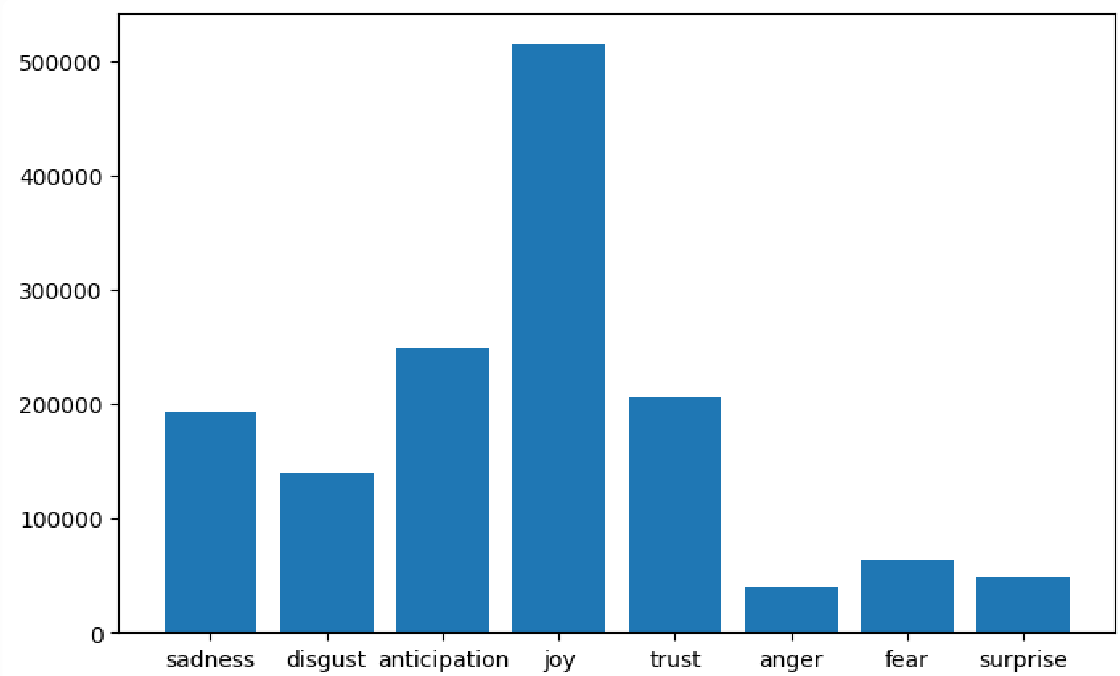
DM Lab2 HW Kaggle Competition

NTU R13227116 黃羿文

The competition

- Task: Classify twitter texts to their correct emotion label.
- Data:
  - **Size**: Training 1455563 | Testing 411972
  - **Emotion labels**: sadness, anger, anticipation, trust , surprise, joy, disgust, fear; 8 in total
  - **Distribution for each label**:



Progress overview

- **Submission attempts**: 3
  - test
  - all-joy: 0.14188
  - Decision Tree model: 0.18748
- **Tested methods**:
  - DT model
  - K-means model

Data Preprocessing

1. Removing stopwords and punctuation

I downloaded the stopwords package from `nltk` and removed them from the texts. I also used `regex` to specify the characters that I want to remove. Here I specified `r'\w+'` to indicate that I want to keep only the alphabets and get rid of puncuations. Lastly, because there are many `<LH>` tags in the tweets, but they don't mean anything to us, so I removed all of them using the `replace` method.

```
tokenizer = RegexpTokenizer(r'\w+') # remove puncuation

for text in texts:
    words = [word for word in text.split() if word.lower() not in stop_words] # re
    filtered_tokens.extend([word for word in words if tokenizer.tokenize(word)])

data["cleaned_text"] = texts.str.replace(r"<LH>", "", regex=True).str.strip() # Re
```

2. BOW Vectorizer

I used the Bag Of Words vectorizer to turn words into vectors. I followed the code in our `Lab2-Master` notebook, first transforming the words, then storing them in a document matrix.

```
from sklearn.feature_extraction.text import CountVectorizer

# build analyzers (bag-of-words)
BOW_vectorizer = CountVectorizer()

# 1. Learn a vocabulary dictionary of all tokens in the raw documents.
BOW_vectorizer.fit(data['cleaned_text'])

# 2. Transform documents to document-term matrix.
train_data_BOW_features = BOW_vectorizer.transform(data['cleaned_text'])
test_data_BOW_features = BOW_vectorizer.transform(data['cleaned_text'])
```

3. Storing the data as .pkl

I stored the word vectors as a pickle file because running it once takes too much time.

```
with open('BOW_features_500.pkl', "rb") as f:
    text_vector = pkl.load(f)
```

4. Rendering the training data and labels

Based on the `identification` file, I assigned the tweets that have labels as training data and those without as testing data.

```
X_train = text_vector[(dat_id['identification'] == 'train').values]
X_test = text_vector[(dat_id['identification'] == 'test').values]

y_train = labels['emotion']
```

Attempts

1. Decision Tree

I followed the code in our `Lab2-Master` notebook and built a Decision Tree model. The `X_train data` and `y_train` data are loaded into the model.

```
## build DecisionTree model
DT_model = DecisionTreeClassifier(random_state=1)

## training!
DT_model = DT_model.fit(X_train, y_train)

## predict!
y_train_pred = DT_model.predict(X_train)

## so we get the pred result
y_train_pred[:10]
```

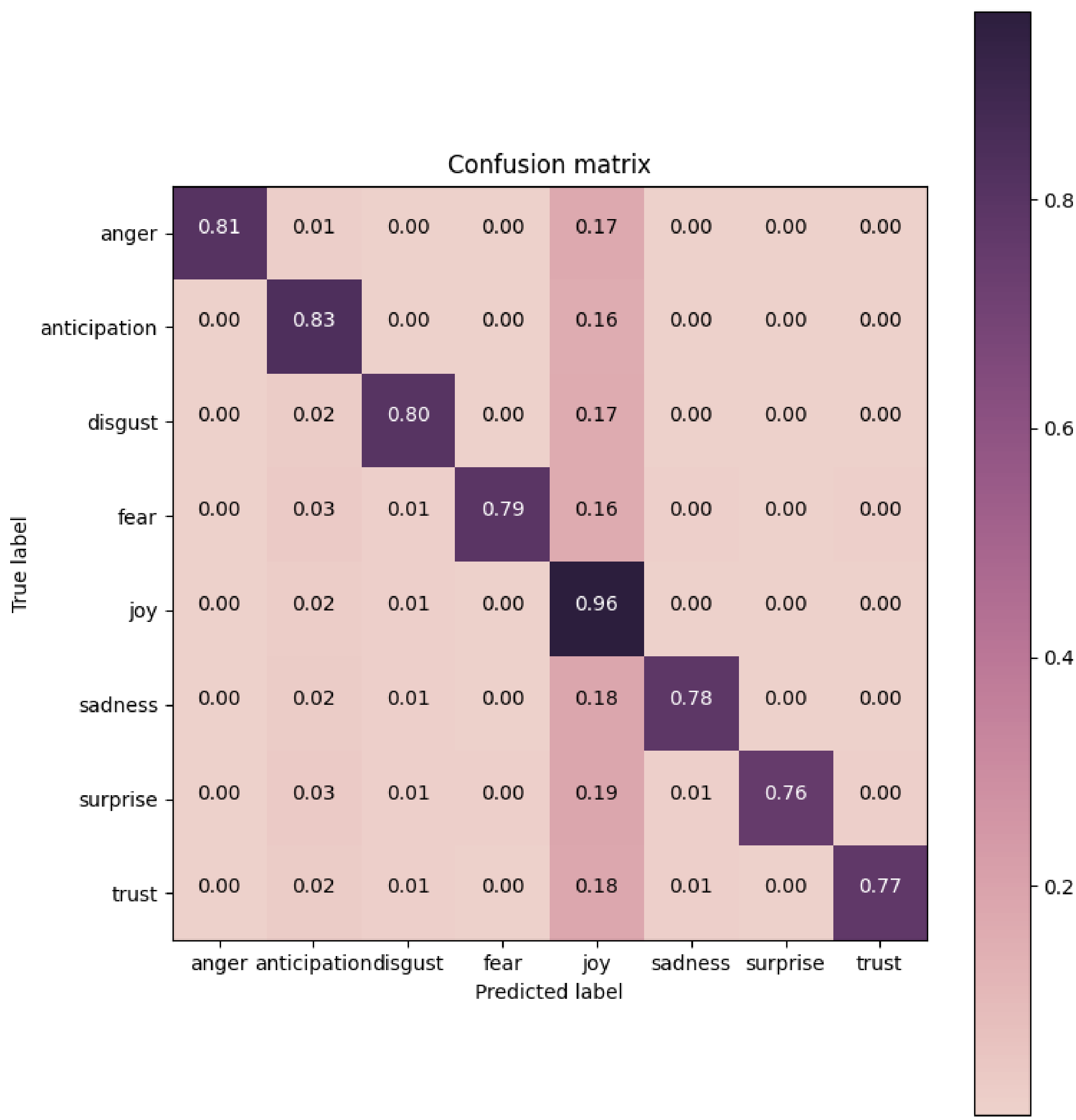
```
X_train.shape:  (1455563, 500)
y_train.shape:  (1455563,)
```

Accuracy

The training accuracy for the DT model is 0.85. The precision, recall, and f1-score performance is as below:

	precision	recall	f1-score	support
anger	0.90	0.81	0.85	39867
anticipation	0.89	0.83	0.86	248935
disgust	0.93	0.80	0.86	139101
fear	0.94	0.79	0.86	63999
joy	0.76	0.96	0.85	516017
sadness	0.97	0.78	0.86	193437
surprise	0.99	0.76	0.86	48729
trust	0.98	0.77	0.86	205478
accuracy			0.85	1455563
macro avg	0.92	0.81	0.86	1455563
weighted avg	0.87	0.85	0.85	1455563

Based on the accuracy for each label, we can visualize the confusion matrix below:



Test performance

The testing performance is nevertheless much worse than the training accuracy. The private score I recieved is 0.19162, and the public score is 0.18748.

I also attempted a submission with **all joy predictions**. I got an accuracy of 0.14525 (private score), which means that using the DT model didn't improve the scores too much.

K-Means

The second attempt is with K-Means. I built a K-Means model as follows:

```
from sklearn.cluster import KMeans
```

```
k = len(target_list)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X_train)
```

However, I encountered some problems with this method. At first I set 8 clusters (number of labels) and trained the classification model. Unfortunately, it wasn't possible to match each cluster to a unique label. As I checked the major label in each cluster, I found out that all of them were labeled 'joy'.

```
[      1      31      68 ... 1455524 1455557 1455561]
joy
[     10      11      14 ... 1455556 1455558 1455562]
joy
[      0       7       8 ... 1455531 1455543 1455548]
joy
[      4       5       9 ... 1455555 1455559 1455560]
joy
[     30      74     184 ... 1455497 1455511 1455512]
joy
[      2      32      42 ... 1455509 1455514 1455537]
joy
[      6      21      25 ... 1455517 1455536 1455542]
joy
[      3      93     110 ... 1455492 1455516 1455554]
joy
```

I later discovered that this is due to an disproportional amount of joy text compared to all the other labels.

To tackle this problem, I thought perhaps separating the `joy` text with others would help with the classification. Therefore, I used a DT model first to separate the `joy` and the `non-joy` text.

Using the DT model, I got an accuracy of **0.92**.

The next step is to take the non-joy data and classify them. I examined the labels in the `non-joy` data. Below is the result:

```
Counter({'anticipation': 242064,
         'trust': 199922,
         'sadness': 188173,
         'disgust': 135261,
         'joy': 91103,
         'fear': 62254,
         'surprise': 47416,
         'anger': 38830})
```

I trained a K-Means classifier on the `non-joy` data. However, the final distribution of the data still had 'joy' as the dominant label across all clusters, which led me to give up on K-Means.

Further dircetions

Due to the time limit, I couldn't try all the methods, but I have a few directions of improvement in mind:

1. Try cross-validation during training
2. Try other classifiers such as SVM
3. Try using LLM for word embedding