112065710
Ko Lih Han

1. Data Preprocessing

Datasets:

- data_identification.csv: Contains tweet_id and identification (train/test).
- emotion.csv: Contains tweet_id and emotion labels.
- tweets_DM.json: Contains raw tweet data with tweet_id and text.

```
# {'tweet': {'hashtags': ['Snapchat'], 'tweet_id': '0x376b20', 'tex

tweets_DM = pd.DataFrame([{
    'tweet_id': item['tweet']['tweet_id'],
    'text': item['tweet']['text']
} for item in tweets['_source']])

tweets_DM.head()
```

	tweet_id	text
0	0x376b20	People who post "add me on #Snapchat" must be ...
1	0x2d5350	@brianklaas As we see, Trump is dangerous to #...
2	0x28b412	Confident of your obedience, I write to you, k...
3	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>
4	0x2de201	"Trust is not the same as faith. A friend is s...

```
merged_data = tweets_DM.merge(data_id, on='tweet_id', how='left').merge(emotion, on='tweet_id', how='left')
```

After merging the data, I separate it to test and train data.

```
df_train = merged_data[merged_data.identification == 'train']
df_test = merged_data[merged_data.identification == 'test']

print(f"Training samples: {df_train.shape[0]}")
print(f"Test samples: {df_test.shape[0]}")
```

```
Training samples: 1455563
Test samples: 411972
```

```
df_train = df_train.sample(frac=1)
df_test = df_test.sample(frac=1)
```

For the first approach I used LSTM to train the dataset.

To prepare the textual data for the LSTM model, the following steps were undertaken:

- Utilized Tokenizer to convert text data into sequences of integers.
- Limited the vocabulary to the top 20,000 most frequent words to manage computational resources effectively.

```
# Tokenize and pad sequences
max_words = 20000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df_train['text'])

X_train = tokenizer.texts_to_sequences(df_train['text'])
X_train = pad_sequences(X_train, maxlen=max_len)

X_test = tokenizer.texts_to_sequences(df_test['text'])
X_test = pad_sequences(X_test, maxlen=max_len)
```

After that we also converted emotion labels into numerical format suitable for model training

```
# Encode labels
emotions = df_train['emotion'].unique()
emotion_to_index = {emotion: index for index, emotion in enumerate(emotions)}
y_train = df_train['emotion'].map(emotion_to_index)
y_train = to_categorical(y_train, num_classes=len(emotions))
```

The LSTM Architecture

```
# Build LSTM model
model = Sequential()
model.add(Embedding(max_words, 128, input_length=max_len))
model.add(LSTM(32, return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(32))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(emotions), activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Architecture:

- Embedding Layer: Converts words into 128-dimensional vectors.
- LSTM Layers: Two LSTM layers with 32 units each to capture sequential dependencies.
- Dropout Layers: Applied dropout of 0.5 to prevent overfitting.
- Dense Layers: A fully connected layer with ReLU activation followed by a softmax output layer for classification.

Using LSTM we get the result of 0.45

Now we try Transformer

For the Second approach we try to use Roberta as our model

Utilized Hugging Face's RobertaTokenizer and RobertaForSequenceClassification for efficient text encoding and classification.

```
# Initialize tokenizer and model
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
model = RobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=len(emotions))
```

```python
# Define custom dataset for PyTorch
class TweetDataset(Dataset):
    def __init__(self, texts, labels=None, tokenizer=None, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

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

    def __getitem__(self, index):
        text = str(self.texts[index])
        inputs = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )
        input_ids = inputs['input_ids'].squeeze()
        attention_mask = inputs['attention_mask'].squeeze()

        if self.labels is not None:
            label = torch.tensor(self.labels[index], dtype=torch.long)
            return {'input_ids': input_ids, 'attention_mask': attention_mask, 'label': label}
        else:
            return {'input_ids': input_ids, 'attention_mask': attention_mask}
```

Created a custom dataset class to handle tokenization and formatting of the data.

Configured training parameters and used the Trainer class to manage the training loop, evaluation, and optimization.

```python
# Update training arguments to include evaluation
training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy='epoch',  # Enable evaluation at the end of each epoch
    save_strategy='epoch',
    learning_rate=1e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=2,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    report_to = []
)

# Initialize Trainer with the validation dataset
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset  # Add the validation dataset
)



# Train the model
trainer.train()
```

After training and predict we get a public score of 0.56

The Result shows that the RoBERTa model achieved a significantly higher validation accuracy compared to the LSTM model, demonstrating superior ability to capture contextual nuances in the tweets.