DM hw2 Kaggle Competition Report

For detail please refer to code comments

Preprocessing steps

```
tweet_ids = []
texts = []
with open('dm2020-hw2-nthu/tweets_DM.json', 'r') as f: # open crawled data file
    i = 0
    for line in f.readlines():
        tmp = line.strip() # deal with space char
        # print(tmp)
        tmp = tmp.split('], "')[1]
        tmp = tmp.split(', "')[:2] # I only want id and text part use this pattern to cut them out

        tweet_id = tmp[0].split('": "')[1][:-1]
        # print(tweet_id)
        text = tmp[1].split('": "')[1][:-3].replace('<LH>', '') # I don't want "<LH>" appear in text since that it seems meani
ngless
        # print(text)
        tweet_ids.append(tweet_id)
        texts.append(text)
```

```
import pandas as pd
data = {'tweet_id': tweet_ids, 'text': texts}
df = pd.DataFrame(data, columns = ['tweet_id', 'text']) # build df which have 2 columns, id and text.
```

```
# split into training and testing dataframe
data_identification = pd.read_csv("dm2020-hw2-nthu/data_identification.csv") # read in the file which seperate training and te
sting data

df = pd.merge(df, data_identification, on='tweet_id') # merge with df based on tweet_id

# seperate training and testing data and drop the identification column
df_train = df.where(df['identification'] == 'train').dropna().drop(['identification'], axis=1)
df_test = df.where(df['identification'] == 'test').dropna().drop(['identification'], axis=1)
emotion = pd.read_csv("dm2020-hw2-nthu/emotion.csv") #read in label file
df_train = pd.merge(df_train, emotion, on='tweet_id') #merge label column with training data based in tweet_id
print(df_train)
print(df_test)
```

feature engineering steps

Firstly, I try BOW accompany with NB classifier. This trial gave me a 0.3 F1-score, which is pretty bad. Therefore, I decided to use BERT as pretrained model in next trial.

```
BOW_500 = CountVectorizer(max_features=500) # build analyzers (bag-of-words)
BOW_500.fit(df_train['text']) # apply analyzer to training data
train_data_BOW_features_500 = BOW_500.transform(df_train['text'])
train_data_BOW_features_500.shape## check dimension
```

```
(1455563, 500)
```

```
from sklearn.naive_bayes import MultinomialNB

X_train = BOW_500.transform(df_train['text'])
y_train = df_train['emotion']
X_test = BOW_500.transform(df_test['text'])
```

explanation of your model

For the first trial, I use a Naïve Bayes classifier and use BOW of 500 feature as input. The result was 0.3 F1-score.

```
NB_model = MultinomialNB() ## build NB model
NB_model = NB_model.fit(X_train, y_train) ## training!
y_test_pred = NB_model.predict(X_test) ## predict!
```

The next page is simple explanation of the 4 attempt of BERT model. Although BERT perform pretty well, I did not train many times since each attempt takes several hours, which is time-consuming.

first BERT trial: use roberta and not adjusting any hyperparameter of the imported model. I get a 0.53xx F1-score on this one, which is pretty good. Therefore, I will start to tune some hyperparameter in next trial.

```
from simpletransformers.classification import ClassificationModel, ClassificationArgs

df_train.rename(columns={'emotion':'labels'}, inplace=True)

# Optional model configuration
model_args = ClassificationArgs()
model_args.labels_list = ["anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", "trust"]

# Create a ClassificationModel
model = ClassificationModel("roberta", "roberta-base", num_labels=8, args=model_args)

# Train the model
model.train_model(df_train)

# df_test
y_test_pred, _ = model.predict(df_test['text'].to_numpy())
# print(y_test_pred)
output = {'id': df_test['tweet_id'], 'emotion': y_test_pred}
output_df = pd.DataFrame(output, columns = ['id', 'emotion'])
output_df.to_csv(r'submission.csv', index=False)
```

Second BERT trial: use roberta and set learning rate to a larger value and use only half of the training instances. I get a 0.3xx F1-score on this one, which is pretty bad. This could due to two possible reasons. One is lack of training samples the other is that larger learning rate could result in unstable update of model weight while back propagating.

```
from simpletransformers.classification import ClassificationModel, ClassificationArgs
# df_train.rename(columns={'emotion':'labels'}, inplace=True)

# Optional model configuration
model_args = ClassificationArgs()
model_args.labels_list = ["anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", "trust"]
model_args.learning_rate = 1e-4
model_args.output_dir = "outputs_lr=1e4_frac=0.5/"

# Create a ClassificationModel
model = ClassificationModel("roberta", "roberta-base", num_labels=8, args=model_args)

# Train the model
model.train_model(df_train.sample(frac=0.5, replace=False, random_state=5))

# df_test
y_test_pred, _ = model.predict(df_test['text'].to_numpy())

output = {'id': df_test['tweet_id'], 'emotion': y_test_pred}
output_df = pd.DataFrame(output, columns = ['id', 'emotion'])
output_df.to_csv(r'submission.csv', index=False)
```

Third BERT trial: use roberta and set learning rate to a smaller value (1e-5). I get a 0.55xx F1-score on this one, which is an improvement of the first one. The reason could be that smaller learning rate let the model update more smoothly. Hence, get a better performance.

```
from simpletransformers.classification import ClassificationModel, ClassificationArgs
# df_train.rename(columns={'emotion':'labels'}, inplace=True)

# Optional model configuration
model_args = ClassificationArgs()
model_args.labels_list = ["anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", "trust"]
model_args.learning_rate = 1e-5
model_args.output_dir = "outputs_lr=1e5/"

# Create a ClassificationModel
model = ClassificationModel("roberta", "roberta-base", num_labels=8, args=model_args)

# Train the model
model.train_model(df_train)

# df_test
y_test_pred, _ = model.predict(df_test['text'].to_numpy())

output = {'id': df_test['tweet_id'], 'emotion': y_test_pred}
output_df = pd.DataFrame(output, columns = ['id', 'emotion'])
output_df.to_csv(r'submission.csv', index=False)
```

For the 4th trial, I use 5e-6 as learning rate, and get 0.53 F1-score.

```
from simpletransformers.classification import ClassificationModel, ClassificationArgs
df_train.rename(columns={'emotion':'labels'}, inplace=True)

# Optional model configuration
model_args = ClassificationArgs()
model_args.labels_list = ["anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", "trust"]
model_args.learning_rate = 5e-6
model_args.output_dir = "outputs_lr=5e-6/"

# Create a ClassificationModel
model = ClassificationModel("roberta", "roberta-base", num_labels=8, args=model_args)

# Train the model
model.train_model(df_train)

# df_test
y_test_pred, _ = model.predict(df_test['text'].to_numpy())

output = {'id': df_test['tweet_id'], 'emotion': y_test_pred}
output_df = pd.DataFrame(output, columns = ['id', 'emotion'])
output_df.to_csv(r'submission.csv', index=False)
```

Since the hyperparameter I’ve tune is all about learning rate, and the performance did improve or decline based on different lr, we can probably conclude that there is a suitable lr value for this dataset and model. When the value is too large, it could make the update of model unstable and hard to converge. On the other hand, when the value is too small, it could happen that the model stuck at local maximum or minimum.