Data Mining Lab2 Kaggle Competition Report

Ei Kyi Phyu Khin (112065422)

Data Preprocessing

First, load two csv files and json files using pandas. Then for json file, it has a lot of columns and I used only _source column and took only tweet_id and text and combine them to one dataframe.

```
tweets.head()
```

	_score	_index	_source	_crawldate	_type
0	391	hashtag_tweets	{'tweet': {'hashtags': ['Snapchat'], 'tweet_id...	2015-05-23 11:42:47	tweets
1	433	hashtag_tweets	{'tweet': {'hashtags': ['freepress', 'TrumpLeg...	2016-01-28 04:52:09	tweets
2	232	hashtag_tweets	{'tweet': {'hashtags': ['bibleverse'], 'tweet_...	2017-12-25 04:39:20	tweets
3	376	hashtag_tweets	{'tweet': {'hashtags': [], 'tweet_id': '0x1cd5...	2016-01-24 23:53:05	tweets
4	989	hashtag_tweets	{'tweet': {'hashtags': [], 'tweet_id': '0x2de2...	2016-01-08 17:18:59	tweets

```
tweets._source[1]
```

```
{'tweet': {'hashtags': ['freepress', 'TrumpLegacy', 'CNN'],
  'tweet_id': '0x2d5350',
  'text': '@brianklaas As we see, Trump is dangerous to #freepress around the world. What a <LH> <LH
> #TrumpLegacy.  #CNN'}}
```

```
tweets._source[1]['tweet']['text']
```

```
'@brianklaas As we see, Trump is dangerous to #freepress around the world. What a <LH> <LH> #TrumpLe
gacy.  #CNN'
```

```python
for tweet in tweets['_source']:
    tweet_id.append(tweet['tweet']['tweet_id'])

tweet_text = []
for tweet in tweets['_source']:
    tweet_text.append(tweet['tweet']['text'])
```

```python
tweet = pd.DataFrame({'tweet_id': tweet_id, 'text': tweet_text})
```

```python
print(tweet.shape)
tweet.head()
```

```
(1867535, 2)
```

	tweet_id	text
0	0x376b20	People who post "add me on #Snapchat" must be ...
1	0x2d5350	@brianklaas As we see, Trump is dangerous to #...
2	0x28b412	Confident of your obedience, I write to you, k...
3	0x1cd5b0	Now ISSA is stalking Tasha 😂 😂 😂 <LH>

Then, I combine 3 data frames: data identification, emotion and tweet using common column tweet_id. After that, I split the dataframe into two, one for train and another for test using identification column.

```
combine_df.head()
```

	tweet_id	identification	emotion	text
0	0x28cc61	test	NaN	@Habbo I've seen two separate colours of the e...
1	0x29e452	train	joy	Huge Respect👏 @JohnnyVegasReal talking about l...
2	0x2b3819	train	joy	Yoooo we hit all our monthly goals with the ne...
3	0x2db41f	test	NaN	@FoxNews @KellyannePolls No serious self respe...
4	0x2a2acc	train	trust	@KIDSNTS @PICU_BCH @uhbcomms @BWCHBoss Well do...

```
train = combine_df[three_df['identification']=='train']
test = combine_df[three_df['identification']=='test']
```

Then dropped the identification column from both train and test as it will not be used anymore.

I tried to investigate train_df about how many tweets are in each emotion categories.

```
emotion
anger            39867
anticipation    248935
disgust         139101
fear             63999
joy             516017
sadness         193437
surprise         48729
trust           205478
Name: text, dtype: int64
```

Methods Used

The text pre-processing functions are done using the built-in tokenizer of TensorFlow and all the words in the dataset are assigned to a specific token. Next, the tokens are padded and truncated so that the model gets input of fixed shape. Then we create a dictionary for converting the name off the classes to their corresponding index. The text labels for the different classes are passes to get them as numeric representations. The sequential model is created using four different layers. The model is then trained and evaluated.

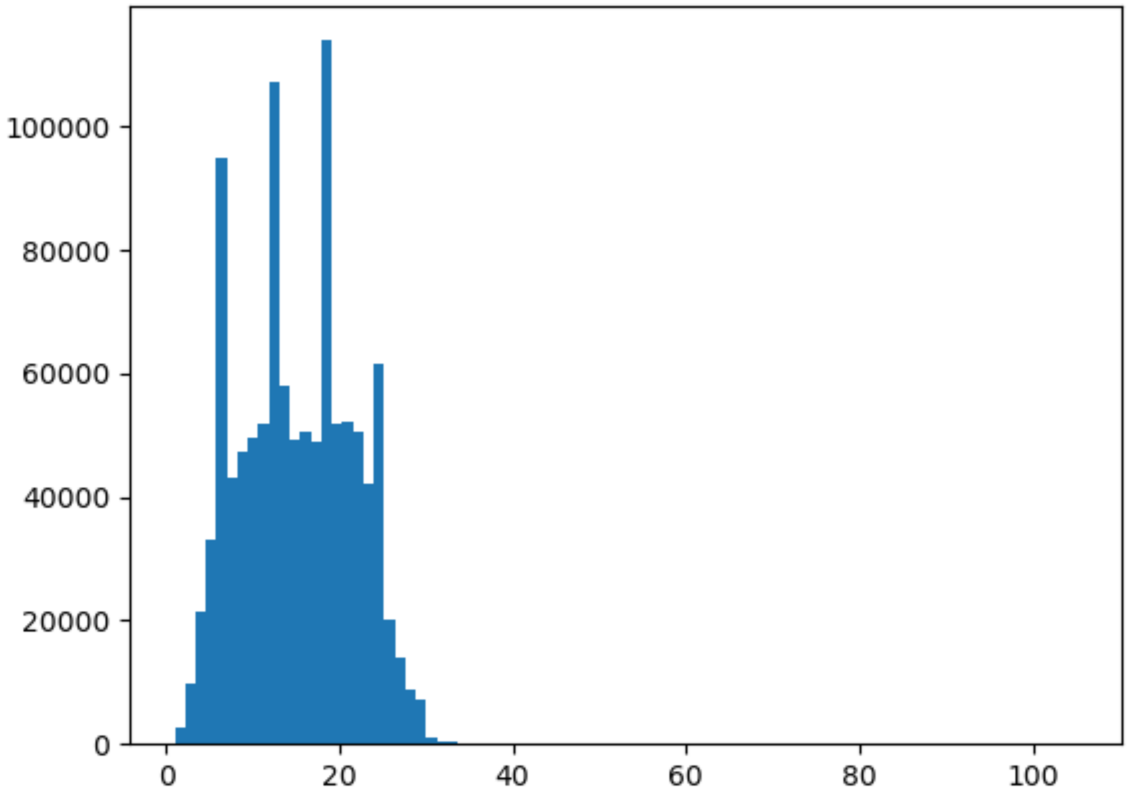
- **Tokenizing the Tweets**

  Used tensor-flow built-in tokenizer library. Tokenization generates random token value for plain text and stores the mapping in a database. The words of tweets need to be tokenized so that numbers get from tokenization of each word feed into the model and train on the data. Tokenizing gives each unique work a unique corresponding token. Here, created tokenizer that has the most frequently used 10,000 words from text corpus. Using the texts_to_sequences function we can see that the tweets have been tokenized.

```
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=10000, oov_token='<UNK>')

tokenizer.fit_on_texts(train_df['text'])

print(tokenizer.texts_to_sequences([train_df['text'][10]]))
 ✓ 22.9s

[[5, 99, 272, 46, 42, 666, 73, 12, 3, 154, 30, 154, 17, 1, 6716, 172, 2]]
```

- Padding and Truncating Sequences

  Did padding and truncating sequences from tokenizer because input has to be fixed size to feed into the model. The length of the tweets is calculated by counting the number of words separated by a space. A histogram is plotted to get the most common lengths of tweets in the dataset. X-axis represents the lengths of the word sequences in the tweets. Y-axis shows frequency or count of tweets falling into each bin.



Most of the tweets in the dataset are about 10 to 20 words long. There are very few tweets which are less than 4 words and also very few tweets of length 30 words or more.

Max length of 30 is set to truncate tweets over 30 words. Anything less than 30 is padded with 0 in its token sequence. This is done using pad_sequence() from Tensorflow function. The function will remove or add words from the end of the token sequence to get the length of 30. This will give the tweets a fixed input size.

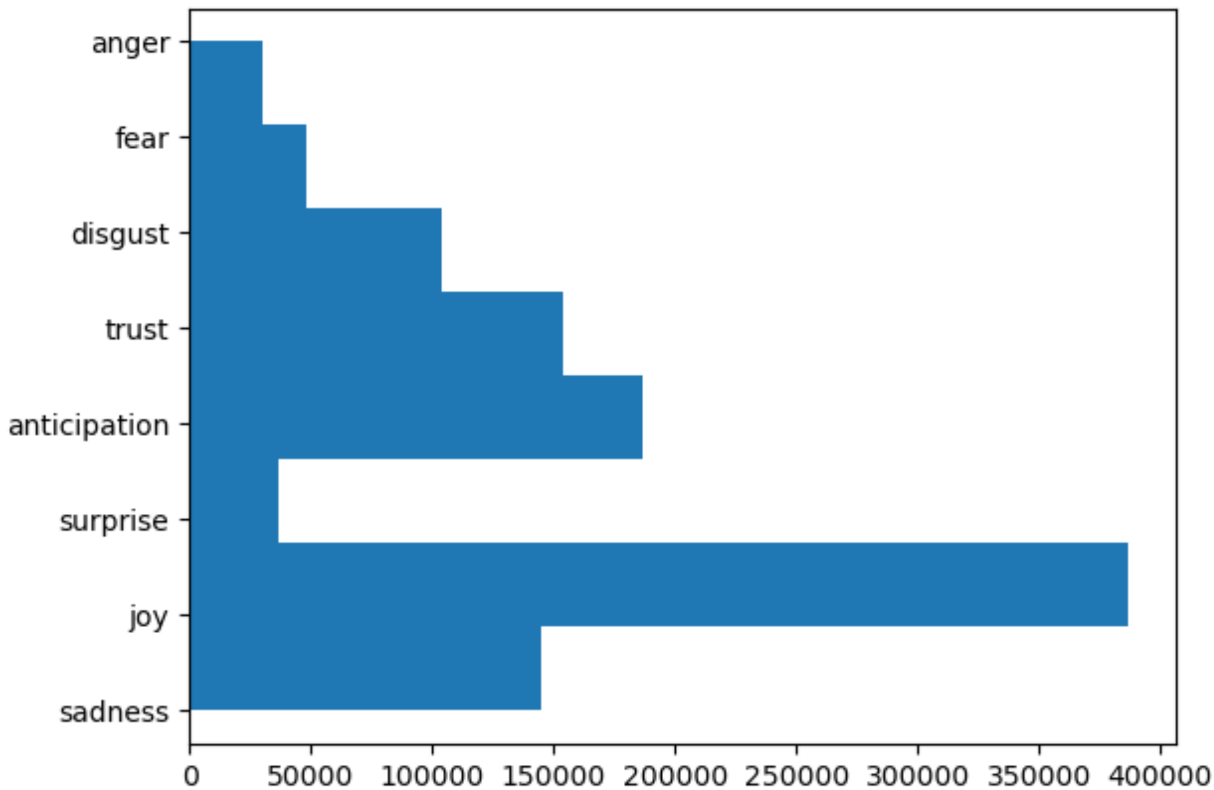
```
padded_train_sequences[20]
```

✓ 0.0s

```
array([2805,   53,  299, 1396,    2,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0])
```

- **Emotion Labels in Dataset**

  The classes are gotten from the labels from the training set. There are seven classes which are: angry, fear, disgust, trust, anticipation, surprise, joy, sadness. The histogram below shows the number of tweets for different classes.



- Modeling

  Recurrent Neural Network(RNN) sequential model is created using Keras. In RNN, neural networks gain information from previous step in the loop. The output of one unit goes into next one and information is passed. But the cons is RNN is not good for large datasets. Repetition updates to weights can lead to error gradients during update and network can be unstable. To overcome this problem, use Long-Short-Term Memory. LSTM can have memory about previous inputs for extended time durations. Use three layers for building the model: first embedding layer, LSTM layers and dense layer.

  The first layer of model is embedding, input dimension is 10,000 (most commonly used words in dataset) and output dimension is 16 which will be the size of the output vectors for each word. The input length of sequence is max length 30.

LSTM preserves the information from inputs passed through. Here used bidirectional which run the inputs in two ways, past to future and future to past, that can run backwards information from the future and also from the past. Use 20 cells are used(each cell has its own inputs, outputs and memory) and return sequence is set to true that means every time output is generated, it will be fed into another bidirectional LSTM layer as a sequence so that subsequent LSTM layer can have the required output.

The last layer is dense layer with 8 units for eight classes in dataset and activation is set to softmax which returns a probability distribution over the target classes.

The model is compiled with loss function 'sparse_categorical_crossentropy' as it is used for multi class classification problem as classes are not one-hot encoded. 'Adam' optimizer is used as it is really efficient for working with large datasets. The training metrics is used is accuracy to see how often predictions are equal to the actual labels. This is the model summary.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 30, 16)            160000

 bidirectional_2 (Bidirecti  (None, 30, 40)            5920
 onal)

 bidirectional_3 (Bidirecti  (None, 40)                9760
 onal)

 dense_1 (Dense)             (None, 8)                 328

=================================================================
Total params: 176008 (687.53 KB)
Trainable params: 176008 (687.53 KB)
Non-trainable params: 0 (0.00 Byte)
```

- Model Training

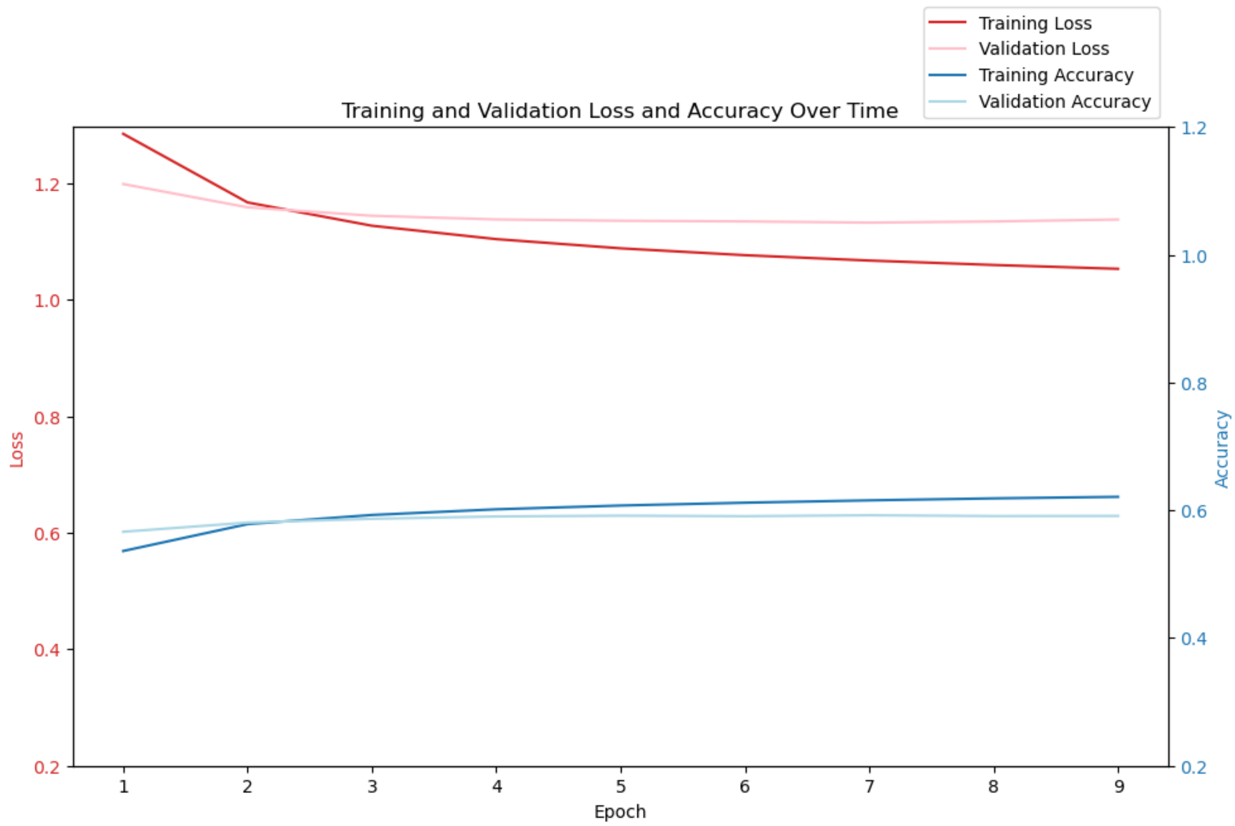
Validation set is prepared and its sequences are generated. Its labels are also converted to their corresponding numerical representations. Model is trained for 15 epochs. Epoch is a hyperparameters of gradient descent which controls number of complete cycle through training dataset. Early stopping callback is also set to stop the training model if the model doesn't improve in validation accuracy for over 2 epochs.

```
from sklearn.model_selection import train_test_split

# Splitting the data into 60% train, 20% validation, and 20% test
train_df, val = train_test_split(train_df, test_size=0.25, random_state=42)
```

```
h = model.fit(
    padded_train_sequences, train_labels,
    validation_data=(val_sequences, val_labels),
    epochs=10,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=2)
    ]
)
```

- Model Evaluation

Plots generated for accuracy and loss for training and validation over epochs.




```python
# Evaluate the model on the training data
train_loss, train_accuracy = model.evaluate(padded_train_sequences, train_labels, verbose=0)

# Evaluate the model on the validation data
val_loss, val_accuracy = model.evaluate(val_sequences, val_labels, verbose=0)

# Print out the accuracy
print(f'Training Accuracy: {train_accuracy}')
print(f'Validation Accuracy: {val_accuracy}')
```

✓ 9m 32.0s

```
Training Accuracy: 0.6250256299972534
Validation Accuracy: 0.5916469693183899
```

```
Training Set Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.54      0.56    145059
           1       0.60      0.87      0.71    386961
           2       0.79      0.24      0.36     36538
           3       0.72      0.64      0.68    186838
           4       0.73      0.37      0.49    154233
           5       0.52      0.51      0.51    104199
           6       0.72      0.47      0.57     47904
           7       0.67      0.28      0.40     29940

    accuracy                           0.63   1091672
   macro avg       0.67      0.49      0.53   1091672
weighted avg       0.64      0.63      0.61   1091672
```

```
Validation Set Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.50      0.52     48378
           1       0.58      0.84      0.69    129056
           2       0.75      0.23      0.35     12191
           3       0.68      0.60      0.63     62097
           4       0.67      0.34      0.45     51245
           5       0.47      0.45      0.46     34902
...
    accuracy                           0.59    363891
   macro avg       0.62      0.46      0.50    363891
weighted avg       0.61      0.59      0.57    363891
```

I tried a lot of models(decision tree, naïve bayes, logistic regression, etc) and vectorizers but I got the highest mark on this one.

DM2023 ISA5810 Lab2 Homework

Late Submission

Overview Data Discussion Leaderboard Rules Team Submissions

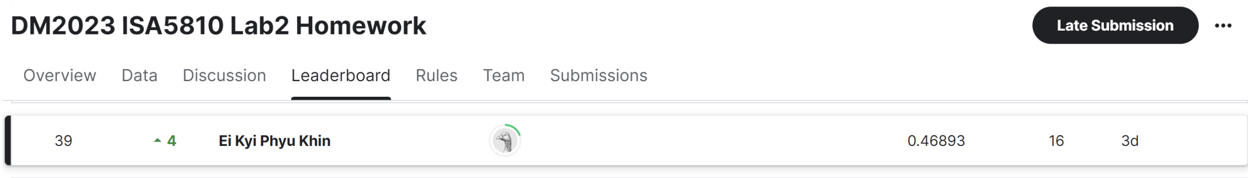
Private Score is used for your final score.

Submissions evaluated for final score

All Successful Selected Errors Recent

Submission and Description	Private Score	Public Score	Selected
test_predictions4.csv Complete · 3d ago	0.45286	0.46412	☐
test_predictions4(e11).csv Complete · 3d ago	0.46347	0.47652	☐
tfidf-logistic-regression1.csv Complete · 3d ago	0.45791	0.47006	☐
tfidf-logistic-regression.csv Complete · 3d ago	0.45817	0.47158	☐
BOW_NN.csv Complete · 3d ago	0.41577	0.4289	☐
BOW_NN-5epoach.csv Complete · 3d ago	0.4604	0.4741	☐
tfidf-naive-bayes.csv Complete · 4d ago	0.44688	0.46185	☐
test_predictions4.csv Complete · 4d ago	0.46712	0.47954	☐
test_predictions4.csv Complete · 4d ago	0.46921	0.47972	☐
Submission1.csv Complete · 4d ago	0.03505	0.03443	☐
test_predictions3.csv Complete · 5d ago	0.46893	0.48075	☑
test_predictions2.csv Complete · 5d ago	0.04419	0.05401	☐
output1.csv Complete · 6d ago	0.41388	0.4209	☐
predicted_emotions.csv Complete · 7d ago	0.37912	0.39257	☐
test_predictions.csv Complete · 7d ago	0.45084	0.46375	☐
submission.csv Complete · 8d ago	0.28309	0.28653	☐

Final Result on Kaggle



Further Improvement

I could enhance existing RNN model by incorporating additional features:

Score: Add sentiment or engagement scores to provide context.

Date: Utilize temporal data for trend analysis.

Hashtags: Analyze hashtags to understand tweet topics.

Experiment with Different Models and explore advanced deep learning models:

Attention Mechanism: Improve long-range dependency capture. Use different attention mechanisms like self-attention or multi-head attention.

BERT(Roberta or Distil BERT): Use BERT model as an encoder to convert input tweets into contextual embeddings. These embeddings can then be passed through additional layers for classification. To improve contextual understanding for classification.

Use Regularization techniques to prevent overfitting. Or try different hyperparameters like learning rate, batch size, etc, to improve model performance.