Kaggle competition report

Prepare the data frame with attributes

```
# Transform JSON data into a DataFrame
df = pd.DataFrame(data)
_source = df['_source'].apply(lambda x: x['tweet'])
df = pd.DataFrame({
    'tweet_id': _source.apply(lambda x: x['tweet_id']),
    'hashtags': _source.apply(lambda x: x['hashtags']),
    'text': _source.apply(lambda x: x['text']),
})
```

Preprocessing

```
# Preprocess text data
df['text'] = df['text'].str.replace('<LH>', '', regex=False).str.strip()# remove <LH>
df['text'] = df['text'].str.replace(r'@\w+', '', regex=True)# remove @
df['text'] = df['text'].str.replace(r'#\w+', '', regex=True)# remove #
df['text'] = df['text'].str.replace(r'http\S+|www.\S+', '', regex=True)# remove URL
df['text'] = df['text'].str.lower()# convert to lower case
```

Initially, there was an attempt to remove duplicate data, but this would lead to an incorrect number of data when submitting, so this process was removed.

```
# Remove duplicate rows
# df.drop_duplicates(subset=['text'], keep=False, inplace=True)
```

Data preparation for BERT

```
x_data = train_data["text"]
y_data = train_data['emotion']
label_encoder = LabelEncoder()
y_data_encoded = label_encoder.fit_transform(y_data)# Encode labels
```

```
# Split into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x_data, y_data_encoded, test_size=0.2, random_state=42)
```

```
# Tokenize text data
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_encodings = tokenizer(list(x_train), truncation=True, padding=True, max_length=128, return_tensors="pt")
val_encodings = tokenizer(list(x_val), truncation=True, padding=True, max_length=128, return_tensors="pt")
```

```python
# Build PyTorch Dataset
class EmotionDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = torch.tensor(labels)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item['labels'] = self.labels[idx]
        return item
```

Load and configure BERT model

```python
# Load pre-trained BERT model
num_labels = len(label_encoder.classes_)
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=num_labels)
```

```python
# Training parameters
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    fp16=torch.cuda.is_available()
)
```

Train 3 epochs: Training time is approximately two and a half hours

Predict on test data

```python
# Preprocess test data
batch_size = 32
inputs = tokenizer(list(test_data['text']), return_tensors="pt", padding=True, truncation=True, max_length=128)
dataset = TensorDataset(*inputs.values())
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=False)
```

Make predictions

```python
all_predictions = []
with torch.no_grad():
    for batch in dataloader:
        batch = {key: val.to(device, non_blocking=True) for key, val in zip(inputs.keys(), batch)}

        outputs = model(**batch)
        batch_predictions = torch.argmax(outputs.logits, dim=-1)

        # Collect prediction results
        all_predictions.extend(batch_predictions.cpu().numpy())

        # Clean up memory
        del batch, outputs, batch_predictions
        torch.cuda.empty_cache()
```

```
# Convert results format
all_predictions = np.array(all_predictions)
```

```
# Convert predictions to labels
y_pred_labels = label_encoder.inverse_transform(all_predictions)
```

Create submission file

```
# Create submission file
submission = pd.DataFrame({
    'id': test_data['tweet_id'],
    'emotion': y_pred_labels
})
submission.to_csv('submission.csv', index=False)
```