Introduction

The problem of sentiment analysis has always been a hot topic for several fields ranging from politics to finance to corporate decision making. With the existence of media like twitter which can easily propagate a topic through various groups of people, corporations and decision makers have to carefully monitor their group of interest's sentiments in order to make various other decisions. Hence, with publicly annotated available data, it becomes paramount to create sentiment analysis models to monitor the public's sentiment regarding certain topics. Therefore, the aim of the project is to create 2 different models from different architecture and compare their performance in the task of sentiment analysis.

Dataset

The dataset used in this experiment is Sentiment140 dataset, a publicly available data set created by three graduate students at Stanford University: Alec Go, Richa Bhayani, and Lei Huang. The data comprises approximately 1,600,000 automatically annotated Tweets.

Brief Description

The dataset given a sampled version of the original dataset with neutral sentiment removed. It sampled 10% of the original dataset so there is a total of 160,000 tweets to classify Hence, the problem become a binary classification problem of classifying whether a tweet is positive or negative

Model Architecture Selected

For the models we used the following model: LSTM, and BERT.
LSTM is an RNN-based model while BERT is a transformer-based model.
The reason this subsection is in the Dataset section is because the 2 different architectures work differently with different preprocessing. BERT requires labelling of separators in sentences and various words in order to be effective because of its 3 embedding layers: token embedding, segment embedding, and position embedding.

Data Analysis and Preprocessing

Initial Label Distribution

The 10% sampling from the original dataset evenly sampled the number of each class as shown in Figure 1. This is a good thing because we no longer have to deal with the fact that the majority classes will dominate the model prediction like in the previous project (we solved it using SMOTE in training dataset).

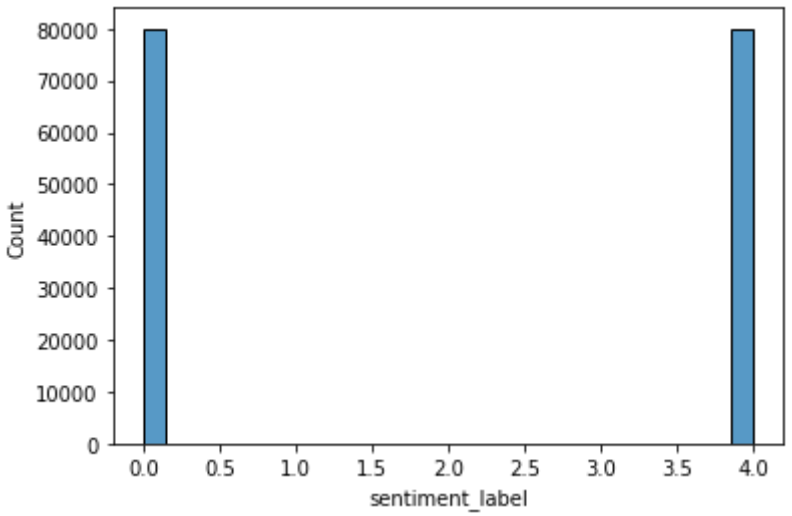

Figure 1, distribution of sentiments

Preprocessing for BERT and LSTM

While BERT has its own tokenizers and preprocessing as shown in Figure 4, LSTM uses a different tokenization and preprocessing which will be explained in this subsection. Different architectures have different input requirements and hence different preprocessing steps. BERT requires its attention mask and special tokens.

This step is important because we need to know a good sequence length to use for the input for BERT and LSTM in order to pad or truncate the input tokens.

The following list show preprocessing steps used for LSTM

1. Lower String
2. Tokenization
3. Remove Punctuations
4. Remove Stopwords
5. Remove rows with no tokens
6. Split train, test, validation set.
7. Out of Vocab masking
8. Pad to max length
9. Relabel positive to 1 from 4 and negative remains as 0

Due to the fact that this preprocessing step has removal of words and punctuation, there are rows with no tokens left. Hence, these rows were removed.
Figure 2, 3 and Table 1, 2 show the before and after boxplot of the length of tokens.

	tweet_processed_raw_len	tweet_processed_cleaned_len
count	160000.000000	160000.000000
mean	16.417069	8.383269
std	8.561186	4.363839
min	1.000000	0.000000
25%	9.000000	5.000000
50%	15.000000	8.000000
75%	23.000000	11.000000
max	103.000000	53.000000

Table 1: table of length distribution of tokens of tweets before row removal

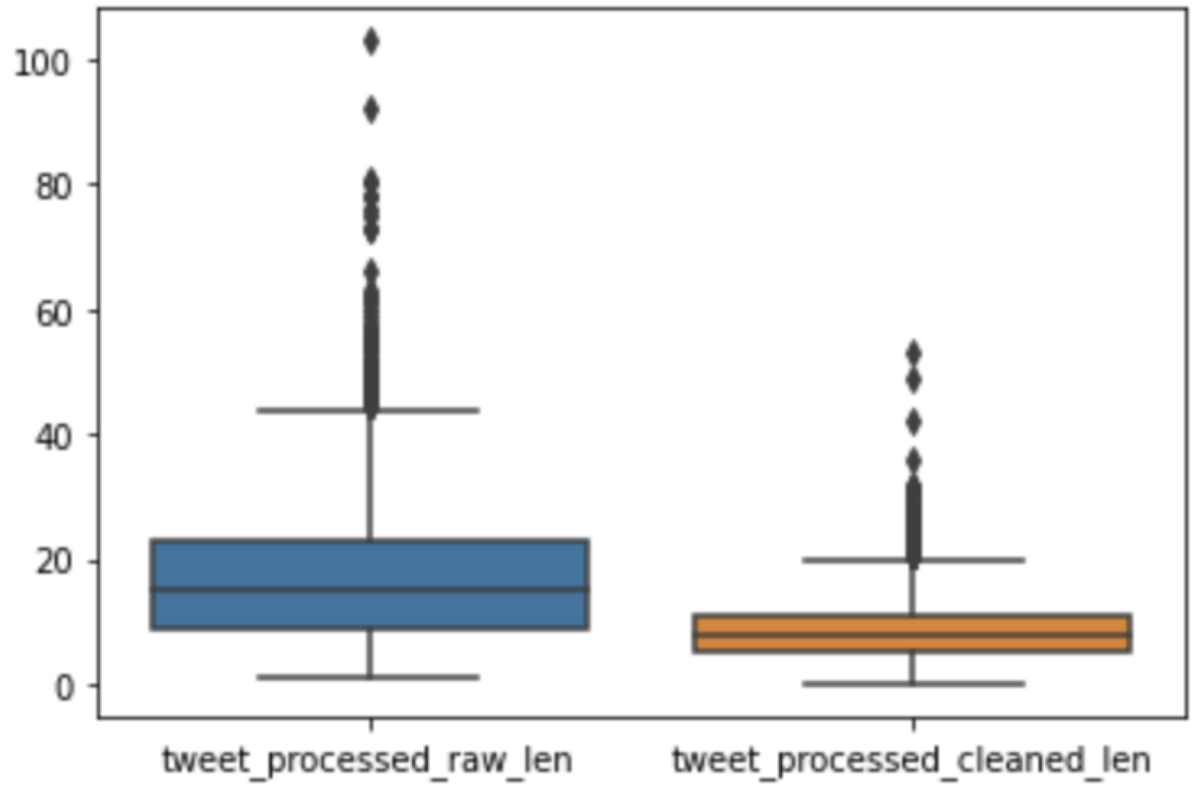

Figure 2: box plot of tokens length before row removal

	tweet_processed_raw_len	tweet_processed_cleaned_len
count	159978.000000	159978.000000
mean	16.418720	8.384422
std	8.560319	4.363032
min	1.000000	1.000000
25%	9.000000	5.000000
50%	15.000000	8.000000
75%	23.000000	11.000000
max	103.000000	53.000000

Table 2: table of length distribution of tokens of tweets after row removal

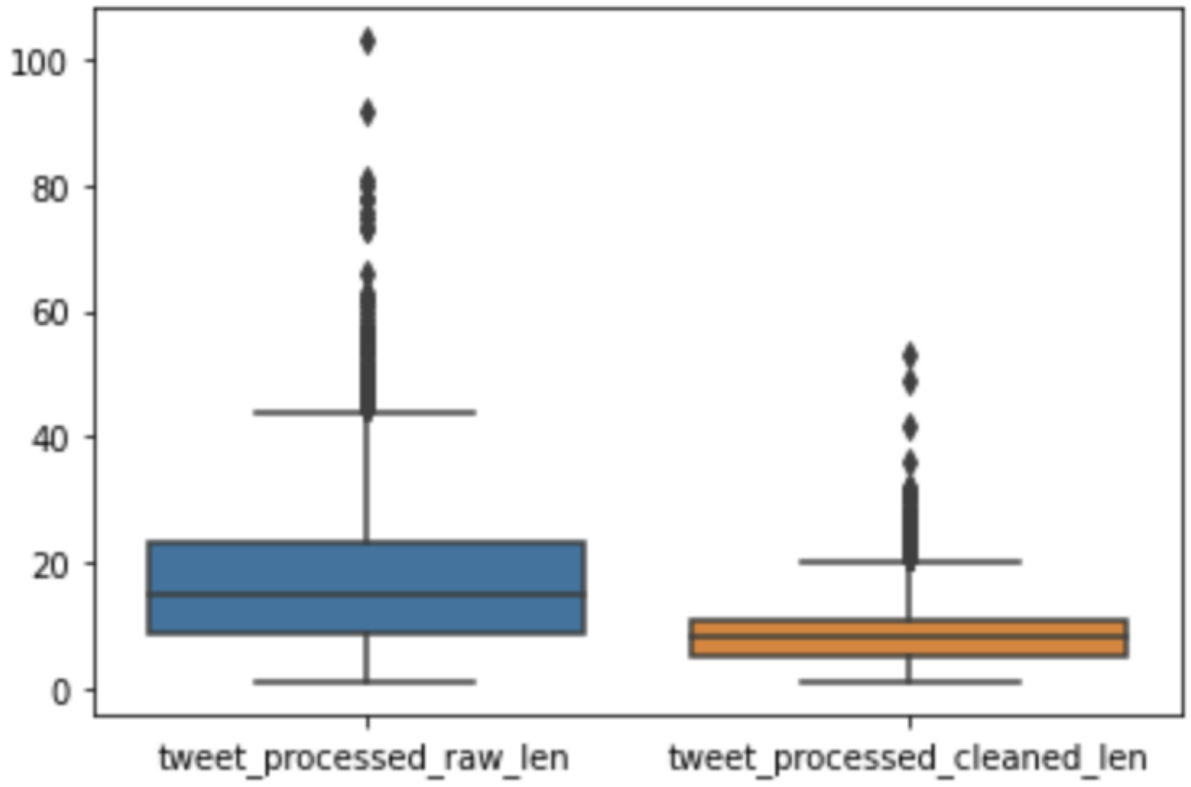

Figure 4: box plot of tokens length after row removal

Through this boxplot, the max sequence length of 50 was selected for BERT and 25 was selected for LSTM. This is because these look to be a good cutoff to truncate some outliers. The reason for BERT relying on the raw length instead of cleaned length like LSTM is because BERT preprocessing methods did not use stop removal or punctuation cleaning since the separator tokens will have to be added into the list of tokens as input (Since this is not mention in the documentation, the author assume as such).

Finally Table 3 shows the distribution of labels before and after the removal. The number of rows removed are negligible and the distribution percentage for each class remains similar so no over and under sampling is required.

```
4    80000
0    80000
Name: sentiment_label, dtype: int64
0    79992
4    79986
Name: sentiment_label, dtype: int64
```

Table 3, distribution of labels before and after removal of rows.

Below Figure 5 shows the BERT transformation step in code. For more detail, please refer to the Transformations codebase.

```
encoded_data_train = bert_tokenizer.batch_encode_plus(
    train_df['tweet_text'].values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=BERT_SEQ_LENGTH,
    return_tensors='pt',
    truncation=True
)
```

Figure 5 shows the function used to encode the raw tweet.https://www.glassdoor.com/Salary/Sievo-Finland-Salaries-EI_IE979469.0,5_IL.6,13_IN79.htm

Models and Results

Dataset used

For model's hyperparameter tuning, training and validation datasets are used.
For result comparison and confusion matrix, test dataset is used.

Possible improvement: we can further split training and validation into k-fold in order to ensure that our performance is not due to pure luck in randomising the right dataset. Statistic reminder: sampling distribution may not always get you the average and hence K-Fold is always recommended.

LSTM Architecture and hyperparameters used

Learning rate = 0.0001
Criterion = Binary Cross Entropy loss
Optimizer = Adam
Decay Rate = 0.5 at epoch 3 forward
Epochs = 7
Gradient Clipping = 3

Figure 6 shows a brief description of the architecture in LSTM.

```
SentimentLSTM(
  (embedding): Embedding(124575, 400)
  (lstm): LSTM(400, 256, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

Figure 6

BERT Architecture and hyperparameters used

lr = 0.00001 (1e-5),
eps = 0.00000001 (1e-8)
Epochs = 7
Scheduler = linear_schedule_with_warmup.
Create a schedule with a learning rate that decreases linearly from the initial lr set in the optimizer to 0, after a warm up period during which it increases linearly from 0 to the initial lr (1e-5 in this case) set in the optimizer.
(detail:
https://huggingface.co/docs/transformers/main_classes/optimizer_schedules#transformers.get_linear_schedule_with_warmup.num_warmup_steps)

For training bert, a pre-trained model is used and the final layer is removed to be replaced with a layer of 2 nodes indicating the 2 classes. This is basically transfer learning. The 2 nodes decision later causes an issue of trying to create an ROC curve comparison in a later section. In this case, the output value indicating positive classes is used to create the ROC curve.

Bert Architecture is extremely large and there are many examples out there. Hence, the screen shot with the final replaced layer section is shown in Figure 7.

```
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Results and Comparison

To note, the validation loss is actually testing loss for this result section. They used the same evaluation function for training so the string was not replaced.
The dataset used is testing dataset which has not been involved with training and hyper parameters tuning at all (unlike training and validation datasets).

Speed, loss and f1 score:

LSTM being the smaller smaller model is expected and is faster than bert when predicting the result as shown in Figure 8 and 9. It is at least 70x faster than BERT on most occasions (sometimes the speed drops to 250 it/s).

However, as we can see, with larger models and more advanced feature extraction techniques, BERT is able to achieve lower validation loss and higher F1-score.

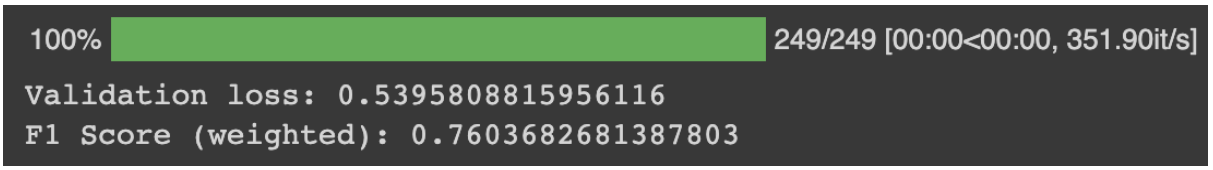

Figure 8: the progress bar shows LSTM prediction speed in batches per second (it/s).

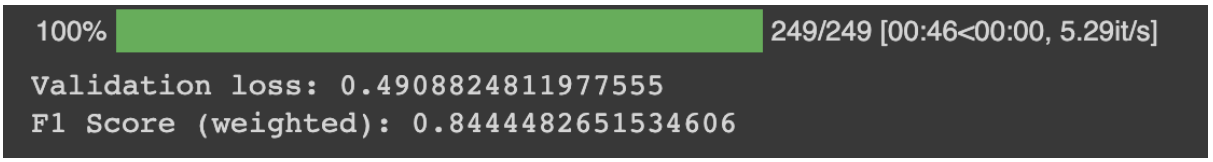

Figure 9: the progress bar shows BERT prediction speed

ROC AUC

As mentioned in the BERT Architecture section, the Bert final layer was replaced with a 2 output nodes layer representing positive and negative. The score from output nodes representing positive is used to create the BERT Positive ROC curve and compute AUC score. The resulting AUC is much higher than when only prediction is used. Hence, there is an argument to be made that trying to create BERT with a single output node final layer might yield a better result IF it is possible to do so.

As expected in figure 10, BERT has a significantly higher AUC compared to LSTM when involved with simply using the single node (0.92 vs 0.8425). However, if only predictions are involved, much of the curve granularity is loss and hence the AUC is not much better.

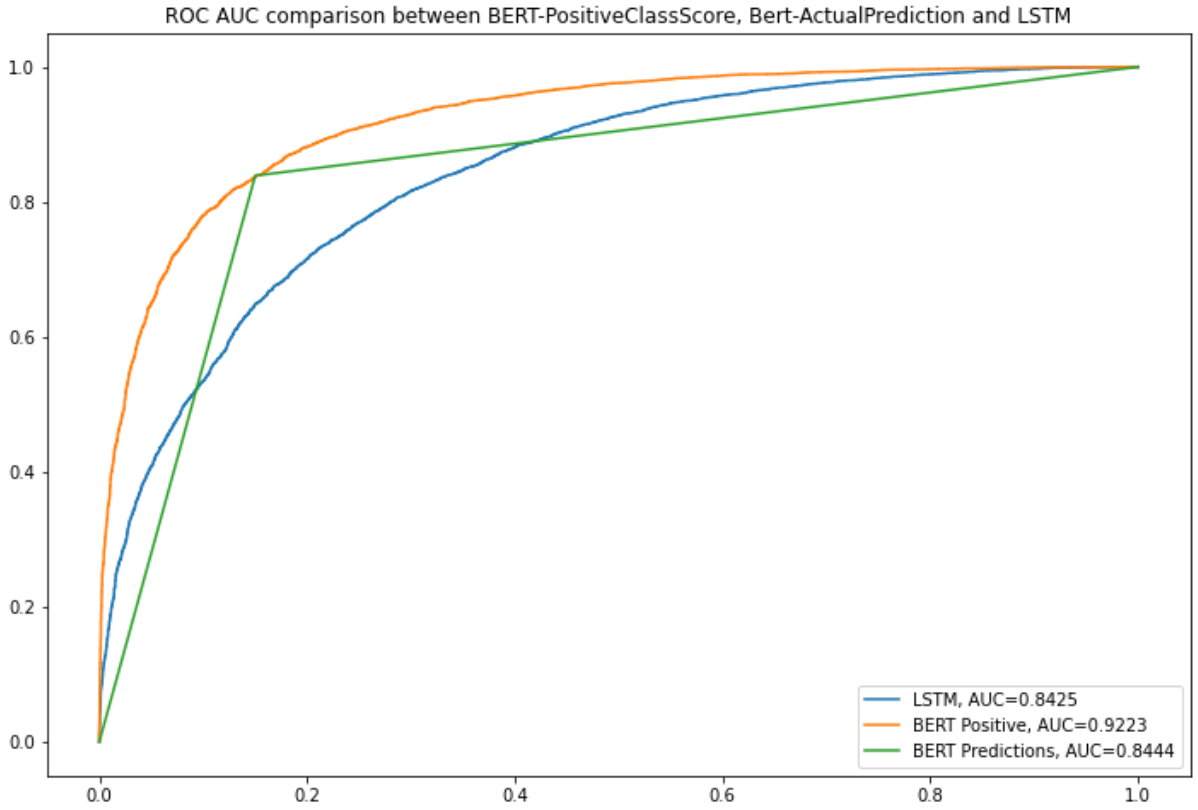

Figure 10

Precision-Recall Curve and Confusion Matrix

From Figure 11, BERT out clearly outperforms LSTM by nearly 10% in all categories (precision and recall). Meanwhile, LSTM has better precision than recall despite even class distribution but the 0.2% difference is within statistical variation.
Figure 12 shows the actual numbers of true positive, false positive, true negative and false negative of each model and why the number in Figure 11 looks similar for BERT's precision and recall.

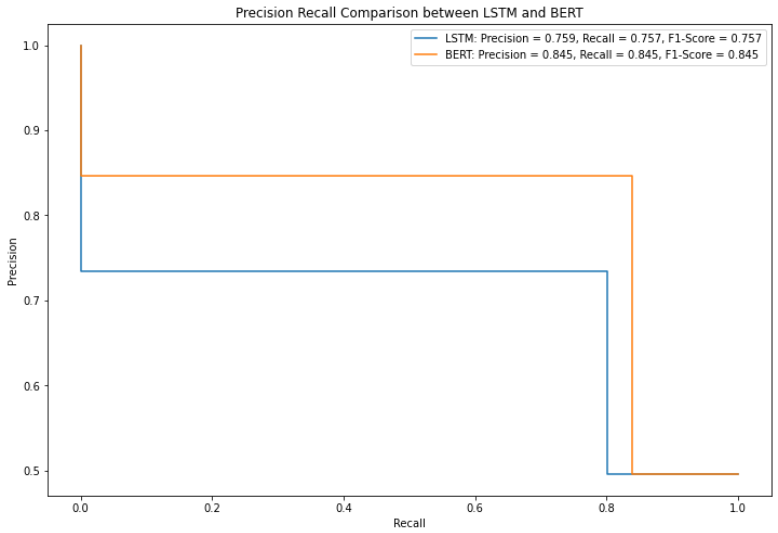

Figure 10 recall-precision curve comparison

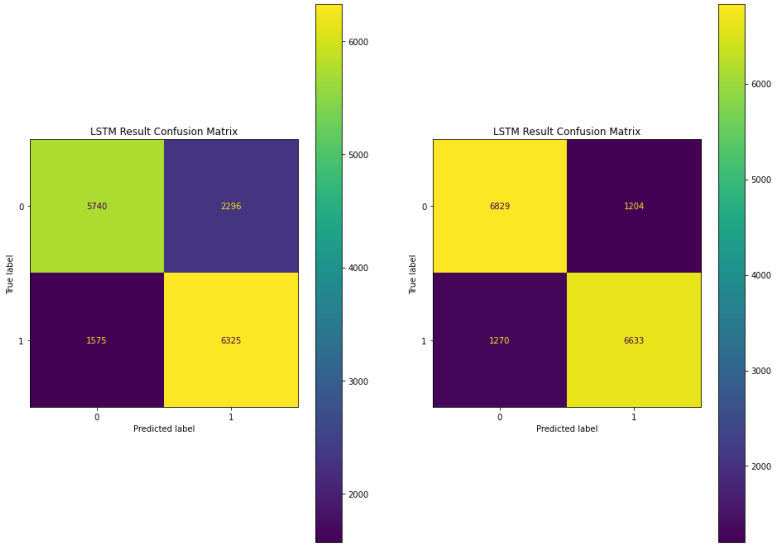

Figure 11 confusion matrix comparison

Weights Size

The weights for LSTM are 2x smaller than BERT's. For edge deployment or smaller computer deployment, this may matter.

Figure 12 shows the downloaded weights for size comparison

Conclusion

With the widely available publicly annotated data, it is possible to achieve above 80% F1-Score with BERT. It is possible to improve it with some better learning rate optimization, preprocessing and potentially changing the 2 nodes in the final layer to single node. However, the price of accuracy comes at the cost of speed. LSTM is 70x faster than BERT in most runs for this experiment. This might have to do with smaller input and model size. However, if accuracy is of concern and the cost of wrong prediction is higher, it is wiser to use BERT. However, if speed and size of model are of concern instead of accuracy, the author would recommend using LSTM in order to perform sentiment analysis.