```python
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split

# Load data
file_path = 'urgency_train_data.csv'
data = pd.read_csv(file_path)

print("Column names in dataset:", data.columns)

# Preprocess text & labels
texts = data['description'].astype(str).tolist()
# Convert categorical urgency labels to numerical values
label_mapping = {'Low': 0, 'Medium': 1, 'High': 2}  # Modify if needed
data['urgency_label'] = data['urgency_label'].map(label_mapping)
# Check for NaN values and handle them
if data['urgency_label'].isna().sum() > 0:
    print("Warning: Found NaN values in 'urgency_label'. Replacing with defaul
    data['urgency_label'] = data['urgency_label'].fillna(0)  # You can choose

# Now convert to numpy array
labels = data['urgency_label'].astype(int).values

# Tokenization
num_words = 10000  # Increased vocabulary size
tokenizer = Tokenizer(num_words=num_words, oov_token='<OOV>')
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
max_len = max(len(seq) for seq in sequences)
X = pad_sequences(sequences, maxlen=max_len, padding='post')
y = np.array(labels)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, strat

# Compute class weights
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y
class_weight_dict = {i: class_weights[i] for i in range(len(class_weights))}

# Load GloVe embeddings
embedding_dim = 200
glove_path = 'glove.6B.200d.txt'  # Make sure to have this file
glove_embeddings = {}
with open(glove_path, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        glove_embeddings[word] = vector

embedding_matrix = np.zeros((num_words, embedding_dim))
for word, index in tokenizer.word_index.items():
    if index < num_words:
        vector = glove_embeddings.get(word)
        if vector is not None:
            embedding_matrix[index] = vector

# Build LSTM model
model = Sequential([
    Embedding(num_words, embedding_dim, weights=[embedding_matrix], input_leng
    LSTM(128, return_sequences=True),
    Dropout(0.5),
    LSTM(64),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dense(len(np.unique(y)), activation='softmax')
])
```

ModuleNotFoundError TypeError ✕ •••

Veuillez expliquer l'erreur :

```
TypeError: 'Tokenizer' object is not callable
```

Hypothesis

The error message `TypeError: 'Tokenizer' object is not callable` indicates that you are trying to call the `tokenizer` object as a function, which is not how it's used for text to sequence conversion after it has been fit. The `tokenizer` object from `tensorflow.keras.preprocessing.text.Tokenizer` is fit once on the training data and then used to transform text to sequences using its methods like `texts_to_sequences`. In the provided code, you directly call `tokenizer` with the `new_texts` which causes the error.

Suggested Changes

```
s, padding='max_length', truncation=True, max


es(new_texts) # Use texts_to_sequences to con
maxlen=max_len, padding='post') # Pad the seq
```

Use code with caution

Évaluer cette réponse

```
# Compile model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metric

# Train model with early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_we
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=30, batch
          class_weight=class_weight_dict, callbacks=[early_stopping])

# Evaluate model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_acc:.4f}')

# Evaluate Model on Test Data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'🔥 Final Test Accuracy: {test_acc:.4f}')

# ================================
# 🔹 Load & Evaluate on New Test Data
# ================================
new_test_file = 'urgency_test_data.csv'
new_test_data = pd.read_csv(new_test_file)

# Check column names
print("Column names in new test dataset:", new_test_data.columns)

# Preprocess text (tokenize & pad sequences)
new_texts = new_test_data['description'].astype(str).tolist()
new_sequences = tokenizer.texts_to_sequences(new_texts)
new_X = pad_sequences(new_sequences, maxlen=max_len, padding='post')

# Convert categorical labels
new_test_data['urgency_label'] = new_test_data['urgency_label'].map(label_mapp
new_test_data['urgency_label'] = new_test_data['urgency_label'].fillna(0).asty
new_y = new_test_data['urgency_label'].values

# Evaluate on the new test dataset
new_test_loss, new_test_acc = model.evaluate(new_X, new_y)
print(f'🚀 Accuracy on New Test Data: {new_test_acc:.4f}')

# ================================
# 🔹 (Optional) Save Predictions for Analysis
# ================================
predictions = model.predict(new_X)
predicted_labels = np.argmax(predictions, axis=1)  # Convert probabilities to

# Compare actual vs. predicted labels
comparison_df = pd.DataFrame({
    'description': new_texts,
    'actual_label': new_y,
    'predicted_label': predicted_labels
})

# Save results to CSV
comparison_df.to_csv('model_evaluation_results.csv', index=False)
print("✅ Model evaluation results saved to 'model_evaluation_results.csv'")
```

```
Column names in dataset: Index(['description', 'category', 'deadline_pro
       'urgency_label'],
      dtype='object')
Warning: Found NaN values in 'urgency_label'. Replacing with default valu
Epoch 1/30
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.p
  warnings.warn(
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - accuracy: 0.3298 - loss: 1.1
Epoch 2/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 31ms/step - accuracy: 0.3157 - loss: 1.09
Epoch 3/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 48ms/step - accuracy: 0.3230 - loss: 1.10
Epoch 4/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 0.3215 - loss: 1.09
Epoch 5/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 29ms/step - accuracy: 0.3279 - loss: 1.10
Epoch 6/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 38ms/step - accuracy: 0.3886 - loss: 1.09
Epoch 7/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 0.3107 - loss: 1.10
```

```
Epoch 8/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 46ms/step - accuracy: 0.3076 - loss: 1.1(
Epoch 9/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 8s 29ms/step - accuracy: 0.3137 - loss: 1.1(
Epoch 10/30
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - accuracy: 0.3910 - loss: 1.0(
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.4200 - loss: 1.089
Test Accuracy: 0.4150
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.4200 - loss: 1.089
🔥 Final Test Accuracy: 0.4150
Column names in new test dataset: Index(['description', 'category', 'dea(
       'urgency_label'],
      dtype='object')
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.4517 - loss: 1.0885
🚀 Accuracy on New Test Data: 0.4600
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 213ms/step
✅ Model evaluation results saved to 'model_evaluation_results.csv'
```

Saisissez une requête ici

0/2000

Les réponses peuvent contenir des informations inexactes ou choquantes qui ne représentent pas le point de vue de Google. En savoir plus