Dataprocessing+train+inference4

January 18, 2024

[ ]:
```python
import pandas as pd
```

[ ]:
```python
import os
import pandas as pd
import re

def read_and_concatenate_files_with_labels_and_user(folder_paths,
     column_names):
    # Initialize an empty DataFrame
    result_df = pd.DataFrame(columns=column_names + ['Label',
                    'User'])

    # Iterate through folder paths
    for folder_label, folder_path in zip(['Reading',
                     'Speaking', 'Watching'],
                     folder_paths):
        # Initialize an empty list to store DataFrames
        dfs = []

        # Get a sorted list of files in the folder
        files_to_process = sorted([file_name for
    file_name in os.listdir(folder_path)
    if file_name.endswith('.csv')])

        # Iterate through sorted files in the folder
        for file_name in files_to_process:
            file_path = os.path.join(folder_path, file_name)

            # Read the CSV file without column names and concatenate rows
            df = pd.read_csv(file_path, header=None,
                             names=column_names)

            # Extract numerical user information from
            #the file name using regular expression
            user_match = re.search(r'(\d+)', file_name)
            user_info = int(user_match.group(1)) if \
            user_match else None
```

```
            # Add 'Label' and 'User' columns
            df['Label'] = folder_label
            df['User'] = user_info
            dfs.append(df)

            # Print statement for debugging
            #print(f"Processed file: {file_name},
            #User: {user_info}, Label: {folder_label}")

        # Concatenate the list of DataFrames vertically
        result_df = pd.concat([result_df, pd.concat(dfs,
                    ignore_index=True)], ignore_index=True)

    return result_df

# Example usage:
folder_paths = ['Data/Reading',
    'Data/Speaking', 'Data/Watching']
column_names = ['EEG1', 'EEG2', 'Acc_X', 'Acc_Y', 'Acc_Z']

result_dataframe = read_and_concatenate_files_with_labels_and_user(folder_paths,
                 column_names)

# Print the unique values in the "User" column
#print(result_dataframe['User'].unique())

# Display the resulting DataFrame
print(result_dataframe)
```

```
              EEG1        EEG2       Acc_X       Acc_Y       Acc_Z     Label  \
0       842.229919  847.164856 -656.251038  789.063721  136.718964   Reading
1       845.519897  853.744812 -660.157288  792.969971  136.718964   Reading
2       847.164856  858.679748 -656.251038  792.969971  136.718964   Reading
3       843.874939  852.099793 -656.251038  792.969971  140.625214   Reading
4       847.164856  857.034729 -656.251038  792.969971  136.718964   Reading
…              …           …           …           …           …         …
104475  847.164856  857.034729 -703.126099  750.001160  136.718964  Watching
104476  875.129517  837.294983 -703.126099  750.001160  136.718964  Watching
104477  852.099793  837.294983 -703.126099  750.001160  136.718964  Watching
104478  832.360046  870.194580 -707.032349  746.094910  132.812714  Watching
104479  843.874939  843.874939 -703.126099  746.094910  136.718964  Watching

       User
0         1
1         1
2         1
```

```
3         1
4         1
…       …
104475    6
104476    6
104477    6
104478    6
104479    6

[104480 rows x 7 columns]
```

[ ]:
```
User = result_dataframe['User']

# Separate the data into features (X) and target variable (y)
X = result_dataframe[['EEG1', 'EEG2']]  # Features for EEG1 and EEG2
y = result_dataframe['Label']  # Target variable

# Filter data for User 1
X_test = X[User == 1]
y_test = y[User == 1]

# Filter data for training (excluding User 1)
X_train = X[User != 1]
y_train = y[User != 1]

# Print the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train: " ,y_train.shape)
print("y_test: ", y_test.shape)
```

```
X_train shape: (93537, 2)
X_test shape: (10943, 2)
y_train:  (93537,)
y_test:  (10943,)
```

[ ]:
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
y_train_encoded = encoder.fit_transform(
    y_train.values.reshape(-1, 1))

y_test_encoded = encoder.transform(
    y_test.values.reshape(-1, 1))


y_train_encoded.shape, y_test_encoded.shape
```

[ ]: ((93537, 3), (10943, 3))

[ ]:
```
y_train_encoded, y_train_encoded.shape
```

[ ]:
```
(array([[1., 0., 0.],
        [1., 0., 0.],
        [1., 0., 0.],
        …,
        [0., 0., 1.],
        [0., 0., 1.],
        [0., 0., 1.]]),
 (93537, 3))
```

[ ]:
```
import pandas as pd
import numpy as np

# Assuming X_train is your DataFrame and activities_array is your one-hot␣
↪encoded activities array

# Set the window size
window_size = 400  # You can adjust this value based on your requirement

# Function to create sliding windows and corresponding labels
def create_sliding_windows(data, labels, window_size):
    X, y = [], []
    for i in range(len(data) - window_size + 1):
        window = data[i:i+window_size]
        label = labels[i+window_size-1]
        X.append(window)
        y.append(label)
    return np.array(X), np.array(y)

# Extract feature columns from the DataFrame
X_features = X_train[['EEG1', 'EEG2']].values

# Create sliding windows and labels
X_windows_train, y_windows_train = create_sliding_windows(X_features,
    y_train_encoded, window_size)

# Print the shape of the resulting arrays
print("X_windows shape:", X_windows_train.shape)
print("y_labels shape:", y_windows_train.shape)
```

```
X_windows shape: (93138, 400, 2)
y_labels shape: (93138, 3)
```

[ ]:
```python
X_features_test = X_test[['EEG1', 'EEG2']].values
X_windows_test, y_windows_test = create_sliding_windows(
X_features_test, y_test_encoded, window_size=window_size
)
```

[ ]:
```python
X_windows_test.shape, y_windows_test.shape, y_windows_test
```

[ ]:
```
((10544, 400, 2),
 (10544, 3),
 array([[1., 0., 0.],
        [1., 0., 0.],
        [1., 0., 0.],
        …,
        [0., 0., 1.],
        [0., 0., 1.],
        [0., 0., 1.]]))
```

[ ]:
```python
import tensorflow as tf
from tensorflow import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, \
Flatten, Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras import regularizers
import matplotlib.pyplot as plt


# Assuming 'X' is your input data of shape
#(1044, 100, 6) and 'y' is your corresponding labels
train_steps_per_epoch = len(X_windows_train)
val_steps_per_epoch  = len(X_windows_test)
# Define the CNN model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, \
MaxPooling1D, Flatten, Dense, Dropout, LSTM
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.callbacks import LearningRateScheduler

tf.random.set_seed(1234)




model = Sequential([
    Conv1D(filters=4, kernel_size=1, strides=1,
```

```python
            activation='relu',
            input_shape=(X_windows_train.shape[1],
                         X_windows_train.shape[2])),
    Dropout(0.1),

    Conv1D(filters=4, kernel_size=2,
           strides=1, activation='relu'),
    Conv1D(filters=4, kernel_size=3, strides=1, activation='relu'),
    Dropout(0.1),
    MaxPooling1D(pool_size=1,
                 strides=1, padding='valid'),
    Flatten(),
    Dense(3, activation='relu'),

    Dense(3, activation='softmax')
])





# Compile the model
model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy'])

# Train the model
history= model.fit(X_windows_train, y_windows_train,
    epochs=10,
    validation_data=(X_windows_test, y_windows_test)



     )



#steps_per_epoch=train_steps_per_epoch,
#validation_steps=val_steps_per_epoch




# Accessing the history of training
training_accuracy = history.history.get('accuracy') \
    or history.history.get('acc')
training_loss = history.history['loss']
validation_accuracy = history.history.get('val_accuracy')\
or history.history.get('val_acc')
```

```python
validation_loss = history.history.get('val_loss') \
or history.history.get('validation_loss')

# Plotting accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
if training_accuracy:
    plt.plot(training_accuracy, label='Training Accuracy')
if validation_accuracy:
    plt.plot(validation_accuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

# Plotting loss
plt.subplot(1, 2, 2)
plt.plot(training_loss, label='Training Loss')
if validation_loss:
    plt.plot(validation_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/10
2911/2911 [==============================] - 12s 4ms/step - loss: 1.0862 -
accuracy: 0.5665 - val_loss: 1.0384 - val_accuracy: 0.5174
Epoch 2/10
2911/2911 [==============================] - 10s 3ms/step - loss: 0.9831 -
accuracy: 0.5684 - val_loss: 1.0211 - val_accuracy: 0.5174
Epoch 3/10
2911/2911 [==============================] - 11s 4ms/step - loss: 0.9547 -
accuracy: 0.5684 - val_loss: 1.0251 - val_accuracy: 0.5174
Epoch 4/10
2911/2911 [==============================] - 11s 4ms/step - loss: 0.9455 -
accuracy: 0.5684 - val_loss: 1.0328 - val_accuracy: 0.5174
Epoch 5/10
2911/2911 [==============================] - 10s 3ms/step - loss: 0.9431 -
accuracy: 0.5684 - val_loss: 1.0378 - val_accuracy: 0.5174
Epoch 6/10
2911/2911 [==============================] - 11s 4ms/step - loss: 0.9426 -
accuracy: 0.5684 - val_loss: 1.0406 - val_accuracy: 0.5174
Epoch 7/10
```

```
2911/2911 [==============================] - 10s 4ms/step - loss: 0.9425 -
accuracy: 0.5684 - val_loss: 1.0420 - val_accuracy: 0.5174
Epoch 8/10
2911/2911 [==============================] - 11s 4ms/step - loss: 0.9424 -
accuracy: 0.5684 - val_loss: 1.0429 - val_accuracy: 0.5174
Epoch 9/10
2911/2911 [==============================] - 10s 4ms/step - loss: 0.9426 -
accuracy: 0.5684 - val_loss: 1.0431 - val_accuracy: 0.5174
Epoch 10/10
2911/2911 [==============================] - 10s 3ms/step - loss: 0.9424 -
accuracy: 0.5684 - val_loss: 1.0433 - val_accuracy: 0.5174
```