Student Information

Name:唐文蔚

Student ID:F74102022

GitHub ID:Tang-Webber

Kaggle name:weiweiwei23

Kaggle private scoreboard snapshot:

snapshot

snapshot2

Instructions

1. First: **This part is worth 30% of your grade.** Do the **take home exercises** in the DM2024-Lab2-master Repo. You may need to copy some cells from the Lab notebook to this notebook.

2. Second: **This part is worth 30% of your grade.** Participate in the in-class Kaggle Competition regarding Emotion Recognition on Twitter by this link: https://www.kaggle.com/competitions/dm-2024-isa-5810-lab-2-homework. The scoring will be given according to your place in the Private Leaderboard ranking:

    - **Bottom 40%**: Get 20% of the 30% available for this section.

    - **Top 41% - 100%**: Get (0.6N + 1 - x) / (0.6N) * 10 + 20 points, where N is the total number of participants, and x is your rank. (ie. If there are 100 participants and you rank 3rd your score will be (0.6 * 100 + 1 - 3) / (0.6 * 100) * 10 + 20 = 29.67% out of 30%.) Submit your last submission **BEFORE the deadline (Nov. 26th, 11:59 pm, Tuesday)**. Make sure to take a screenshot of your position at the end of the competition and store it as '"pic0.png'" under the **img** folder of this repository and rerun the cell **Student Information**.

3. Third: **This part is worth 30% of your grade.** A report of your work developing the model for the competition (You can use code and comment on it). This report should include what your preprocessing steps, the feature engineering steps and an explanation of your model. You can also mention different things you tried and insights you gained.

4. Fourth: **This part is worth 10% of your grade.** It's hard for us to follow if your code is messy :'(, so please **tidy up your notebook**.

Upload your files to your repository then submit the link to it on the corresponding e-learn assignment.

Make sure to commit and save your changes to your repository **BEFORE the deadline (Nov. 26th, 11:59 pm, Tuesday)**.

In [2]:
```
### Begin Assignment Here
```

Step 1. Import Library and Dataset

In [2]:
```
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer
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_score

import os
import json
import re
import emoji
import pandas as pd

import nltk
import pickle
from tqdm import tqdm
```

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [3]:

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

# file paths
tweets_file_path = 'tweets_DM.json'
data_identification_path = "data_identification.csv"
emotion_labels_path = "emotion.csv"

# Read data
tweets_data = []
with open(tweets_file_path, 'r') as file:
    for line in file:
        tweets_data.append(json.loads(line))

split_mapping = pd.read_csv(data_identification_path, index_col="tweet_id").to_dict()["identification"]
emotion_mapping = pd.read_csv(emotion_labels_path, index_col="tweet_id").to_dict()["emotion"]

# Generate DataFrame
tweets_dataset = (
    {
        "id": tweet["_source"]["tweet"]["tweet_id"],
        "hashtags": tweet["_source"]["tweet"]["hashtags"],
        "content": tweet["_source"]["tweet"]["text"],
        "emotion_label": emotion_mapping.get(tweet["_source"]["tweet"]["tweet_id"], np.nan),
        "data_split": split_mapping.get(tweet["_source"]["tweet"]["tweet_id"], np.nan),
    }
    for tweet in tweets_data
)
df = pd.DataFrame(tweets_dataset)

# View the first few rows
df.head()
```

Out [3]:

	id	hashtags	content	emotion_label	data_split
0	0x376b20	[Snapchat]	People who post "add me on #Snapchat" must be ...	anticipation	train
1	0x2d5350	[freepress, TrumpLegacy, CNN]	@brianklaas As we see, Trump is dangerous to #...	sadness	train
2	0x28b412	[bibleverse]	Confident of your obedience, I write to you, k...	NaN	test
3	0x1cd5b0	[]	Now ISSA is stalking Tasha 😂😂😂 <LH>	fear	train
4	0x2de201	[]	"Trust is not the same as faith. A friend is s...	NaN	test

Step 2. Preprocessing

From the above few rows of dataframe, we can find that there are some emojis (ex. 😂) and tags (ex. <LH>) in the content column.

So, we transform some common emojis into words and delete other emojis and tags.

In [4]:

```
# Create a dictionary of emojis and their corresponding labels
emoji_dict = {
    '😂': '[joy]',
    '😍': '[love]',
    '😭': '[cry]',
    '😊': '[happy]',
    '😘': '[kiss]',
    '😩': '[weary]',
    '🤔': '[think]',
    '🙄': '[annoyed]',
    '😁': '[happy]',
    '❤️': '[heart]',
    '💕': '[love]',
    '🔥': '[fire]',
    '💯': '[perfect]',
    '🙌': '[cheer]',
    '👍': '[nice]',
    '🙏🏾': '[pray]',
    '🙏': '[pray]',
}
```

```python
def clean_tweet(text, emoji_dict):
    # Replace emojis
    for emj, keyword in emoji_dict.items():
        text = text.replace(emj, keyword)
    # Remove remaining emojis
    text = emoji.replace_emoji(text, replace='')
    # Remove tags
    text = re.sub(r'<LH>', '', text)
    text = text.strip()
    return text

df['content'] = df['content'].apply(lambda x: clean_tweet(x, emoji_dict))
```

Step 3. Split data into train & test

In [5]:
```python
df_train = df[df['data_split'] == 'train'].drop(columns="data_split")
df_test = df[df['data_split'] == 'test'].drop(columns="data_split")
df_train.drop_duplicates(subset=['content'], keep=False, inplace=True)
```

In [6]:
```python
print("Total examples: ", len(df_train))
df_train.groupby("emotion_label").size()
```

```
Total examples:  1439344
```

Out[6]:
```
emotion_label
anger            39545
anticipation    247839
disgust         138811
fear             63326
joy             509310
sadness         191558
surprise         45402
trust           203553
dtype: int64
```

(Extra) Down-Sampling

Since the data is unbalnced, I try to down-sample to make all categories to the same size. But after experiments, it seems that ubalanced data doesn't matter

In [7]:
```
"""
# Down-Sample for unbalanced data
from sklearn.utils import resample
label_counts = df_train.groupby("emotion_label").size()

min_samples = df_train["emotion_label"].value_counts().min()

balanced_train_df = (
    df_train.groupby('emotion_label', group_keys=False)
    .apply(lambda x: x.sample(n=min_samples, random_state=42))
)

print(balanced_train_df['emotion_label'].value_counts())
df_train = balanced_train_df
"""
```

```
emotion_label
anger           39545
anticipation    39545
disgust         39545
fear            39545
joy             39545
sadness         39545
surprise        39545
trust           39545
Name: count, dtype: int64
```

```
/tmp/ipykernel_2700114/245116476.py:8: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operat ion. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after gro upby to silence this warning.
  df_train.groupby('emotion_label', group_keys=False)
```

In [8]:
```
df_train = df_train.sample(frac=1)
df_test = df_test.sample(frac=1)
df_train.head()
```

Out[8]:

	id	hashtags	content	emotion_label
50925	0x2a1a1f	[blog, shoponline, weirdo]	Let's get weird SoCal Residents on the Hunt f...	surprise
1846286	0x2d187f	[beginning, knowledge, fools, wisdom]	@bruyne_kdb17 @MesutOzil1088 Proverbs 1:7 The ...	fear
843557	0x1d731b	[annoYonce]	@ChinYugyeom @GOT7Official we will rise with t...	anger
313667	0x267d4e	[breakfast, delicious, healthy, coffee]	My #breakfast @ @UrbanPod_India: #delicious #h...	joy
901089	0x316b40	[Trumpkins]	Someone really needs to explain to @FoxNews th...	sadness

Step 4. Prepare training & validation data

In [ ]:

```python
# set dict for the labels
label_map = {
    'anger': 0, 'anticipation': 1, 'disgust': 2, 'fear': 3,
    'joy': 4, 'sadness': 5, 'surprise': 6, 'trust': 7
}
reverse_label_map = {v: k for k, v in label_map.items()}

df_train['label'] = df_train['emotion_label'].map(label_map)
X = df_train['content']
y = df_train['label']

# train_val split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Step 5. Encoding

Because we can't directly use text data as input, I tried a few ways to transform text data into vector.

(I) Sparse - TF-IDF

Use tf-idf to turn text into sparse vector.

In [ ]:
```
vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

(II) Dense - Text Embedding Model

Use SentenceTransformer 'multilingual-e5-large' to get dense vector.

In [ ]:
```
model = SentenceTransformer('intfloat/multilingual-e5-large')
X_train_embeddings = model.encode(X_train.tolist(), show_progress_bar=True)
X_test_embeddings = model.encode(X_test.tolist(), show_progress_bar=True)
```

Step 6. Classifying

I use several conbination of Encoder and classifier to test the performance.

#	Encoder	Classifier	Public Score	Private Score
1	multilingual-e5-large	K-Nearest-Neighbor	0.499	0.483
2		BertForSequenceClassification	0.469	0.455
3	multilingual-e5-large	Logistic Regression	0.466	0.448
4	TF-IDF	Logistic Regression	0.423	0.407
5	TF-IDF	Naive-Bayes	0.393	0.375
6	TF-IDF	LightGBM	0.375	0.360
7	TF-IDF	SVM	0.236	0.237
8	multilingual-e5-large	Neural Network	0.165	0.187

After testing, it seems that using SentenceTransformer with K-Nearest-Neighbor can get the best performance if only use single model.

Example 1 : TF-IDF as Encoder with Logistic Regression as classifier

In [ ]:
```
vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

clf = LogisticRegression(max_iter=1000, random_state=42)
clf.fit(X_train_tfidf, y_train)

y_pred = clf.predict(X_test_tfidf)
print(classification_report(y_test, y_pred, target_names=[reverse_label_map[i] for i in range(len(label_map))]))

df_test_tfidf = vectorizer.transform(df_test['content'])
df_test['predicted_label'] = clf.predict(df_test_tfidf)
df_test['emotion'] = df_test['predicted_label'].map(reverse_label_map)

output = df_test[['id', 'emotion']]
output.to_csv('predicted_emotions_1.csv', index=False)
print("Saved to predicted_emotions_1.csv")
```

Example 2 : SentenceTransformer as Encoder with K-Nearest-Neighbor as classifier

In [ ]:
```
model = SentenceTransformer('intfloat/multilingual-e5-large')
X_train_embeddings = model.encode(X_train.tolist(), show_progress_bar=True)
X_test_embeddings = model.encode(X_test.tolist(), show_progress_bar=True)

clf = KNeighborsClassifier(n_neighbors=20)
clf.fit(X_train_embeddings, y_train)

y_pred = clf.predict(X_test_embeddings)
print(classification_report(y_test, y_pred, target_names=[reverse_label_map[i] for i in range(len(label_map))]))

df_test_embeddings = model.encode(df_test['content'].tolist(), show_progress_bar=True)
df_test['predicted_label'] = clf.predict(df_test_embeddings)
df_test['emotion'] = df_test['predicted_label'].map(reverse_label_map)

df_test[['id', 'emotion']].to_csv('predicted_emotions_4.csv', index=False)
```

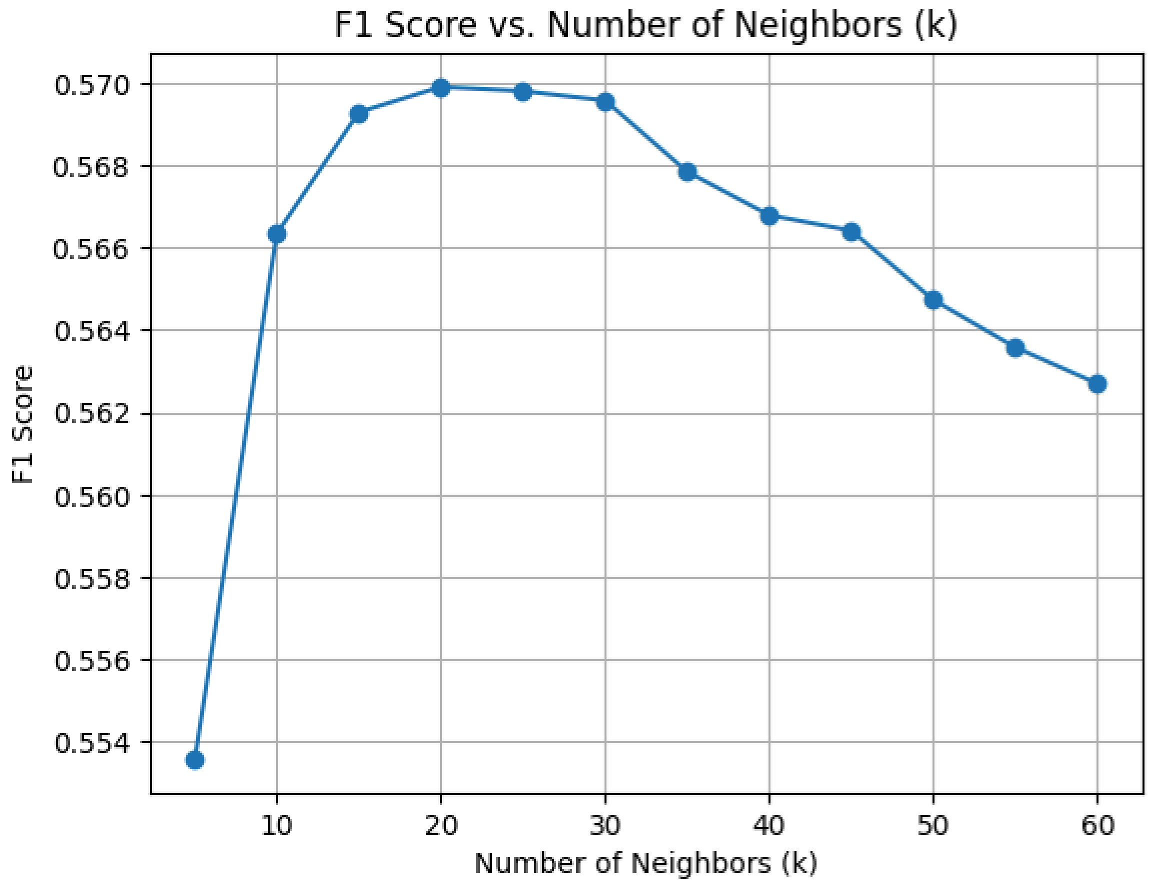
For choosing hyperparemeter of KNN (k), I plot a curve to find out the best k, and it seems that the choose of k doesn't significantly change the result.

In [34]:

```
import lightgbm as lgb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
import pickle
from sklearn.metrics import classification_report, f1_score
import pandas as pd
from sklearn.ensemble import VotingClassifier
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np
import matplotlib.pyplot as plt

with open('tfidf.pkl', 'rb') as f:
    X_train_tfidf, X_test_tfidf, y_train, y_test = pickle.load(f)
with open('multilingual-e5-large.pkl', 'rb') as f:
    X_train_embeddings, X_test_embeddings, y_train, y_test = pickle.load(f)
with open('multilingual-e5-large_df_test_embeddings.pkl', 'rb') as f:
    df_test_embeddings = pickle.load(f)
with open('df_test_tfidf.pkl', 'rb') as f:
    df_test_tfidf = pickle.load(f)

# Define the range of k values to test
k_values = range(5, 65, 5)
f1_scores = []

for k in k_values:
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train_embeddings, y_train)
    y_pred = clf.predict(X_test_embeddings)
    f1 = f1_score(y_test, y_pred, average='weighted')
    f1_scores.append(f1)
    print(f"k={k}, F1 Score={f1}")

# Plot the F1 scores
```

```python
plt.plot(k_values, f1_scores, marker='o')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('F1 Score')
plt.title('F1 Score vs. Number of Neighbors (k)')
plt.grid(True)
plt.show()
```

```
k=5, F1 Score=0.5535890971428357
k=10, F1 Score=0.5663400071698093
k=15, F1 Score=0.569287963815516
k=20, F1 Score=0.5699013948338362
k=25, F1 Score=0.5698001452528162
k=30, F1 Score=0.5695707259172089
k=35, F1 Score=0.5678405060887591
k=40, F1 Score=0.5667869089132466
k=45, F1 Score=0.5664138614309522
k=50, F1 Score=0.5647329347900865
k=55, F1 Score=0.5635790429901766
k=60, F1 Score=0.5627004795399552
```



Example 3 : BertForSequenceClassification

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from torch.utils.data import Dataset
import torch

data = df_train

# Preprocessing
```

```
data['content'] = data['content'].astype(str)
label_map = {
    'anger': 0, 'anticipation': 1, 'disgust': 2, 'fear': 3,
    'joy': 4, 'sadness': 5, 'surprise': 6, 'trust': 7
}
data['emotion_label'] = data['emotion_label'].map(label_map)

# Split the data into train and test sets
train_texts, val_texts, train_labels, val_labels = train_test_split(
    data['content'].tolist(),
    data['emotion_label'].tolist(),
    test_size=0.1,
    random_state=23
)

# Load tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Tokenize the data
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 = self.texts[idx]
        label = self.labels[idx]

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

```
        return {
            'input_ids': encoding['input_ids'].squeeze(0),
            'attention_mask': encoding['attention_mask'].squeeze(0),
            'labels': torch.tensor(label, dtype=torch.long)
        }

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

# Load pre-trained model
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=8
)

# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_strategy="epoch",
    load_best_model_at_end=True
)

# Define Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer
)

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

```python
# Save the model
trainer.save_model("./emotion_classifier")
```

In [ ]:
```python
# Load the trained model and tokenizer
model_path = "./emotion_classifier"
model = BertForSequenceClassification.from_pretrained(model_path)
tokenizer = BertTokenizer.from_pretrained(model_path)

model.eval()
label_map = {
    0: 'anger', 1: 'anticipation', 2: 'disgust', 3: 'fear',
    4: 'joy', 5: 'sadness', 6: 'surprise', 7: 'trust'
}

# Predict emotions
def predict_emotion(text):
    inputs = tokenizer(
        text,
        max_length=128,
        padding="max_length",
        truncation=True,
        return_tensors="pt"
    )

    with torch.no_grad():
        outputs = model(
            input_ids=inputs["input_ids"],
            attention_mask=inputs["attention_mask"]
        )

    logits = outputs.logits
    predicted_label = torch.argmax(logits, dim=1).item()
    return label_map[predicted_label]

# Apply the model to test data
df_test['emotion'] = df_test['content'].apply(predict_emotion)

# Save the result to a CSV file
output_df = df_test[['tweet_id', 'emotion']]
```

```
output_df.rename(columns={'tweet_id': 'id'}, inplace=True)
output_df.to_csv("test_predictions.csv", index=False)

print("Predictions saved to test_predictions.csv")
```

Step 7. Aggregate Model

I tried to use multiple model to vote and get final answer.

I have tried Hard voting, soft voting and soft voting with f1-score weighted

And the result is that using top3 model in the above table to do hard voting can get the highest score.

#	Voting	Model	Public Score	Private Score
1	Hard	1. 2. 3.	0.508	0.492

7-1 Soft Voting

In [ ]:

```
import lightgbm as lgb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
import pickle
from sklearn.metrics import classification_report
import pandas as pd
from sklearn.ensemble import VotingClassifier

with open('tfidf.pkl', 'rb') as f:
    X_train_tfidf, X_test_tfidf, y_train, y_test = pickle.load(f)
with open('multilingual-e5-large.pkl', 'rb') as f:
    X_train_embeddings, X_test_embeddings, y_train, y_test = pickle.load(f)

# Define classifiers
clf1 = MultinomialNB()
```

```
clf2 = LogisticRegression(max_iter=1000)
clf3 = RandomForestClassifier()
clf4 = lgb.LGBMClassifier()
clf5 = MLPClassifier(max_iter=1000)
clf6 = LogisticRegression(max_iter=1000)

# Train classifiers
clf1.fit(X_train_tfidf, y_train)
clf2.fit(X_train_tfidf, y_train)
clf3.fit(X_train_tfidf, y_train)
clf4.fit(X_train_embeddings, y_train)
clf5.fit(X_train_embeddings, y_train)
clf6.fit(X_train_embeddings, y_train)

from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import VotingClassifier

class MultiInputVotingClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, estimators, voting='soft'):
        self.estimators = estimators
        self.voting = voting

    def fit(self, X, y):
        for name, estimator, input_key in self.estimators:
            estimator.fit(X[input_key], y)
        return self

    def predict(self, X):
        predictions = []
        for name, estimator, input_key in self.estimators:
            predictions.append(estimator.predict_proba(X[input_key]))
        avg_predictions = np.mean(predictions, axis=0)
        return np.argmax(avg_predictions, axis=1)

# 定義模型及對應輸入
multi_voting_clf = MultiInputVotingClassifier(estimators=[
    ('nb', clf1, 'tfidf'),
    ('lr_t', clf2, 'tfidf'),
    ('rf', clf3, 'tfidf'),
    ('lgb', clf4, 'embedding'),
    ('mlp', clf5, 'embedding'),
```

```
    ('lr_e', clf6, 'embedding'),
])

# 定義多輸入數據
X_train_multi = {'tfidf': X_train_tfidf, 'embedding': X_train_embeddings}
X_test_multi = {'tfidf': X_test_tfidf, 'embedding': X_test_embeddings}

# 訓練與測試
# multi_voting_clf.fit(X_train_multi, y_train)
y_pred = multi_voting_clf.predict(X_test_multi)

print(classification_report(y_test, y_pred, target_names=[reverse_label_map[i] for i in range(len(label_map))]))

# Predict on test data

with open('multilingual-e5-large_df_test_embeddings.pkl', 'rb') as f:
    df_test_embeddings = pickle.load(f)
df_test_tfidf = vectorizer.transform(df_test['content'])

X_test_multi = {'tfidf': df_test_tfidf, 'embedding': df_test_embeddings}

df_test['predicted_label'] = multi_voting_clf.predict(X_test_multi)
df_test['emotion'] = df_test['predicted_label'].map(reverse_label_map)

# Save results to CSV
output = df_test[['id', 'emotion']]
output.to_csv('predicted_emotions_ensemble_multiple.csv', index=False)
print("Saved to predicted_emotions_ensemble_multiple.csv")
```

7-2 Hard voting (using predict output csv file)

In [ ]:

```
# Load the prediction results from the CSV files
test_predictions = pd.read_csv('test_predictions.csv')
predicted_emotions_logistic = pd.read_csv('predicted_emotions_all_embedding_logistic.csv')
predicted_emotions_knn = pd.read_csv('predicted_emotions_all_embedding_knn.csv')

# Ensure the order of IDs is the same in all files by sorting them
test_predictions = test_predictions.sort_values(by='id').reset_index(drop=True)
predicted_emotions_logistic = predicted_emotions_logistic.sort_values(by='id').reset_index(drop=True)
predicted_emotions_knn = predicted_emotions_knn.sort_values(by='id').reset_index(drop=True)
```

```python
# Ensure the order of IDs is the same in all files
assert (test_predictions['id'] == predicted_emotions_logistic['id']).all()
assert (test_predictions['id'] == predicted_emotions_knn['id']).all()

# Perform hard voting
predictions = pd.DataFrame({
    'test_predictions': test_predictions['emotion'],
    'logistic': predicted_emotions_logistic['emotion'],
    'knn': predicted_emotions_knn['emotion']
})

# Get the mode of the predictions
hard_voted_predictions = predictions.mode(axis=1)[0]

# Create the final DataFrame with the hard voted predictions
final_predictions = pd.DataFrame({
    'id': test_predictions['id'],
    'emotion': hard_voted_predictions
})

# Save the final predictions to a CSV file
final_predictions.to_csv('hard_voted_predictions.csv', index=False)
print("Saved to hard_voted_predictions.csv")
```

In [ ]: