Lab2 Homework - predict the emotion

黃麗蘋 NCU 111409519

Step 1: Loading the Data

In this first step, I begin by loading data from different sources (CSV and JSON files). I use `pandas` to load the CSV files into DataFrames, which allows me to work with structured tabular data. For the tweet data stored in a JSON file, I parse the file line-by-line, extract key attributes (such as tweet ID, text, and hashtags), and organize them into a DataFrame. This structured data will form the foundation for further analysis in the upcoming steps.

```
[ ] # Paths to uploaded files
    data_identification_file = '/content/data_identification.csv'
    emotion_file = '/content/emotion.csv'
    tweets_DM_file = '/content/tweets_DM.json'

    # Load CSV files
    data_identification_df = pd.read_csv(data_identification_file)
    emotion_df = pd.read_csv(emotion_file)


    # Load the JSON file
    tweets_data = []
    with open(tweets_DM_file, 'r') as f:
        for line in f:
            tweet_json = json.loads(line)
            tweet_id = tweet_json['_source']['tweet']['tweet_id']
            text = tweet_json['_source']['tweet']['text']
            hashtags = tweet_json['_source']['tweet']['hashtags']
            tweets_data.append({'tweet_id': tweet_id, 'text': text, 'hashtags': hashtags})

    # Convert the list of dictionaries to a DataFrame
    tweets_df = pd.DataFrame(tweets_data)
```

Step 2. Merging and Splitting the Data

In this step, I merge multiple DataFrames (tweets_df, data_identification_df, and emotion_df) to create a comprehensive dataset that includes tweet information, metadata (train/test identification), and emotional labels. I then split the data into two subsets: one for training (training_df) and one for testing (ans_df). After merging the emotion data into the training set, I drop the identification column as it is no longer needed.

Finally, I try to sample 5%, 10%, 30% of the training data to reduce the size for further analysis, ensuring that the random sampling is reproducible by setting a fixed random seed.

```
[ ] # Merge tweets_df and data_identification_df on 'tweet_id'
    merged_df = pd.merge(tweets_df, data_identification_df, on='tweet_id', how='inner')

    # Split into training_df and ans_df based on 'identification'
    training_df = merged_df[merged_df['identification'] == 'train']
    ans_df = merged_df[merged_df['identification'] == 'test']

    # Merge emotion to the training_df
    training_df = pd.merge(training_df, emotion_df, on='tweet_id', how='inner')

    # Drop the 'identification' column if not needed
    training_df = training_df.drop(columns=['identification'])
    ans_df = ans_df.drop(columns=['identification'])
```

```
[ ] training_df = training_df.sample(frac=0.05, random_state=42).reset_index(drop=True)
```

Step 3. Merging and Splitting the Data

In this step, I preprocess the data by performing tokenization and label encoding. First, I split the training data into training and testing subsets. Then, I separate the features (tweet text and hashtags) from the target labels (emotion labels).

To convert the emotion labels into numerical values, I use LabelEncoder and save the encoder for future use. I load the pre-trained BERT tokenizer and define a function to tokenize the tweet text, ensuring each tweet is truncated or padded to a consistent length.

I then convert the data into Hugging Face Dataset objects and apply tokenization to both the training and test datasets. Finally, I add the encoded labels to these datasets so that they are ready for model training.

```
[ ] import joblib
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder

    # 1. Train/test split
    train_df, test_df = train_test_split(training_df, test_size=0.2, random_state=42)

    # 2. Separate features and labels
    X_train = train_df.drop(columns=['emotion'])  # Features (input data), drop 'emotion' column
    y_train = train_df['emotion']  # Labels (target)

    X_test = test_df.drop(columns=['emotion'])  # Features (input data), drop 'emotion' column
    y_test = test_df['emotion']  # Labels (target)

    # 3. Label encoding the emotion labels
    label_encoder = LabelEncoder()
    train_labels = label_encoder.fit_transform(y_train)  # Encode training labels
    test_labels = label_encoder.transform(y_test)  # Encode test labels

    # Save the LabelEncoder to a file
    joblib.dump(label_encoder, 'label_encoder.joblib')
    print("LabelEncoder has been saved to 'label_encoder.joblib'.")
```

顯示隱藏的輸出內容

```
[ ] # Load BERT tokenizer
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

    def tokenize_function(examples):
        return tokenizer(examples['text'], truncation=True, padding='max_length', max_length=128)
```

顯示隱藏的輸出內容

```
[ ] # Convert train_df and test_df to HuggingFace Datasets
    train_dataset = Dataset.from_pandas(X_train)
    test_dataset = Dataset.from_pandas(X_test)

    # Apply tokenization
    train_dataset = train_dataset.map(tokenize_function, batched=True)
    test_dataset = test_dataset.map(tokenize_function, batched=True)

    # Add labels to datasets
    train_dataset = train_dataset.add_column("labels", train_labels)
    test_dataset = test_dataset.add_column("labels", test_labels)
```

顯示隱藏的輸出內容

Step 4. TensorFlow Dataset Conversion

In this step, I convert the tokenized datasets into TensorFlow datasets using the dataset_to_tf() function. This function takes the Hugging Face Datasets, extracts the required features (such as input_ids, attention_mask, and optionally token_type_ids), and prepares them for use in TensorFlow. I also handle the inclusion of labels when with_labels=True and apply batching, shuffling, and prefetching for efficient training. The result is two TensorFlow datasets, train_dataset_tf and test_dataset_tf, which are ready to be used for training and evaluation in TensorFlow.

```
[ ] def dataset_to_tf(dataset, with_labels=True, batch_size=32):
        # Prepare inputs
        inputs = {
            'input_ids': dataset['input_ids'],
            'attention_mask': dataset['attention_mask'],
        }

        # Include token_type_ids if available
        if 'token_type_ids' in dataset.features:
            inputs['token_type_ids'] = dataset['token_type_ids']

        if with_labels and 'labels' in dataset.column_names:
            labels = dataset['labels']
            return tf.data.Dataset.from_tensor_slices((
                inputs,
                labels
            )).batch(batch_size).shuffle(buffer_size=1000).prefetch(tf.data.AUTOTUNE)
        else:
            return tf.data.Dataset.from_tensor_slices(inputs).batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

```
[ ] # Convert train and test datasets to TensorFlow dataset
    train_dataset_tf = dataset_to_tf(train_dataset, with_labels=True, batch_size=32)
    test_dataset_tf = dataset_to_tf(test_dataset, with_labels=True, batch_size=32)
```

Step 5. Model Training

In this step, I define and train a BERT-based model for emotion classification. After determining the number of labels (unique emotions), I load a pre-trained BERT model for sequence classification and customize it for this task.

I compile the model with the Adam optimizer, a sparse categorical cross-entropy loss function, and accuracy as the evaluation metric. The model is trained on the train_dataset_tf and validated on the test_dataset_tf for 2, 3 epochs. After training, I print the final training and validation accuracy and save the trained model for future use.

```
# Define number of labels
num_labels = len(set(train_dataset['labels']))

# Load pre-trained BERT model for sequence classification
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=num_labels)

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

# Ensure to use the correct `Adam` optimizer
model.compile(optimizer=optimizer,  # Pass the optimizer instance
              loss=loss,
              metrics=metrics)

# Train the model
history = model.fit(train_dataset_tf, validation_data=test_dataset_tf, epochs=2)

# Print the training and validation accuracy after training
print("Training Accuracy: ", history.history['accuracy'][-1])
print("Validation Accuracy: ", history.history['val_accuracy'][-1])

# Save the trained model
model.save_pretrained("./emotion_classification_model")
```

Step 6. Predict and Add Predictions to ans_df

In this step, I load the previously saved model and LabelEncoder to perform predictions on the test data (ans_df). I convert the test DataFrame into a Hugging Face Dataset, apply the same tokenization and preprocessing as with the training data, and then convert it into a TensorFlow dataset. I run the predictions using the trained BERT model, extract the predicted labels, and inverse transform them back to the original emotion categories using the LabelEncoder. Finally, I add the predicted emotions to the DataFrame and save the results to a CSV file for future analysis or submission.

```
import pickle

# Load the LabelEncoder from the file
with open('label_encoder.pkl', 'rb') as file:
    label_encoder = pickle.load(file)
print("LabelEncoder has been loaded from 'label_encoder.pkl'.")
```

```
# Load the configuration
config = BertConfig.from_json_file('config.json')

# Load the model with the configuration
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', config=config)

# Load the weights from the .h5 file (TensorFlow model)
model.load_weights('tf_model.h5')  # Ensure the path is correct

# Verify the model architecture
print(model.summary())
```

顯示隱藏的輸出內容

6.2 Convert ans_df

```
# Convert ans_df to Hugging Face Dataset
ans_dataset = Dataset.from_pandas(ans_df)


# Tokenize ans_df using the tokenizer
ans_dataset = ans_dataset.map(tokenize_function, batched=True)
```

顯示隱藏的輸出內容

```
# Convert ans_dataset to TensorFlow dataset without labels
ans_dataset_tf = dataset_to_tf(ans_dataset, with_labels=False)
```

6.3 Prediction

```
# Run predictions on the dataset
predictions = model.predict(ans_dataset_tf)

# Extract the predicted labels (logits) and convert to classes (use argmax)
predicted_labels = np.argmax(predictions.logits, axis=-1)

# Inverse transform the numerical labels to original emotion categories
predicted_emotions = label_encoder.inverse_transform(predicted_labels)

# Add the predictions to the dataframe as a new 'emotion' column
ans_df['emotion'] = predicted_emotions

print(ans_df.head())
```

顯示隱藏的輸出內容

6.4 Save CSV File

```
ans_df_subset = ans_df[['tweet_id', 'emotion']]

# Rename 'tweet_id' to 'id'
ans_df_subset = ans_df_subset.rename(columns={'tweet_id': 'id'})

ans_df_subset.to_csv('predictions.csv', index=False)
```