import tensorflow as tf

import numpy as np

import nltk

from nltk.stem import WordNetLemmatizer

from tensorflow.keras.layers import Input, Dense, Dropout

from tensorflow.keras.models import Model

# Load the education dataset

data = np.load('education\_data.npy', allow\_pickle=True)

X = data[:, 0]

y = data[:, 1]

# Initialize the lemmatizer

lemmatizer = WordNetLemmatizer()

# Tokenize and lemmatize the text data

corpus = []

for i in range(len(X)):

words = nltk.word\_tokenize(X[i].lower())

words = [lemmatizer.lemmatize(word) for word in words]

corpus.append(' '.join(words))

# Convert the text data into a one-hot encoded matrix

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

tokenizer = Tokenizer(num\_words=5000, oov\_token="<OOV>")

tokenizer.fit\_on\_texts(corpus)

X\_train = tokenizer.texts\_to\_sequences(corpus)

X\_train = pad\_sequences(X\_train, padding='post', maxlen=100)

vocab\_size = len(tokenizer.word\_index) + 1

# Define the model

inputs = Input(shape=(100,))

x = Dense(128, activation='relu')(inputs)

x = Dropout(0.5)(x)

x = Dense(64, activation='relu')(x)

x = Dropout(0.5)(x)

outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs=inputs, outputs=outputs)

# Compile the model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y, epochs=50, batch\_size=32, validation\_split=0.2)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

# Save the model

model.save('education\_chatbot\_model.h5')

# Use the model to generate responses

def generate\_response(text):

words = nltk.word\_tokenize(text.lower())

words = [lemmatizer.lemmatize(word) for word in words]

sequences = tokenizer.texts\_to\_sequences([' '.join(words)])

padded\_sequences = pad\_sequences(sequences, padding='post', maxlen=100)

prediction = model.predict(padded\_sequences)[0][0]

if prediction > 0.5:

return "Yes"

else:

return "No"

# Importing necessary libraries

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Reading the dataset

data = pd.read\_csv('student\_data.csv')

# Splitting the dataset into training and testing sets

X = data.drop('Grade', axis=1)

y = data['Grade']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Training the decision tree classifier

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

# Predicting grades for the test set

y\_pred = clf.predict(X\_test)

# Evaluating the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

print('Accuracy:', accuracy)

#Project code