epoch 30

Accuracy: 77.66%

Epoch 40

Accuracy: 77.66%

Epoch 50

Accuracy: 79.19%

Epoch 60

Accuracy: 79.19%

Epoch 70

Accuracy: 78.68%

Epoch 80

Accuracy: 79.19%

Epoch 90

Accuracy: 79.70%

Epoch 100

Accuracy: 79.70%

Bot: The faculty at our college are highly qualified and experienced in their respective fields.

LINK FOR COLAB :

<https://colab.research.google.com/drive/1D42cbWv7IZStrKITeHkM5QvbOa-H7ROB?usp=sharing>

ABOUT MODEL

1. The model uses BERT for sequence classification to predict user intents.
2. It is trained on a dataset with Total Number of Patterns: 983
3. Total Number of Responses: 455
4. The dataset is preprocessed by tokenizing text and encoding labels.
5. The model is trained using the AdamW optimizer and CrossEntropyLoss.
6. The model's response is selected randomly from predefined responses based on the predicted intent.
7. The code uses the Hugging Face Transformers library to load and fine-tune BERT.