2024 Data Mining Lab 2 Kaggle Competition Report

NTHU 112061591 賴彥霖

Brief Introduction

To introduce this project in short, I use Word2Vec to generate features and use Random Forest classifier to predict result.

1. Data Load

First of all, os.walk function is used to go through the file inside Kaggle input.

```
import json
import numpy as np
import pandas as pd
import os

# Check what file is inside and their path
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/dm-2024-isa-5810-lab-2-homework/tweets_DM.json
/kaggle/input/dm-2024-isa-5810-lab-2-homework/sampleSubmission.csv
/kaggle/input/dm-2024-isa-5810-lab-2-homework/data_identification.csv
/kaggle/input/dm-2024-isa-5810-lab-2-homework/emotion.csv
```

Data was written into the ‘data’ list for the following uilization.

```
# store json file to list: data
data = []
with open('/kaggle/input/dm-2024-isa-5810-lab-2-homework/tweets_DM.json', 'r') as f:
    for line in f:
        data.append(json.loads(line))
```

2. Data Observation

```
# Check first data format
print(f'num of data: {len(data)}')
data[5:10]
```

```
[{'_score': 120,
  '_index': 'hashtag_tweets',
  '_source': {'tweet': {'hashtags': ['authentic', 'LaughOutLoud'],
    'tweet_id': '0x1d755c',
    'text': '@RISKshow @TheKevinAllison Thx for the BEST TIME tonight. What stories! Hear
tbreakingly <LH> #authentic #LaughOutLoud good!!'}},
  '_crawldate': '2015-06-11 04:44:05',
  '_type': 'tweets'},
 {'_score': 1021,
  '_index': 'hashtag_tweets',
  '_source': {'tweet': {'hashtags': [],
    'tweet_id': '0x2c91a8',
    'text': 'Still waiting on those supplies Liscus. <LH>'}},
  '_crawldate': '2015-08-18 02:30:07',
  '_type': 'tweets'},
 {'_score': 481,
  '_index': 'hashtag_tweets',
  '_source': {'tweet': {'hashtags': [],
    'tweet_id': '0x368e95',
    'text': 'Love knows no gender. 😢😭 <LH>'}},
```


To preprocess and clean up the data, checking the raw data is a necessary step. In the raw data, we can observe main information are packaged in the 'tweet' item of '_source' dictionary, ['hashtags', 'tweet_ids', 'text'] are package as input. I observed that not every dictionary have hashtags, None value problem may occur if I iterate this list, so I plan to append the hashtags in the end of 'text'. Moreover, emoji, # hashtag, @ hashtag and <LH> mark showed up in many text to be removed.

3. Preparing Data

The code to easily append 'hashtags' to the end of text is demonstrated as follows:

```
ids = []
texts = []

for i in range(len(data)):
    '''
    There are three things in the 'tweet' dictionary:
    'tweet'
        |-- 'hashtags'
        |-- 'tweet_id'
        |-- 'text'
    '''

    try:
        tweet_id = data[i]['_source']['tweet']['tweet_id']
        tweet_hashtag = data[i]['_source']['tweet']['hashtags']
        tweet_text = data[i]['_source']['tweet']['text']

        if not tweet_hashtag:
            combine_text = tweet_text
        else:
            combined_text = tweet_text + ' ' + ' '.join(tweet_hashtag)
        ids.append(tweet_id)
        texts.append(combined_text)
    except KeyError:
        continue
    except Exception as e:
        continue
```

4. Preprocessing

After understanding the raw data, I listed 4 target preprocessing tasks to be implemented.

1. # hashtag and @ hashtag inside the text should be removed because these tags mostly occurred for tagging friends' name to view this article. People's name are not essential information for the emotion detection.
2. <LH> mark may implied the end of the sentence, which can be removed.
3. There are many repeated sentences inside the data, which can be removed.
4. Emoji are not able to be detected, which should be transformed into text descriptions.

```
# Preprocessing
'''
Several things observered from raw data:
1. #hashtags and @hashtag frequently occur in the text
    ,which could be removed due to no influence on emotion.
2. Always have <LH> in the end to be remove.
3. Repeat sentence inside.
4. emoji need to be replace by word.

'''
import emoji
import re
import string

def preprocessing_text(text):
    preprocessed_texts = []
    for text in texts:
        text = re.sub(r"#\w+", "", text) # 移除 #hashtags
        text = re.sub(r"@\w+", "", text) # 移除 @hashtags
        text = re.sub(r"<LH>", "", text) # 移除 <LH>
        text = text.translate(str.maketrans("", "", string.punctuation)) # 移除 標點符號
        text = re.sub(r":(.*?):", r"\1", text) # 移除 emoji轉成中文後的冒號

        # 替換表情符號為文字描述
        text = emoji.demojize(text)

        # 移除多餘的空格
        text = text.strip()

        preprocessed_texts.append(text)

    return preprocessed_texts

after_preprocess_texts = preprocessing_text(texts)
```

To pack the id, text and identification type together, I generated a new data frame as follows:

```
data_dict = {'tweet_id': ids,
             'text':after_preprocess_texts,
           }
data_dict_df = pd.DataFrame(data_dict)

# 讀取 identification.csv
data_identification = pd.read_csv('/kaggle/input/dm-2024-isa-5810-lab-2-homework/data_identificat
ion.csv')

#

# 合併 dataframe
df = pd.merge(data_identification,   # 原始 DataFrame
                data_dict_df,         # 包含 tweet_id 和 text  的 DataFrame
                on='tweet_id',        # 以 tweet_id 作為合併的鍵
                how='right'           # 使用 left join 保留 data_identification
                )

train_df = df[df['identification'] == 'train']
test_df = df[df['identification'] == 'test']
print(len(train_df))
print(len(test_df))
```

We can image that the training data for the model is the text and hashtag, the label is emotion. To apply classifier, we need to use numerical data as label, label encoding is applied to fulfill this purpose.

```
label = pd.read_csv('/kaggle/input/dm-2024-isa-5810-lab-2-homework/emotion.csv')

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
label_encoded = label_encoder.fit_transform(label['emotion'])
# 可以將編碼結果加到原本的 DataFrame
label['encoded_emotion'] = label_encoded

new_train_df = train_df.merge(label, on='tweet_id', how='left')
new_train_df.head()
```

Finally, the clean data frame is generated as follows for the following training steps.

	tweet_id	identification	text	emotion	encoded_emotion
1170758	0x22c127	train	Everyone in the world can see the same full mo...	joy	4
424273	0x371130	train	I dont want the Lord to visit me I want Him to...	joy	4
485387	0x28e4ea	train	with standing up for issues and getting treat...	anger	0
1404288	0x2c3f57	train	sexy goddess Wednesday Wishing you have a won...	joy	4
593379	0x31f7ea	train	The second floor of the Marriott is basically ...	joy	4

5. Use Word2Vec to transform feature

To analyze the Twitter data, I choose to use the Word2Vec model pretrained on Twitter dataset, due to the computational resources limit, I only can choose ‘glove-twitter-100’ to speed up.

```
# Load Word2Vec pre-trained model
import gensim.downloader
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Cause we using twitter dataset, I use a pretrained model for twitter
MODEL_NAME = 'glove-twitter-100'
model = gensim.downloader.load(MODEL_NAME)
print("The Gensim model loaded successfully!")
```

To simplify generate the word vector after tokenization by gensim tools.

```
# 將文本轉換為向量
def text_to_vector(text, model):
    """
    將文本轉換為詞向量的平均值。
    """
    words = text.split()  # 將文本拆分成單詞
    word_vectors = [model[word] for word in words if word in model]  # 獲取每個單詞的向量
    if len(word_vectors) == 0:
        return np.zeros(model.vector_size)  # 如果沒有任何單詞匹配，返回全零向量
    return np.mean(word_vectors, axis=0)
```

Stratified method was used for data splitting process to solve label imbalanced problem.

```
# 劃分訓練集和測試集 (stratify 保持情緒標籤分佈一致)
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train_data,
                                                  y_train_data,
                                                  test_size=0.2,
                                                  random_state=42,
                                                  stratify=y_train_data)

# 檢查分佈
print(f"Training data size: {len(X_train)}")
print(f"Validation data size: {len(X_val)}")
```

```
Training data size: 36340
Validation data size: 9086
```

The labels was transformed into digit number already so that we just simply get the values.

```
# 分別轉換成向量
X_train = np.array([text_to_vector(text, model) for text in X_train])
X_val = np.array([text_to_vector(text, model) for text in X_val])
y_train = y_train.values
y_val = y_val.values
```

6. Build classifier: Linear SVM, Random Forest

Dimension reduction was initially performed to increase the training speed on SVM model, however, SVM could not afford that much data if using linear phase, the speed would be pretty slow, so I changed to use Linear SVM but failed as well, so finally I use Random Forest to speed up the training phase.

6. PCA dimension reduction (need for SVM, no need for Random Forest)

```
# from sklearn.decomposition import PCA

# # 降維至 100 維
# pca = PCA(n_components=100)
# X_train_reduced = pca.fit_transform(X_train)
# X_test_reduced = pca.transform(X_val)
```

```
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

# # 資料量太大時，建議使用LinearSVC
# # SVC的kernel=linear時，使用了類似於 libsvm 庫的二次規模複雜度，當訓練樣本數量很大時，計算複雜度會大幅度增加
# # 訓練 SVM 模型
# svm = LinearSVC(random_state=42, max_iter=10000)
# svm.fit(X_train, train_data['label'])
# # 預測測試集
# predictions = svm.predict(X_test)


# 使用 Random Forest以加快訓練速度
rf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)  # 使用所有可用的 CPU
rf.fit(X_train, y_train)

# 預測與評估
predictions = rf.predict(X_val)
f1 = f1_score(y_val, predictions, average='macro')

print(f'Mean F1-score: {f1}')
```

Testing process is pretty simple, just remember to inverse decode the label encoding to get emotion.

```
X_test_data = test_df['text']
X_test = np.array([text_to_vector(text, model) for text in X_test_data])
test_predictions = rf.predict(X_test)

# 生成提交結果
submission = pd.DataFrame({'id': test_df['tweet_id'],
                           'emotion': label_encoder.inverse_transform(test_predictions)
                           })

# 保存結果為 CSV 文件
submission.to_csv('/kaggle/working/submission.csv', index=False)

print('Submission saved as submission.csv')
```

```
submission.head()
```

	id	emotion
2	0x28b412	joy
4	0x2de201	joy
9	0x218443	joy
30	0x2939d5	joy
33	0x26289a	joy

Submission history