**鄭凱文**

**科管所博一**

**學號413708002**

**情緒識別競賽開發模型的工作報告**

本報告詳細記錄了參加Kaggle情緒識別競賽的過程，包括數據預處理、特徵工程、模型訓練與優化，以及最終的預測生成。在這個過程中，遇到了多個挑戰，並採取了相應的解決措施，報告最後對未來的工作方向提出了展望。(**程式及說明附在最後**)

1.數據預處理

在數據預處理階段，首先進行了數據清洗，特別是缺失值處理。通過檢查`data\_identification.csv`和`emotion.csv`，發現兩個數據集中的`tweet\_id`、`identification`和`emotion`字段都沒有缺失值，這為後續的數據處理打下了良好的基礎。接下來，根據`identification`列將數據分為訓練集和測試集，並檢查了兩個集合中`tweet\_id`的重疊情況。幸運的是，沒有發現任何重疊，這確保了模型可以在獨立的數據集上進行訓練和測試。在合併資料集時，使用`tweet\_id`將`train\_data`和`emotion.csv`合併，確保每個推文都有對應的情緒標籤。這一步驟中，沒有遇到缺失值問題，這使得數據集的完整性得到了保障。最後，從JSON文件中讀取了推文文本，並與合併後的資料集進行關聯。這一過程中，遇到了JSON解析錯誤，但通過打印錯誤資訊和對應的行，及時發現並解決了這些問題。

2.特徵工程

在特徵工程階段，首先進行了文本特徵提取，包括詞幹提取。這一過程中，使用了NLTK庫中的PorterStemmer，將文本中的詞語轉換為其基本形式，以減少詞語的變體形式。接著，提取了時間特徵、使用者行為特徵和推文長度。這些特徵的提取涉及到對時間戳的解析和對文本的分析，過程中遇到了一些格式不一致的問題，但通過調整解析函數，成功地解決了這些問題。在特徵選擇階段，使用梯度提升樹模型來評估特徵的重要性，並選擇了平均重要性以上的特徵。這一過程中，發現了一些不太重要的特徵，這有助於減少模型的複雜度。

3.模型訓練與評估

在模型訓練階段，選擇了樸素貝葉斯模型作為基礎模型。對文本特徵進行了TF-IDF處理，並將其與其他數值特徵合併。通過3折交叉驗證，得到了平均分數0.4952(調整參數後為0.5245)，這為提供了一個參考基線。隨後，進行了模型優化，通過網格搜索來尋找最佳的超參數。這一步驟顯著提高了模型的F1 score，達到了0.49562(調整參數後為0.5245)。在這一過程中，遇到了計算資源的限制，但通過優化代碼和使用更高效的算法，成功地完成了網格搜索。

4.未來工作展望

對於未來的工作，計劃探索更先進的機器學習技術，如深度學習和集成學習方法，以提高模型的準確性和泛化能力。此外，還將深入研究特徵工程，以發現更多有信息量的特徵。最後，將考慮使用更大的數據集進行訓練，以進一步提高模型的性能。總之，參加Kaggle情緒識別競賽是一個充滿挑戰和學習的過程。雖然模型的F1 score為0.49562(調整參數後為0.5245)，但這個過程讓對數據預處理、特徵工程、模型選擇和優化有了更深的理解。期待未來能夠取得更好的成績。

程式及說明

1.數據預處理

1-1數據清洗-缺失值處理

import pandas as pd

import json

# 定義檔路徑

data\_identification\_path = r'F:/111/DM2024-Lab2-Kaggle-CHENG Hoi Man/data\_identification.csv'

emotion\_path = r'F:/111/DM2024-Lab2-Kaggle-CHENG Hoi Man/emotion.csv'

# 讀取數據

data\_identification = pd.read\_csv(data\_identification\_path)

emotion = pd.read\_csv(emotion\_path)

# 1-1 資料清洗 - 缺失值處理

# 檢查data\_identification.csv中的缺失值

missing\_data\_identification = data\_identification.isnull().sum()

print("缺失值統計 - data\_identification.csv:")

print(missing\_data\_identification)

# 檢查emotion.csv中的缺失值

missing\_emotion = emotion.isnull().sum()

print("缺失值統計 - emotion.csv:")

print(missing\_emotion)

**1-2資料劃分-訓練集和測試集**

# 1-2 資料劃分 - 訓練集和測試集

# 根據identification列將資料分為訓練集和測試集

train\_data = data\_identification[data\_identification['identification'] == 'train'].copy()

test\_data = data\_identification[data\_identification['identification'] == 'test'].copy()

# 檢查訓練集和測試集中的tweet\_id是否有重疊

重疊tweet\_id = train\_data['tweet\_id'].isin(test\_data['tweet\_id']).sum()

print("訓練集和測試集中重疊的tweet\_id數量:", 重疊tweet\_id)

# 如果有重疊，列印出重疊的tweet\_id

if 重疊tweet\_id > 0:

    overlapping\_ids = train\_data[train\_data['tweet\_id'].isin(test\_data['tweet\_id'])]['tweet\_id'].unique()

    print("重疊的tweet\_id列表:", overlapping\_ids)

else:

    print("訓練集和測試集中的tweet\_id沒有重疊。")

**1-3合併資料集**

# 1-3 合併資料集

# 使用tweet\_id將train\_data和emotion.csv合併，確保每個推文都有對應的情緒標籤

merged\_train\_data = pd.merge(train\_data, emotion, on='tweet\_id', how='left')

# 檢查合併後的缺失值

missing\_values = merged\_train\_data.isnull().sum()

print("合併後的缺失值統計:")

print(missing\_values)

# 查看合併後的資料集前幾行

print(merged\_train\_data.head())

**1-4文本資料讀取**

import os

# 定義JSON檔路徑

tweets\_dm\_path = r'F:/111/DM2024-Lab2-Kaggle-CHENG Hoi Man/tweets\_DM.json'

# 初始化一個空字典來存儲推文資料

tweets\_data = {}

# 讀取JSON文件

with open(tweets\_dm\_path, 'r', encoding='utf-8') as file:

    for line in file:

        try:

            # 將每行解析為JSON對象

            tweet = json.loads(line)

            # 提取tweet\_id和推文文本

            tweet\_id = tweet['\_source']['tweet']['tweet\_id']

            text = tweet['\_source']['tweet']['text']

            # 將推文文本存儲在字典中

            tweets\_data[tweet\_id] = text

        except json.JSONDecodeError as e:

            # 如果解析出錯，列印錯誤資訊和對應的行

            print(f"Error parsing line: {line}")

            print(e)

# 將推文文本資料轉換為DataFrame

tweets\_df = pd.DataFrame(list(tweets\_data.items()), columns=['tweet\_id', 'text'])

# 將推文文本資料與合併後的資料集關聯，使用tweet\_id作為關聯鍵

final\_data = pd.merge(merged\_train\_data, tweets\_df, on='tweet\_id', how='left')

# 檢查最終關聯後的資料集前幾行

print(final\_data.head())

# 將推文文本資料與合併後的資料集關聯，使用tweet\_id作為關聯鍵

final\_test\_data = pd.merge(test\_data, tweets\_df, on='tweet\_id', how='left')

# 檢查最終關聯後的資料集前幾行

print(final\_test\_data.head())

**2.特徵工程**

**2-1文本特徵(詞幹提取)**

import nltk

from nltk.stem import PorterStemmer

# 初始化詞幹提取器

stemmer = PorterStemmer()

# 定義詞幹提取函數

def stemming(text):

    words = text.split()

    stemmed\_words = [stemmer.stem(word) for word in words]

    return ' '.join(stemmed\_words)

# 應用文本特徵提取函數到推文文本列

final\_data['text\_stemmed'] = final\_data['text'].apply(stemming)

# 檢查最終關聯後的資料集前幾行

print(final\_data.head())

# 應用文本特徵提取函數到推文文本列

final\_test\_data['text\_stemmed'] = final\_test\_data['text'].apply(stemming)

# 檢查最終關聯後的資料集前幾行

print(final\_test\_data.head())

**2-2中繼資料特徵(時間特徵、使用者行為特徵、tweet長度)**

import datetime

# 定義提取時間特徵的函數

def extract\_time\_features(date\_str):

    date = datetime.datetime.strptime(date\_str, '%Y-%m-%d %H:%M:%S')

    return {

        'year': date.year,

        'month': date.month,

        'day': date.day,

        'weekday': date.weekday(),  # 0 is Monday, 6 is Sunday

        'hour': date.hour

    }

# 定義提取使用者行為特徵的函數

def extract\_user\_behavior\_features(text):

    hashtags = text.count('#')

    retweet = text.count('RT ')

    reply = text.count('@')

    return {

        'hashtags\_count': hashtags,

        'retweets\_count': retweet,

        'replies\_count': reply

    }

# 定義提取tweet長度特徵的函數

def tweet\_length(text):

    return len(text)

# 讀取tweets\_DM.json檔並提取特徵

tweets\_features = []

with open(tweets\_dm\_path, 'r', encoding='utf-8') as file:

    for line in file:

        try:

            tweet = json.loads(line)

            tweet\_id = tweet['\_source']['tweet']['tweet\_id']

            crawl\_date = tweet['\_crawldate']

            # 提取時間特徵

            time\_features = extract\_time\_features(crawl\_date)

            # 將特徵組合在一起

            tweets\_features.append({

                'tweet\_id': tweet\_id,

                \*\*time\_features

            })

        except json.JSONDecodeError as e:

            print(f"Error parsing line: {line}")

            print(e)

# 將特徵轉換為DataFrame

tweets\_time\_features\_df = pd.DataFrame(tweets\_features)

# 合併時間特徵到tweets\_df

tweets\_df = pd.merge(tweets\_df, tweets\_time\_features\_df, on='tweet\_id', how='left')

# 提取使用者行為特徵和tweet長度特徵

tweets\_df['user\_behavior\_features'] = tweets\_df['text'].apply(extract\_user\_behavior\_features)

tweets\_df['tweet\_length'] = tweets\_df['text'].apply(tweet\_length)

# 展開使用者行為特徵

user\_behavior\_columns = ['hashtags\_count', 'retweets\_count', 'replies\_count']

tweets\_df[user\_behavior\_columns] = pd.DataFrame(tweets\_df['user\_behavior\_features'].tolist(), index=tweets\_df.index)

# 刪除臨時列

tweets\_df.drop(['user\_behavior\_features'], axis=1, inplace=True)

# 合併最終資料集

final\_data = pd.merge(final\_data, tweets\_df, on='tweet\_id', how='left')

# 檢查最終資料集的前幾行

print(final\_data.head())

# 合併最終資料集

final\_test\_data = pd.merge(final\_test\_data, tweets\_df, on='tweet\_id', how='left')

# 檢查最終資料集的前幾行

print(final\_test\_data.head())

**2-3特徵選擇(中繼資料特徵)**

from sklearn.ensemble import GradientBoostingClassifier

# 選擇特徵和目標變數

X = final\_data.drop(['tweet\_id', 'identification', 'emotion', 'text\_x', 'text\_stemmed', 'text\_y'], axis=1)

y = final\_data['emotion']

# 初始化梯度提升樹模型

gb = GradientBoostingClassifier(n\_estimators=10, random\_state=42)

# 訓練模型

gb.fit(X, y)

# 使用模型評估特徵重要性

importances = gb.feature\_importances\_

# 將特徵重要性與特徵名稱結合起來

feature\_importances = pd.Series(importances, index=X.columns).sort\_values(ascending=False)

# 選擇平均以上的特徵

threshold = importances.mean()

selected\_features = feature\_importances[feature\_importances > threshold].index.tolist()

# 創建一個新的 DataFrame，包含選中的特徵和其他非特徵列

selected\_data = final\_data[['tweet\_id', 'identification', 'emotion'] + selected\_features]

# 列印最終選擇的特徵和目標變數的前幾行

print(selected\_data.head())

# 創建一個新的 DataFrame，包含選中的特徵和其他非特徵列

selected\_test\_data = final\_test\_data[['tweet\_id', 'identification'] + selected\_features]

# 列印最終選擇的特徵和目標變數的前幾行

print(selected\_test\_data.head())

**2-4特徵合併**

# 從第二個程式中獲取選擇的特徵清單

selected\_features = feature\_importances[feature\_importances > threshold].index.tolist()

# 將text\_stemmed添加到selected\_features列表中

selected\_features.append('text\_stemmed')

# 創建一個新的DataFrame，包含選中的特徵和其他非特徵列

selected\_data = final\_data[['tweet\_id', 'identification', 'emotion'] + selected\_features]

# 列印最終選擇的特徵和目標變數的前幾行

print(selected\_data.head())

# 創建一個新的DataFrame，包含選中的特徵和其他非特徵列

selected\_test\_data = final\_test\_data[['tweet\_id', 'identification'] + selected\_features]

# 列印最終選擇的特徵和目標變數的前幾行

print(selected\_test\_data.head())

**3.模型訓練與評估**

**3-1模型訓練(樸素貝葉斯模型)**

import numpy as np

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import cross\_val\_score

from sklearn.feature\_extraction.text import TfidfVectorizer

from joblib import dump

from scipy.sparse import hstack

# 創建樸素貝葉斯模型

nb\_classifier = MultinomialNB()

# 準備數據

X = selected\_data.drop(['tweet\_id', 'identification', 'emotion'], axis=1)

y = selected\_data['emotion']

# 準備數據

X\_sample = selected\_test\_data.drop(['tweet\_id', 'identification'], axis=1)

# 文本特徵處理，優化詞彙量

vectorizer = TfidfVectorizer(max\_features=5000)  # 假設我們只保留最重要的5000個詞彙

X\_text = vectorizer.fit\_transform(X['text\_stemmed'])

X\_sample\_test = vectorizer.fit\_transform(X\_sample['text\_stemmed'])

# 將其他數值特徵與文本特徵合併

X\_numeric = X.drop('text\_stemmed', axis=1).values

X\_combined = hstack((X\_numeric, X\_text))  # 使用疏鬆陣列合併

X\_sample\_numeric = X\_sample.drop('text\_stemmed', axis=1).values

X\_sample\_combined = hstack((X\_sample\_numeric, X\_sample\_test))  # 使用疏鬆陣列合併

# 交叉驗證

scores = cross\_val\_score(nb\_classifier, X\_combined, y, cv=3)  # 使用3折交叉驗證

print(f"Cross-validation scores: {scores}")

print(f"Average score: {scores.mean()}")

# 訓練模型

nb\_classifier.fit(X\_combined, y)

# 保存模型

dump(nb\_classifier, 'naive\_bayes\_model.joblib')

**3-2模型優化及評估（網格搜索）**

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import GridSearchCV

from joblib import load

from sklearn.metrics import make\_scorer, f1\_score

# 載入已經訓練好的模型

nb\_classifier = load('naive\_bayes\_model.joblib')

# 定義超參數網格

param\_grid = {

    'alpha': [0.5, 1.0, 1.5, 2.0],  # 平滑參數的不同取值

    'fit\_prior': [True, False]  # 是否學習類別的先驅概率

}

# 創建F1分數評分器

f1\_scorer = make\_scorer(f1\_score, average='micro')

# 創建GridSearchCV對象，並設置scoring為包含'accuracy'和'f1'的字典，同時指定refit参数

grid\_search = GridSearchCV(estimator=nb\_classifier, param\_grid=param\_grid, cv=3,

                           scoring={'accuracy': 'accuracy', 'f1': f1\_scorer},

                           refit='f1',  # 指定refit参数为'f1'

                           verbose=2, n\_jobs=-1)

# 執行網格搜索

grid\_search.fit(X\_combined, y)

# 輸出最佳參數和得分

print("Best parameters found: ", grid\_search.best\_params\_)

print("Best cross-validation scores: ")

print("Accuracy: ", grid\_search.cv\_results\_['mean\_test\_accuracy'][grid\_search.best\_index\_])

print("F1 score: ", grid\_search.cv\_results\_['mean\_test\_f1'][grid\_search.best\_index\_])

# 使用最佳參數訓練模型

best\_nb\_classifier = grid\_search.best\_estimator\_

# 保存模型

dump(best\_nb\_classifier, 'naive\_bayes\_optimized\_model.joblib')

**4.生成預測**

import joblib

import pandas as pd

# 載入訓練好的模型

optimized\_nb\_classifier = joblib.load('naive\_bayes\_optimized\_model.joblib')

# 使用模型進行預測

predicted\_emotions = optimized\_nb\_classifier.predict(X\_sample\_combined)

# 創建提交文件的DataFrame

submission\_data = pd.DataFrame({

    'tweet\_id': selected\_test\_data['tweet\_id'],

    'emotion': predicted\_emotions

})

# 保存為CSV檔

submission\_data.to\_csv('sampleSubmission.csv', index=False)