Part 3

心得

在這次情緒分類的小競賽中，從RandomForest+TF-IDF到BERT的轉換讓我獲得許多洞見。傳統方法雖然簡單高效，但主要依賴詞頻統計，難以捕捉"not happy"和"very happy"這類強調上下文理解的情緒表達，但是RandomForest+TF-IDF的運行速度是bert的兩倍以上，或許在數據預處理的部分如果做得更好，傳統方法也不見得會遜於bert。

就此次試驗來看，BERT模型帶來了顯著改進，主要優勢在於：

1. 能理解詞語在不同語境下的差異，特別是處理諷刺和反話的部分，這是在社交媒體上特別常見的用語
2. 自注意力機制可以更好地理解完整的情緒表達，也能捕捉複雜的口語用法，以及前後文不對應的複雜情況，BERT模型也能很好適應

未來優化方向：

1. 針對社交數據進行更針對，處理表情符號和hashtag
2. 嘗試BERTweet等專門的社交媒體模型
3. 深入分析容易混淆的情緒類別（如anticipation和trust），並對其進行專屬的訓練或貼標
4. 增加模型可解釋性，理解決策依據

1st Try - TF-IDF + Random Forest score: 0.32768

I try this method in the formal competition, but it didn't get a good performance.However, due to the time pressure, I submited it as my result.

After the competition ended, I try another method and I although write it below.

```
import json
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_scor
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
```

Read data

```
def load_json_data(file_path):
tweets = []
with open(file_path, 'r', encoding='utf-8') as file:
for line in file:
tweet = json.loads(line)
source = tweet['_source']['tweet']
tweets.append({
'tweet_id': source['tweet_id'],
'hashtags': source['hashtags'],
'text': source['text']
})
return pd.DataFrame(tweets)
```

Data Prreprocessing

```
def preprocess_text(text):
# 移除URL
```

```
text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MUI
# 移除@mentions
text = re.sub(r'@\w+', '', text)
# 移除數字
text = re.sub(r'\d+', '', text)
# 轉換為小寫
text = text.lower()
# 移除標點符號
text = re.sub(r'[^\w\s]', '', text)
# 移除多餘空格
text = re.sub(r'\s+', ' ', text).strip()
return text
```

```
df = load_json_data('/kaggle/input/dm-2024-isa-5810-lab-2-homewo
emotion_df = pd.read_csv('/kaggle/input/dm-2024-isa-5810-lab-2-h
identification_df = pd.read_csv('/kaggle/input/dm-2024-isa-5810-
```

```
df = df.merge(identification_df, on='tweet_id', how='left')
train_df = df[df['identification'] == 'train'].merge(emotion_df,
test_df = df[df['identification'] == 'test']
```

```
train_df['processed_text'] = train_df['text'].apply(preprocess_t
test_df['processed_text'] = test_df['text'].apply(preprocess_tex
```

TF-IDF

- max_features=4000：實驗後發現這個數量能在特徵數和效率間取得平衡
- min_df=2：避免過於罕見的詞
- max_df=0.95：過濾掉嚴重的常用詞
- ngram_range=(1,3)：捕捉最多3個詞的詞組，希望可以抓住更多情緒表達方式

```
tfidf = TfidfVectorizer(
max_features=4000,  # 增加特徵數
min_df=2,           # 最小文檔頻率
max_df=0.95,        # 最大文檔頻率
stop_words=stopwords.words('english'),
ngram_range=(1, 3)  # 加入trigrams
)
```

Training Set & Testing Set

```
X = vectorizer.fit_transform(train_df['processed_text'])
y = train_df['emotion']
```

```
X_train, X_val, y_train, y_val = train_test_split(
X, y, test_size=0.25,
random_state=42,
stratify=y
)
```

Random Forest

- n_estimators=200：增加樹的數量提高穩定性
- max_depth=15：限制樹數，避免過擬合
- min_samples_split和min_samples_leaf：較小的值讓模型學習細節
- class_weight='balanced'：處理類別不平衡問題

```
rf_classifier = RandomForestClassifier(
n_estimators=200,max_depth=15,
min_samples_split=5,
min_samples_leaf=2,
max_features='sqrt',
class_weight='balanced',
n_jobs=-1,
```

```
random_state=42
)

rf_classifier.fit(X_train, y_train)
```

Prediction, Validation

暫時以f1-score作為評估指標

```
val_pred = rf_classifier.predict(X_val)
print("Validation F1-score:", f1_score(y_val, val_pred, average=
print("\nClassification Report:")
print(classification_report(y_val, val_pred))
```

```
feature_names = tfidf.get_feature_names_out()
feature_importance = pd.DataFrame({
'feature': feature_names,
'importance': rf_classifier.feature_importances_
})
feature_importance = feature_importance.sort_values('importance
print("\nTop 10 Most Important Features:")
print(feature_importance.head(10))
```

```
X_test = tfidf.transform(test_df['processed_text'])
test_predictions = rf_classifier.predict(X_test)
```

```
submission = pd.DataFrame({
'id': test_df['tweet_id'],
'emotion': test_predictions
})
```

```
submission.to_csv('/kaggle/working/submission.csv', index=False)
```

2nd try - BERT

Score : 0.46939

Preprocessing

```
import json
import pandas as pd
import numpy as np
import torch
from torch import nn
from transformers import AutoTokenizer, AutoModel, AdamW
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm
import re
```

```
def preprocess_text(text):
text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MUI
text = re.sub(r'@\w+', '', text)
text = re.sub(r'\s+', ' ', text).strip()
return text
```

Define Dataset

```
class EmotionDataset(Dataset):
def init(self, texts, labels, tokenizer, 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, idx):
    text = str(self.texts[idx])
    label = self.labels[idx] if self.labels is not None else
None

    encoding = self.tokenizer(
        text,
        add_special_tokens=True,
        max_length=self.max_len,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    return {
        'input_ids': encoding['input_ids'].flatten(),
        'attention_mask': encoding['attention_mask'].flatten
(),
        'labels': torch.tensor(label) if label is not None el
se None
    }
```

Modeling

```
class EmotionClassifier(nn.Module):
def init(self, n_classes):
super().init()
self.bert = AutoModel.from_pretrained('distilbert-base-uncased')
self.drop = nn.Dropout(0.3)
self.fc = nn.Linear(self.bert.config.hidden_size, n_classes)
```

```
def forward(self, input_ids, attention_mask):
    outputs = self.bert(
        input_ids=input_ids,
        attention_mask=attention_mask
    )
    pooled_output = outputs[0][:, 0]  # 使用[CLS]输出
    output = self.drop(pooled_output)
    return self.fc(output)
```

```
def load_json_data(file_path):
tweets = []
with open(file_path, 'r', encoding='utf-8') as file:
for line in file:
tweet = json.loads(line)
source = tweet['_source']['tweet']
tweets.append({
'tweet_id': source['tweet_id'],
'text': source['text']
})
return pd.DataFrame(tweets)
```

```
def train_model(model, train_loader, val_loader, criterion, opti
best_val_loss = float('inf')
```

```
for epoch in range(epochs):
    model.train()
    total_train_loss = 0

    for batch in tqdm(train_loader, desc=f'Training Epoch {ep
och+1}'):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
```

```
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        total_train_loss += loss.item()

    model.eval()
    total_val_loss = 0
    val_preds = []
    val_labels = []

    with torch.no_grad():
        for batch in val_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(devic
e)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask)
            loss = criterion(outputs, labels)
            total_val_loss += loss.item()

            val_preds.extend(outputs.argmax(dim=1).cpu().nump
y())
            val_labels.extend(labels.cpu().numpy())

    avg_train_loss = total_train_loss / len(train_loader)
    avg_val_loss = total_val_loss / len(val_loader)

    print(f'Epoch {epoch+1}:')
    print(f'Average Training Loss: {avg_train_loss:.4f}')
    print(f'Average Validation Loss: {avg_val_loss:.4f}')
```

```
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), 'best_model.pt')
```

Main

```
device = torch.device('cuda' if torch.cuda.is_available() else
```

```
#讀取資料
df = load_json_data('/kaggle/input/dm-2024-isa-5810-lab-2-homew
emotion_df = pd.read_csv('/kaggle/input/dm-2024-isa-5810-lab-2-
identification_df = pd.read_csv('/kaggle/input/dm-2024-isa-5810
```

```
df = df.merge(identification_df, on='tweet_id', how='left')
train_df = df[df['identification'] == 'train'].merge(emotion_df,
test_df = df[df['identification'] == 'test']
```

```
#預處理
train_df['processed_text'] = train_df['text'].apply(preprocess_t
test_df['processed_text'] = test_df['text'].apply(preprocess_tex
```

```
# 編碼
label_encoder = LabelEncoder()
train_df['encoded_emotion'] = label_encoder.fit_transform(train_
```

```
#初始化tokenizer
tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncas
```

```
#分割訓練和驗證集
train_texts, val_texts, train_labels, val_labels = train_test_sp
train_df['processed_text'].values,
```

```
train_df['encoded_emotion'].values,
test_size=0.2,
random_state=42,
stratify=train_df['encoded_emotion']
)
```

```
train_dataset = EmotionDataset(train_texts, train_labels, token
val_dataset = EmotionDataset(val_texts, val_labels, tokenizer)
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=
val_loader = DataLoader(val_dataset, batch_size=32)
```

```
#模型
model = EmotionClassifier(n_classes=len(label_encoder.classes_))
model = model.to(device)
optimizer = AdamW(model.parameters(), lr=2e-5)
criterion = nn.CrossEntropyLoss()
```

```
#訓練
train_model(model, train_loader, val_loader, criterion, optimize
```

```
test_dataset = EmotionDataset(
test_df['processed_text'].values,
[0] * len(test_df),
tokenizer
)
test_loader = DataLoader(test_dataset, batch_size=32)
```

```
#預測
model.eval()
predictions = []
with torch.no_grad():
```

```
for batch in test_loader:
input_ids = batch['input_ids'].to(device)
attention_mask = batch['attention_mask'].to(device)
outputs = model(input_ids, attention_mask)
preds = outputs.argmax(dim=1).cpu().numpy()
predictions.extend(preds)
```

```
#轉換預測結果
predicted_emotions = label_encoder.inverse_transform(predictions
```

```
#提交文件
submission = pd.DataFrame({
'id': test_df['tweet_id'],
'emotion': predicted_emotions
})
submission.to_csv('/kaggle/working/submission.csv', index=False)
```