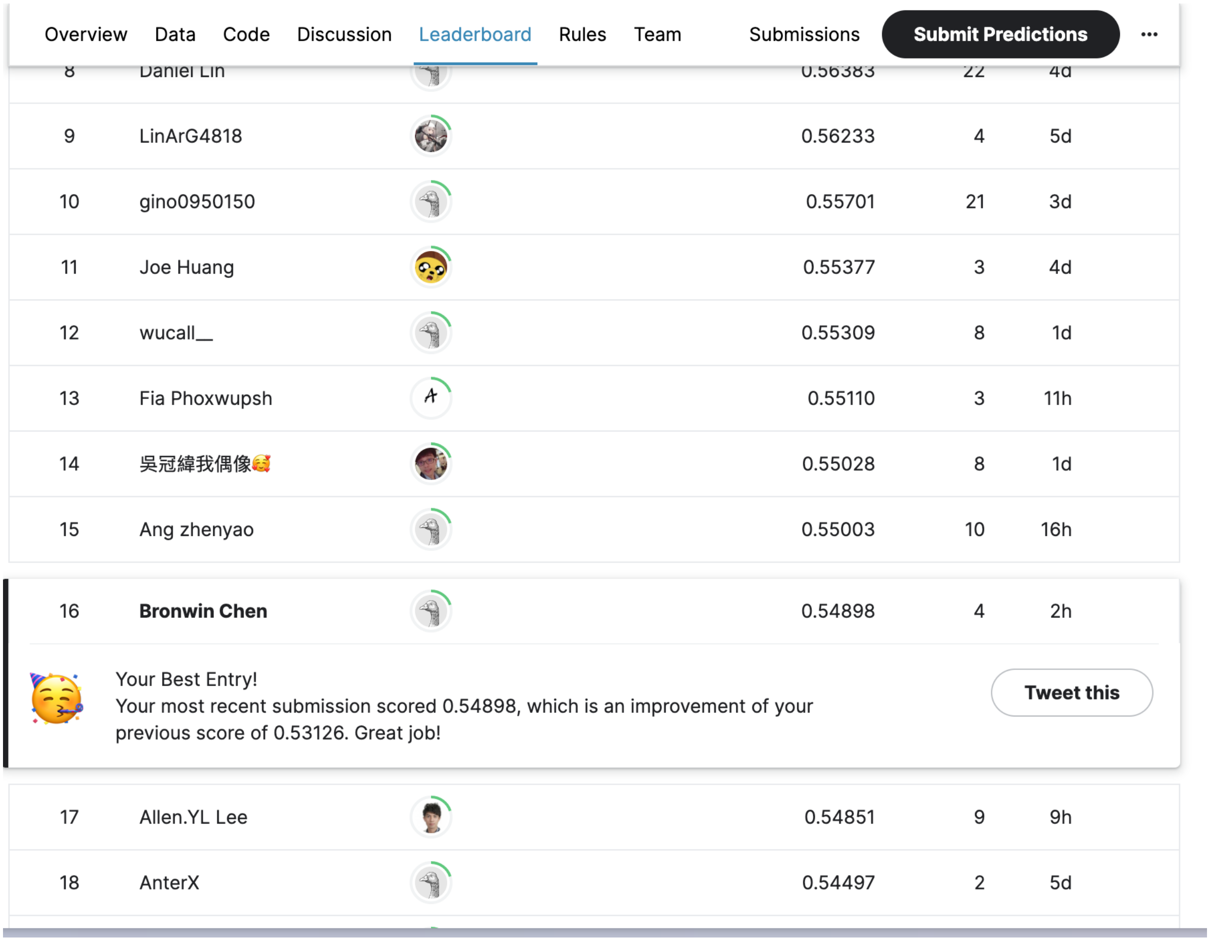
HW2 - Report

Student Information

- Name: 陳奕君
- Student ID: 111062610
- GitHub ID: wazenmai
- Kaggle name: Bronwin Chen
- Kaggle private scoreboard snapshot:



8	Daniel Lin	0.56383	22	4d
9	LinArG4818	0.56233	4	5d
10	gino0950150	0.55701	21	3d
11	Joe Huang	0.55377	3	4d
12	wucall__	0.55309	8	1d
13	Fia Phoxwupsh	0.55110	3	11h
14	吳冠緯我偶像🥰	0.55028	8	1d
15	Ang zhenyao	0.55003	10	16h
16	**Bronwin Chen**	0.54898	4	2h
17	Allen.YL Lee	0.54851	9	9h
18	AnterX	0.54497	2	5d

I did two kinds of experiment. First is using decision tree, naive bayes, and logistic regression. Second is fine-tuning on pre-trained BERT models. So my report would split to two parts.

Simple Model

Ref: `Data-Analysis.ipynb`

Preprocessing

Following are the steps that I use to preprocessing, since I don't want my simple model to have too many complex features, I did a lot of transformation on get rid of punctuation, emoji, and other unclean words.

Basic Preprocessing

1. Drop unuse columns

```
train_data.drop(["hashtags"], axis=1, inplace=True)
```

2. Change the data type of each columns

```
train_data["tweet_id"] = train_data["tweet_id"].astype(str)
train_data["text"] = train_data["text"].astype(st
```

3. Drop duplicated and missing values

```
# drop missing values
train_data.dropna(inplace=True)

# drop duplicated values
train_data.drop_duplicates(keep="first", inplace=True)
```

I did the same step at testing dataset, then we get the final shape of both dataframe

```
cleaned train:  (1455563, 3)
cleaned test:  (411972, 2)
```

Advanced Preprocessing

1. Remove userhandles

```
import neattext.functions as nfx
data["text_userhandles"] = data["text"].apply(nfx.remove_userhandles)
```

2. Lower casing

```
data["text_lower"] = data["text_userhandles"].str.lower()
```

3. Handle emoji

```
data["text_emoji"] = data["text_lower"].apply(lambda text: emoji.demojize(text))
```

4. Remove puntuation

```
punc_to_remove = string.punctuation
def remove_punctuation(text):
    return text.translate(str.maketrans('','', punc_to_remove))
data["text_punc"] = data["text_emoji"].apply(lambda text: remove_punctuation(text))
```

5. Remove stopwords

```
STOPWORDS = set(stopwords.words("english")) # from nltk

def remove_stopword(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
data["text_stop"] = data["text_punc"].apply(lambda text: remove_stopword(text))
```

6. Remove unwanted words

I notice that there is a lot of <LH> in text, so I remove it since it's meaningless.

```
unwanted_words = ["lh"]
text = lambda x: ' '.join(w for w in x.split() if not w in unwanted_words)
data["cleantext"] = data['text_stop'].apply(text)
```

The column of `cleantext` is the text after advanced preprocessing.

	tweet_id	text	emotion	cleantext
0	0x376b20	People who post "add me on #Snapchat" must be ...	anticipation	people post add snapchat must dehydrated cuz m...
1	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	sadness	see trump dangerous freepress around world tru...
2	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	fear	issa stalking tasha facewithtearsofjoyfacewith...
3	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	joy	thx best time tonight stories heartbreakingly ...
4	0x2c91a8	Still waiting on those supplies Liscus. <LH>	anticipation	still waiting supplies liscus
5	0x368e95	Love knows no gender. 😢😭 <LH>	joy	love knows gender cryingfaceloudlycryingface
6	0x249c0c	@DStvNgCare @DStvNg More highlights are being ...	sadness	highlights shown actual sports watches triathl...
7	0x359db9	The #SSM debate; <LH> (a manufactured fantasy ...	anticipation	ssm debate manufactured fantasy used distract ...
8	0x23b037	I love suffering 🙃🙃 I love when valium does no...	joy	love suffering upsidedownfaceupsidedownface lo...
9	0x1fde89	Can someone tell my why my feeds scroll back t...	anger	someone tell feeds scroll back 30 tweets saw 1...

Feature Engineering

Here I use word frequency (`CountVectorizer`) and term-frequency and inversed-document frequency (`TfidfVectorizer`) as two features and feed in naive bayes model seperately.

For decision tree and logistic regression, I feed the word frequency feature as input to it.

For the train-test splitting, I use `sklearn.model_selection.train_test_split` to split the dataset, I choose `cleantext` column as data feature and `emotion` column as label.

Model Explanation

Naive Bayes

This model consider every feature as a independent possibility distirubtion.

And I find that both features get similar results on this tweet data.

```
Model accuracy on train set :  0.5983
Model accuracy on test set :  0.5141
```

I submit the naive bayes result on submission dataset, and only get `0.44656` score on public leaderboard.

For the logistic regression, it failed to converge within the limit iteration, and only get `0.4` accuracy on test test set.

For the decision tree algorithm, even if I set the `max_depth` to 10, it still need to run more than 4 hours, and my colab just terminate it automatically, it shows that may these simple models are not suitable for this task.

Data Exploration

Besides training the model, I also did some key word exploration on the dataset. Since we have 8 emotion, I wonder that what is the most important words in each emotion, maybe my model could cheat on it or find some interesting insight?

1. Extract emotion keywords from text data, here I use `Counter()` in `nltk` to find the most common words in the category.

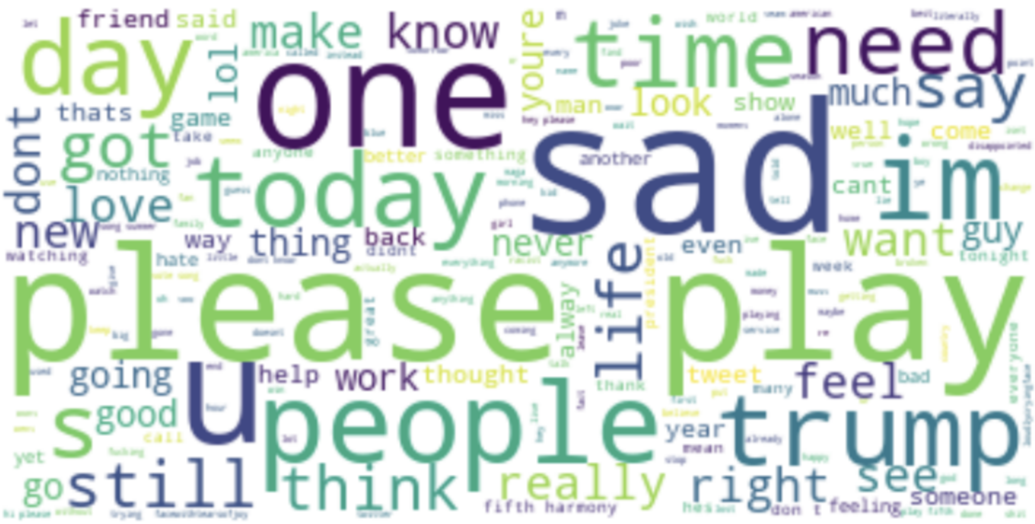
```
def extract_keywords(text, num = 100):
    tokens = [tok for tok in text.split()]
    most_common_tokens = Counter(tokens).most_common(num)
    return dict(most_common_tokens)

# extract joy keywords
joyList = data[data['emotion'] == 'joy']['cleantext'].tolist()  # create a joy list
joyDoc = ' '.join(joyList)
joyKeywords = extract_keywords(joyDoc)                          # extract the keywords

# extract other 7 emotion...
```

2. Use WordCloud to visualize the keywords in each emotion.

```
def plot_wordcloud(docx):
    mywordcloud = WordCloud(background_color="white").generate(docx)
    plt.figure(figsize = (13,10))
    plt.imshow(mywordcloud, interpolation = 'bilinear')
    plt.axis('off')
    plt.show()
plot_wordcloud(sadnessDoc)
```



100 most important words in sadness emotion.
For more wordcloud picture, please refer to `Data-Analysis.ipynb`

BERT Model

Ref: `Preprocess-bert.ipynb`, `Model-bert.ipynb`

After trying simple model, my strategy turn to find a much bigger pre-trained model and fine-tune the pre-trained model using our tweet data.

Preprocessing

Since the vocabulary size of the BERT is very big, it also include emoji and punctuation, I did not do much pre-processing step on the text data. I'm afraid that if I did too much preprocessing on text, it may reduce the information of the text and get worse result.

Here, I only lower casing the data.

```
train_data["text"] = train_data["text"].str.lower()
test_data["text"] = test_data["text"].str.lower()
```

Feature Engineering

One of the most important thing before training a BERT, is to transform the text data to id numbers according to the vocabulary dictionary. By turning all of the text data to id numbers, BERT can simply transform it to vectors and feed in the transformer model.

Thanks the help of `huggingface` `transformers.AutoTokenizer`, we can simply use the downloading tokenizer to transform the text. The tokenizer would do three things: One, transform text to id numbers. Second, add special token *[CLS]* at the head of the sentence, and *[SEP]* at the end of the sentence, so that BERT cau be familiar with these text data. Third, add attention mask to the dataset.

```
input_ids = []
attention_mask = []

for ind in train_data.index:
    tokenized_text = tokenizer(train_data.loc[ind]["text"], truncation=True)
    input_ids.append(tokenized_text["input_ids"])
    attention_mask.append(tokenized_text["attention_mask"])

train_data["input_ids"] = input_ids
train_data["attention_mask"] = attention_mask
```

Also, we need to transform our emotion to number labels.

1. Append emotion label to from emotion.csv to dataset

   ⭐⭐ Here I make the hashmap `[tweet_id: emotion]` first according to emotion.csv, then append the emotion on dataset based on this hashmap. Therefore, the runtime of this process would be O(N + N) = O(N). It save me many time since if I did `train_data["emotion"] = train_data["tweet_id"].apply(lambda x : emotion.loc[emotion["tweet_id"] == x].emotion.item()`, the runtime would be $O(N^2)$.

   ```
   # make hashmap [tweet_id: emotion]
   hashmap = {}
   for ind in emotion.index:
       hashmap[emotion.loc[ind]["tweet_id"]] = emotion.loc[ind]["emotion"]

   # get emotion from hashmap
   def get_emotion_from_id(id):
       return hashmap[id]

   # Therefore, the whole process is O(N)
   train_data["emotion"] = train_data["tweet_id"].apply(lambda x : get_emotion_from_id(x))
   ```

2. Turn emotion to number labels

   ```
   emotion_map = {
       "joy": 0,
       "anticipation": 1,
       "trust": 2,
       "surprise": 3,
       "sadness": 4,
       "fear": 5,
       "disgust": 6,
       "anger": 7,
   }
   emotion_list = ["joy", "anticipation", "trust", "surprise", "sadness", "fear", "disgust", "anger"]
   train_data["label"] = train_data["emotion"].apply(lambda x : emotion_map[x])
   ```

3. Transform DataFrame to Dataset

   ```
   total = train_data.shape[0]
   df1 = train_data.iloc[:int(total * 0.8), :]
   df2 = train_data.iloc[int(total * 0.8):, :]
   ```

```
# prepare training dataset
train_ds = Dataset.from_pandas(df1)
valid_ds = Dataset.from_pandas(df2)
```

Model Explanation

I use three kinds of BERT in my experiment, so first I would simply introduce what is BERT, then explan the difference between each model.

BERT, a Bidirectional Encoder Representations from Transformers, is a unsupervised natural language model. The architecture is same as Transformer Encoder, which use self-attention mechanism to attend each words than feed in a fully-connected network to learn the representation vector of each words. Transformer has been proved to beat many tasks on performance, and BERT is one of the successful model architecture from Transformer family.

There are two main task in BERT training duration, Masked Language Model (MLM) and Next Sentence Prediction (NSP). MLM is to randomly mask the words in input sentence and require BERT to output the correct words based on the neighborhood of this masked word. NSP is giving BERT two sentence and require it to output whether sentence 2 is the next sentence of sentence 1.

After training ther BERT till convergence, we can simply add any kinds of small models behind the BERT output layer, and fine-tune it with a few epochs, it has proved that this method get great performance at may natural language processing tasks, ex: the task in BLUE benchmark.

Then now we can go to my model explanation.

1. `distilbert-base-uncased`

   DistillBERT is the small version of BERT, it only half of number of layers of BERT, and using knowledge distillation technique to train the model, so that it can have similar result of BERT.

   I get `0.53125` and `0.52123` on the submission data on this model.

2. `bert-large-uncased`

   The bigger size of BERT, simply increase the model size but did not change any other training techniques or architecture.

   I

3. `vinai/bertweet-base`

   BERTweet is a mode that has same architecture of BERT but pre-trained on a large english tweet dataset. I think it could have better performance on our dataset since they are all tweets. Unfortunately, there is no more 40 hours for me to train another model, so I only leave my thoughts here.

Below is the result of my experiment, the last column is corresponding to the score I get on kaggle leaderboard. It shows that with more epoch of training and larger model, we can get better results, a simple but brute-force conclusion.

	Split (train, valid)	Train-Acc	Eval-Acc	Test-Acc (submission)

	Split (train, valid)	Train-Acc	Eval-Acc	Test-Acc (submission)
DistillBERT - sequence classification	(16000, 4000)	0.8471875	0.519	
DistillBERT - sequence classification	(80000, 20000)	0.8805375	0.583	
DistillBERT - sequence classification	(1164450, 291113)	0.8508214	0.658	0.53126
DistillBERT - sequence classification	(1455563, 291113)	0.9172	0.9172	0.52123
BERT-Large - sequence classification	(1455563, 291113)	0.9108	0.9108	**0.54898**

Training the Model

Here I provide the explanaiton of training BERT with `huggingface`.

1. Donwload the tokenizer and model you need

   Here I use sequence classification model to classify tweet text to one emotion label.

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=8)
```

2. Load the dataset

```
train_ds = load_from_disk("train_dataset")
test_ds = load_from_disk("test_dataset")
```

3. Make the compute metrics you want

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    total = predictions.shape[0]
    correct = 0
    for i in range(total):
        if predictions[i] == labels[i]:
            correct += 1
    return {'accuracy': correct / total, 'correct': correct}
```

4. Set the training parameters and start trianing!

```
training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=2e-5,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
```

```
    num_train_epochs=3,
    weight_decay=0.01,
    save_total_limit=5,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer.train()
trainer.save_model("DistilledBERT-Model")
```

5. Get the predictions from `predict` function

```
predictions = trainer.predict(test_ds)
```