app

April 14, 2024

1 AIDI 2004 - GROUP 3

2 FINAL PROJECT

ONTARIO LANDLOARD AND TENANT TRIBUNAL CHATBOT

1) Import Libraries.

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import gradio as gr
import joblib
```

2) Load data and split dataset.

[ ]:
```
# Load data from CSV
data = pd.read_csv("landlord_data.csv")

# Split data into train, validation, and test sets
X_train, X_test, y_train, y_test = train_test_split(data["instruction"],␣
↪data["output"], test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪2, random_state=42)
```

3) Define Multinomial Naive Bayes model.

[ ]:
```
# Define a vectoriser
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
```

```
# Train the Multinomial Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_counts, y_train)
```

```
[ ]: MultinomialNB()
```

4) Preprocess data.

```
[ ]: # Define Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

# Define max_len
max_len = 100  # Adjust as needed

# Tokenize and pad sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
vocab_size = len(tokenizer.word_index) + 1
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Combine training and test sets for label encoding
combined_labels = pd.concat([y_train, y_test])

# Initialize LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(combined_labels)

# Fit label encoder and transform labels
y_train_encoded = label_encoder.transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

num_unique_labels = len(set(combined_labels))
```

5) Define and train LSTM model.

```
[ ]: model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=100, input_length=max_len),
    LSTM(units=100),
    Dense(units=num_unique_labels, activation='softmax')  # Adjust output units␣
 ↪based on the number of unique labels
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train_pad, y_train_encoded, epochs=10, batch_size=32,␣
 ↪validation_data=(X_test_pad, y_test_encoded))
```

```
Epoch 1/10

C:\Users\d-kin\AppData\Roaming\Python\Python312\site-
packages\keras\src\layers\core\embedding.py:86: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(

60/60              10s 89ms/step -
accuracy: 0.0000e+00 - loss: 7.8001 - val_accuracy: 0.0000e+00 - val_loss:
7.7755
Epoch 2/10
60/60              5s 74ms/step -
accuracy: 2.2291e-04 - loss: 7.8415 - val_accuracy: 0.0000e+00 - val_loss:
7.9366
Epoch 3/10
60/60              4s 72ms/step -
accuracy: 9.6742e-04 - loss: 7.7530 - val_accuracy: 0.0000e+00 - val_loss:
8.8198
Epoch 4/10
60/60              4s 74ms/step -
accuracy: 0.0012 - loss: 7.6365 - val_accuracy: 0.0050 - val_loss: 8.6347
Epoch 5/10
60/60              5s 77ms/step -
accuracy: 0.0030 - loss: 7.6050 - val_accuracy: 0.0050 - val_loss: 8.8605
Epoch 6/10
60/60              4s 75ms/step -
accuracy: 0.0034 - loss: 7.5806 - val_accuracy: 0.0050 - val_loss: 9.8093
Epoch 7/10
60/60              4s 74ms/step -
accuracy: 0.0049 - loss: 7.4931 - val_accuracy: 0.0050 - val_loss: 10.2487
Epoch 8/10
60/60              5s 79ms/step -
accuracy: 0.0037 - loss: 7.4405 - val_accuracy: 0.0050 - val_loss: 10.2096
Epoch 9/10
60/60              5s 77ms/step -
accuracy: 0.0068 - loss: 7.1973 - val_accuracy: 0.0050 - val_loss: 11.0172
Epoch 10/10
60/60              5s 76ms/step -
accuracy: 0.0042 - loss: 7.0424 - val_accuracy: 0.0050 - val_loss: 11.2280
```

[ ]: <keras.src.callbacks.history.History at 0x163276d20c0>

6) Save local instances of the models

[ ]:
```python
# Save the Naive Bayes classifier
joblib.dump(clf, "naive_bayes_model.joblib")

# Load the Naive Bayes classifier
clf = joblib.load("naive_bayes_model.joblib")
```

```
# Save LSTM Model
model.save("lstm_model.keras")
```

7) Evaluate Both model's performance

[ ]:
```
# Evaluate Naive Bayes model
X_val_counts = vectorizer.transform(X_val)
y_val_pred_nb = clf.predict(X_val_counts)
accuracy = accuracy_score(y_val, y_val_pred_nb)
print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 0.012578616352201259
```

[ ]:
```
# Evaluate LSTM model
X_val_pad = pad_sequences(tokenizer.texts_to_sequences(X_val), maxlen=max_len,␣
 ↪padding='post')
y_val_pred_lstm = model.predict(X_val_pad)
y_val_pred_lstm_classes = label_encoder.inverse_transform(y_val_pred_lstm.
 ↪argmax(axis=-1))
loss, accuracy = model.evaluate(X_test_pad, y_test_encoded)
print("LSTM Model Performance:")
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
15/15           1s 44ms/step
19/19           1s 38ms/step -
accuracy: 0.0042 - loss: 11.2447
LSTM Model Performance:
Test Loss: 11.22800350189209
Test Accuracy: 0.005025125574320555
```

8) Define the function to get the most similar question from the training set.

[ ]:
```
def get_most_similar_question(input_question):
    input_question_vector = vectorizer.transform([input_question])
    similarity = input_question_vector.dot(X_train_counts.T)
    most_similar_index = similarity.argmax()
    max_similarity = similarity[0, most_similar_index]
    if max_similarity > 0.7:  # Adjust threshold as needed
        return data.iloc[most_similar_index]["instruction"]
    else:
        return None
```

9) Define the Naive Bayes-based chatbot function.

[ ]:
```
def chatbot_nb(question):
    # Vectorize the input question using the same CountVectorizer object
    input_question_vector = vectorizer.transform([question])
```

```
    # Calculate the probability scores for each class (answer) using the␣
↪trained Naive Bayes classifier
    probabilities = clf.predict_proba(input_question_vector)

    # Find the index of the class (answer) with the highest probability
    most_probable_index = np.argmax(probabilities)

    # Retrieve the corresponding answer from the classes
    answer_from_nb = clf.classes_[most_probable_index]

    # Calculate the similarity between the input question and the training␣
↪questions
    similarity = input_question_vector.dot(X_train_counts.T)
    most_similar_index = similarity.argmax()
    max_similarity = similarity[0, most_similar_index]

    # If both the Naive Bayes prediction and similarity-based prediction agree␣
↪or if the similarity is above a certain threshold, return the answer
    if (answer_from_nb == data.iloc[most_similar_index]["instruction"]) or␣
↪(max_similarity > 0.7):
        return answer_from_nb
    else:
        return None
```

10) Define the LSTM-based chatbot function.

[ ]:
```
def chatbot_lstm(question):
    input_sequence = tokenizer.texts_to_sequences([question])
    input_sequence_pad = pad_sequences(input_sequence, maxlen=max_len,␣
↪padding='post')
    prediction = model.predict(input_sequence_pad)
    predicted_class = label_encoder.inverse_transform([prediction.argmax()])[0]
    return predicted_class
```

11) Fine tune both models

[ ]:
```
# Fine-tune Naive Bayes model
def fine_tune_naive_bayes(X_train_new, y_train_new):
    # Load the saved Naive Bayes classifier
    clf = joblib.load("naive_bayes_model.joblib")

    # # Update the model with new data
    # X_train_new_counts = vectorizer.transform(X_train_new)
    clf.partial_fit(X_train_new, y_train_new, classes=np.unique(y_train_new))

    # Save the fine-tuned model
    joblib.dump(clf, "naive_bayes_fine.joblib")
```

[ ]:
```
# Fine-tune the Naive Bayes model with additional data
fine_tune_naive_bayes(X_train_counts, y_train)
```

[ ]:
```
def fine_tune_lstm(X_train_new, y_train_new):
    # Load the saved LSTM model
    model = tf.keras.models.load_model("lstm_model.keras")

    # # Preprocess the new data
    # X_train_new_seq = tokenizer.texts_to_sequences(X_train_new)
    # X_train_new_pad = pad_sequences(X_train_new_seq, maxlen=max_len,␣
↪padding='post')

    # Fine-tune the model
    model.fit(X_train_new, y_train_new, epochs=5, batch_size=32,␣
↪validation_split=0.1)

    # Save the fine-tuned model
    model.save("lstm_model_fine.keras")
```

[ ]:
```
# Fine-tune the LSTM model with additional data
fine_tune_lstm(X_test_pad, y_test_encoded)
```

```
Epoch 1/5
17/17            4s 109ms/step -
accuracy: 0.0000e+00 - loss: 9.7133 - val_accuracy: 0.0000e+00 - val_loss:
7.2741
Epoch 2/5
17/17            2s 110ms/step -
accuracy: 0.0000e+00 - loss: 6.8575 - val_accuracy: 0.0000e+00 - val_loss:
9.0272
Epoch 3/5
17/17            2s 105ms/step -
accuracy: 9.9122e-04 - loss: 6.5712 - val_accuracy: 0.0000e+00 - val_loss:
9.5163
Epoch 4/5
17/17            2s 90ms/step -
accuracy: 0.0000e+00 - loss: 6.5752 - val_accuracy: 0.0000e+00 - val_loss:
9.7205
Epoch 5/5
17/17            1s 68ms/step -
accuracy: 0.0054 - loss: 6.4956 - val_accuracy: 0.0000e+00 - val_loss: 9.6865
```

12) Evaluate Finetuned models

[ ]:
```
# Evaluate Naive Bayes model
X_val_counts = vectorizer.transform(X_val)
y_val_pred_nb = clf.predict(X_val_counts)
accuracy = accuracy_score(y_val, y_val_pred_nb)
```

```
print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 0.012578616352201259
```

[ ]:
```
# Evaluate LSTM model
X_val_pad = pad_sequences(tokenizer.texts_to_sequences(X_val), maxlen=max_len,␣
 ↪padding='post')
y_val_pred_lstm = model.predict(X_val_pad)
y_val_pred_lstm_classes = label_encoder.inverse_transform(y_val_pred_lstm.
 ↪argmax(axis=-1))
loss, accuracy = model.evaluate(X_test_pad, y_test_encoded)
print("LSTM Model Performance:")
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
15/15              0s 22ms/step
19/19              1s 27ms/step -
accuracy: 0.0042 - loss: 11.2447
LSTM Model Performance:
Test Loss: 11.22800350189209
Test Accuracy: 0.005025125574320555
```

13) Define user interface.

[ ]:
```
# Create a Gradio interface for Naive Bayes-based chatbot
chatbot_nb_interface = gr.Interface(fn=chatbot_nb, inputs="text",␣
 ↪outputs="text", title="Naive Bayes Chatbot")

# Create a Gradio interface for LSTM-based chatbot
chatbot_lstm_interface = gr.Interface(fn=chatbot_lstm, inputs="text",␣
 ↪outputs="text", title="LSTM Chatbot")
```

[ ]:
```
chatbot_nb_interface.launch(share=False)
```

```
Running on local URL:  http://127.0.0.1:7864

To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[ ]:

[ ]:
```
chatbot_lstm_interface.launch(share=False)
```

```
Running on local URL:  http://127.0.0.1:7865


To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[ ]:

```
1/1                0s 34ms/step
1/1                0s 27ms/step
1/1                0s 19ms/step
1/1                0s 36ms/step
1/1                0s 19ms/step
1/1                0s 19ms/step
```