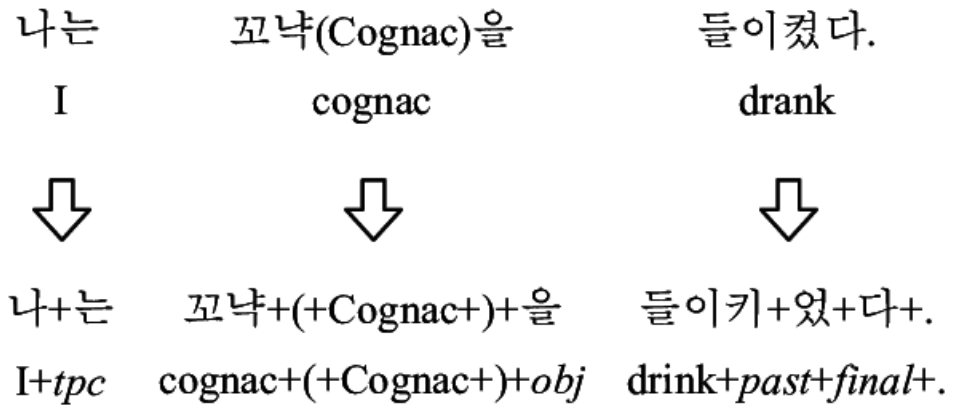
Synopsis

Korean film industry has recently gathered international interest after the release of several high-profile films and shows in the last few years. Parasite, Okja, and Squid Games are some of the most well-received and highly consumed media originating from Korea. Fueled by this piqued interest in Korean media and my Korean heritage, I had several friends and co-workers asking for recommendations regarding movies and TV series. Unfortunately, as a Korean-Canadian who grew up playing hockey near the Rocky Mountains, I was pretty oblivious to the Korean popular culture despite my fluency in Korean.

This inquiry and a Kaggle competition I saw regarding sentiment analysis of movie reviews motivated me to utilize Korean NLP. As my courses and online resources mainly dealt with English for NLP tutorials and exercises, I was highly curious how the model development process would differ when done in a different language. Plus, by doing a sentiment analysis of Korean movies using Korean reviews, I can have some good films I can recommend to my non-Korean friends and co-workers. Some similar older Korean film sentiment analysis exist, but it is my goal to take newer RNN architectures and try to further improve on the model's. Also, I wished to take some different approaches in data pre-processing compared to existing works done.

Background

Korean is an agglutinative language. This means that words are made up of a linear sequence of distinct morphemes. Morpheme is a short segment of languages that can be seen as the "smallest meaningful lexical item in a language". It is basically the smallest possible unit of a language that has its own meaning. So, a Korean "word" can be compromised of several morphemes. On the contrary, English is an isolating language. In a nutshell, each word typically consists of one morpheme. From the example below you can see each English word contains one morpheme. I, cognac, or drank cannot be split into a smaller lexical item. While Korean words 나는, 꼬냑을, and 들이켰다 consists of multiple morphemes.



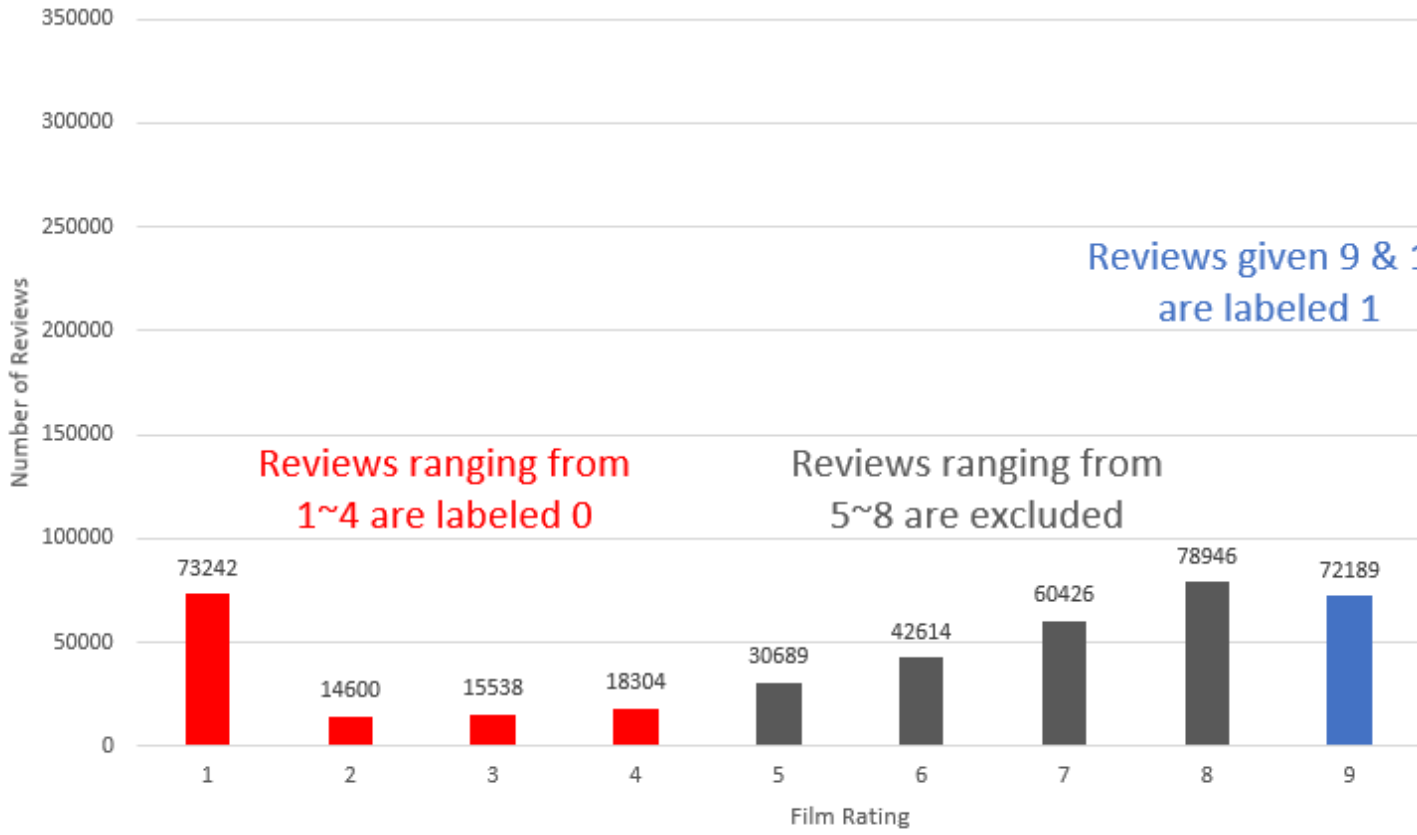
This means that in English, tokenization is done via a word. However, in Korean, the morphemes used within that word needs to be considered as well. A lot of NLP development efforts in Korea involve proper refinement/building of the morpheme analyzer tool.
https://konlpy.org/en/latest/references/#engines

Although there are official language rules about when to use spaces in sentences. Most Korean do not properly follow the grammar on when/where to have spaces between the words. Part of this is because

of the whole Morpheme/word system making the writer sometimes confused on the appropriate place to separate their word.

Dataset

I utilized an existing dataset for Korean film ratings along with their corresponding sentiment. The dataset was found from https://github.com/e9t/nsmc/. The review system rates films from 1-10 with 1 being the lowest rating. If reviewer gave the film 9 or 10, the corresponding sentiment was labeled with 1. If the reviewer gave the film 1~4, the corresponding sentiment was labeled with 0. Reviews from 5~8 were excluded from the dataset. The final dataset consists of 200000 reviews with a 75-25 split between the training and testing dataset.

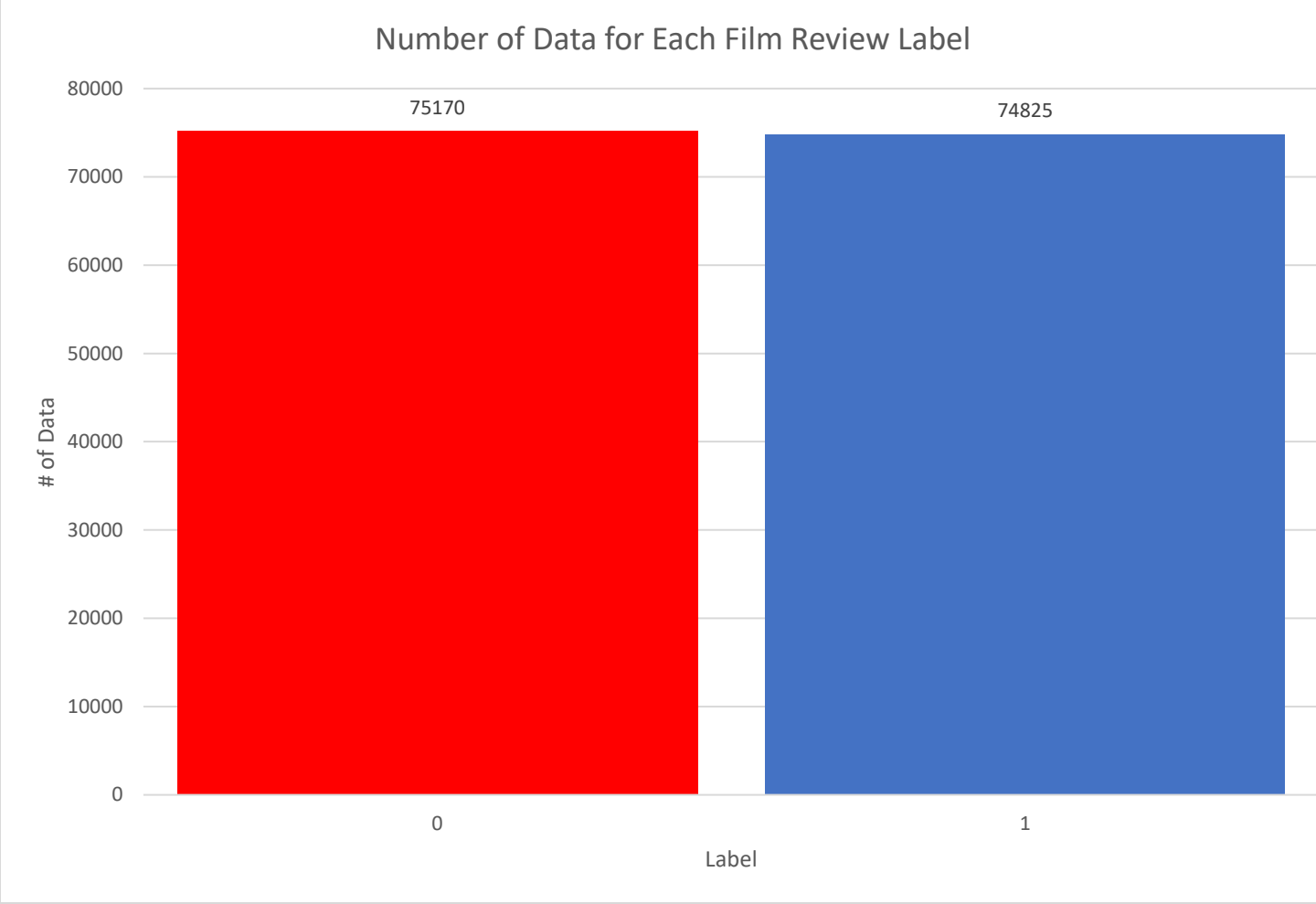


From the imported data, it can be seen that there are 3 columns: user id, document (sentiment), and the label. As user id is useless, this column will be dropped from the dataset.

	id	
0	9976970	C
1	3819312	흠...포스터보고 초딩영화
2	10265843	너무재
3	9045019	교도소 이야기구먼
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에

Looking at the high-level information about the data – there are 149995 non-null entries in the document (sentiment) column with none of the values in the label column being null. There are very minimal data missing from this overall dataset. There are no worries regarding duplicate reviews as in so many entries, it would be common to see duplicate sentiments/reviews especially if they are simple ones like “Good Movie”, “Recommend”, or “Horrible”. It can also be seen that the labels are object datatype. As there are only 5 empty values they will be dropped.

```
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   document  149995 non-null  object
 1   label     150000 non-null  int64
```



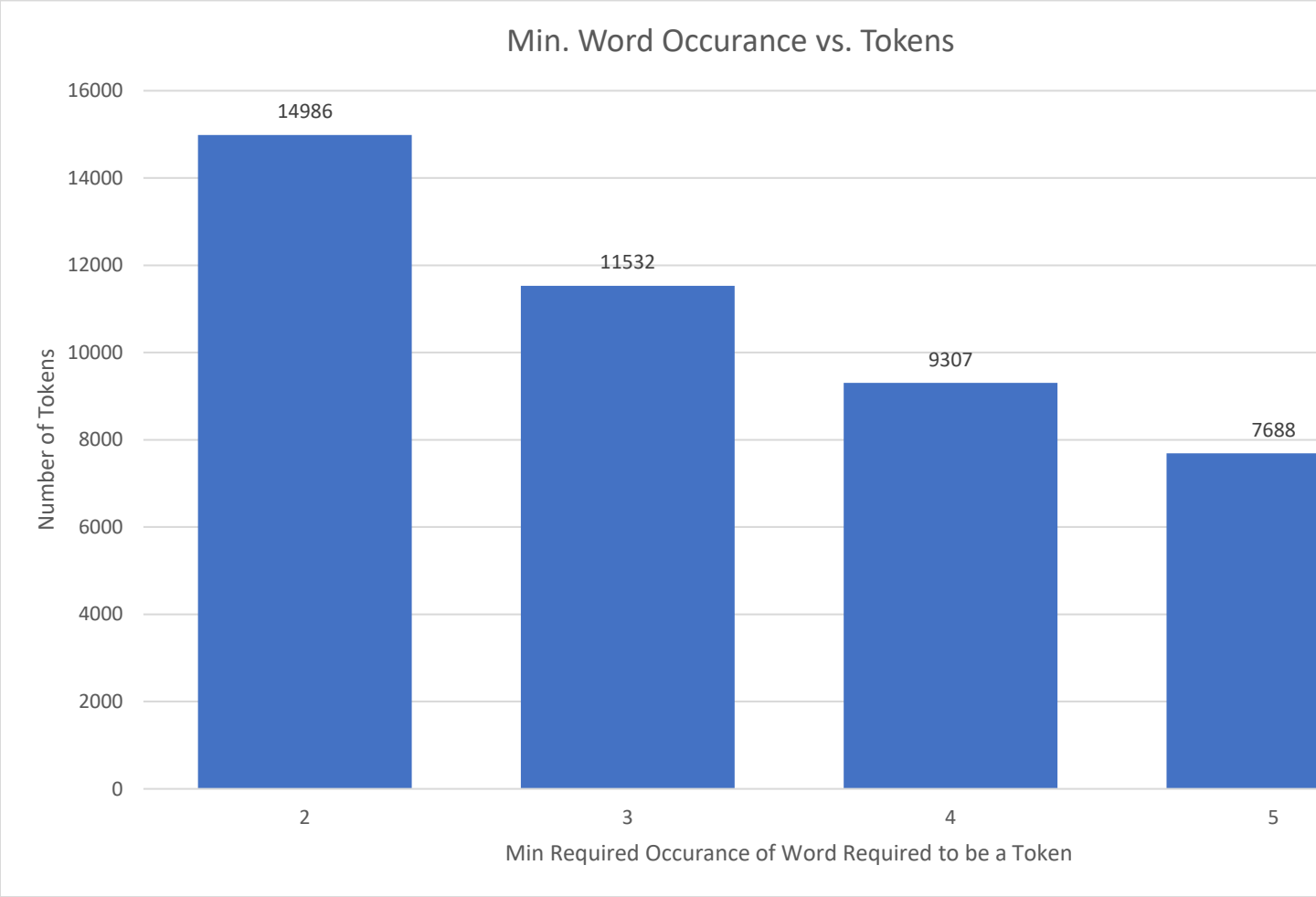
Just like in English NLP data pre-processing, special characters and punctuation needs to be removed. Since this is a Korean language, the Korean character range of ㄱ-ㅎ ㅏ-ㅣ 가-힣 was used. This range can be confirmed by looking at how they are listed in the Unicode chart. Thus, all characters excluding Korean and spaces were removed. It can be seen that punctuation were removed in the newly created column.

0	아 더빙.. 진짜 짜증나네요 목소리	0	
1	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1	
2	너무재밋었다그래서보는것을추천한다	0	
3	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0	
4	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1	사이몬페그의

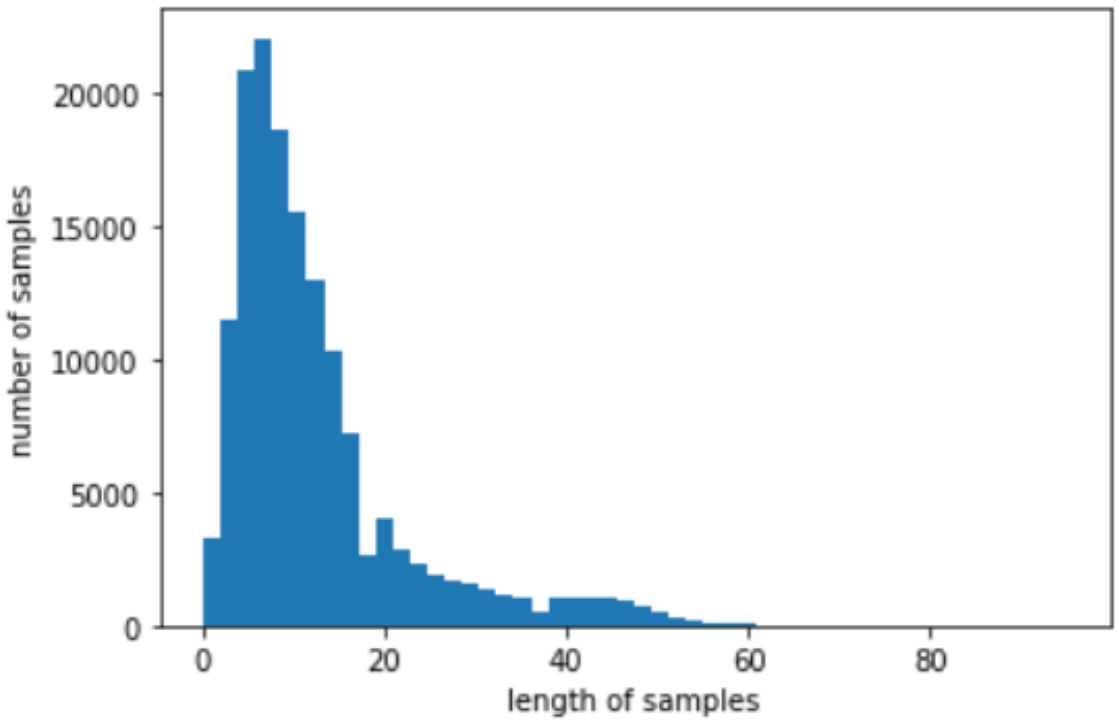
Korean stopwords are not supported by nltk.corpus; thus, they must be entered manually. This is combined with the tokenization from Open Korean Text Processor (OKT). This helps to simplified/unify the combinations/possibilities that arise from the morphemes as well.

```
7%|▉          | 10056/149995 [00:22<06:02, 385.93it/s]
```

From initial Tokenizing, there are more than 40000 tokens, the aim was to reduce that to <10000. Number of tokens were compared for each word occurrence threshold (in the dataset) required qualify as a token. From analysis, if a word occurred minimum 4 times in the dataset – it was tokenized.



Finally, the reviews must be padded/trimmed. Looking at the review length distribution graph – length of 50 words were chosen to minimize review cutoffs.

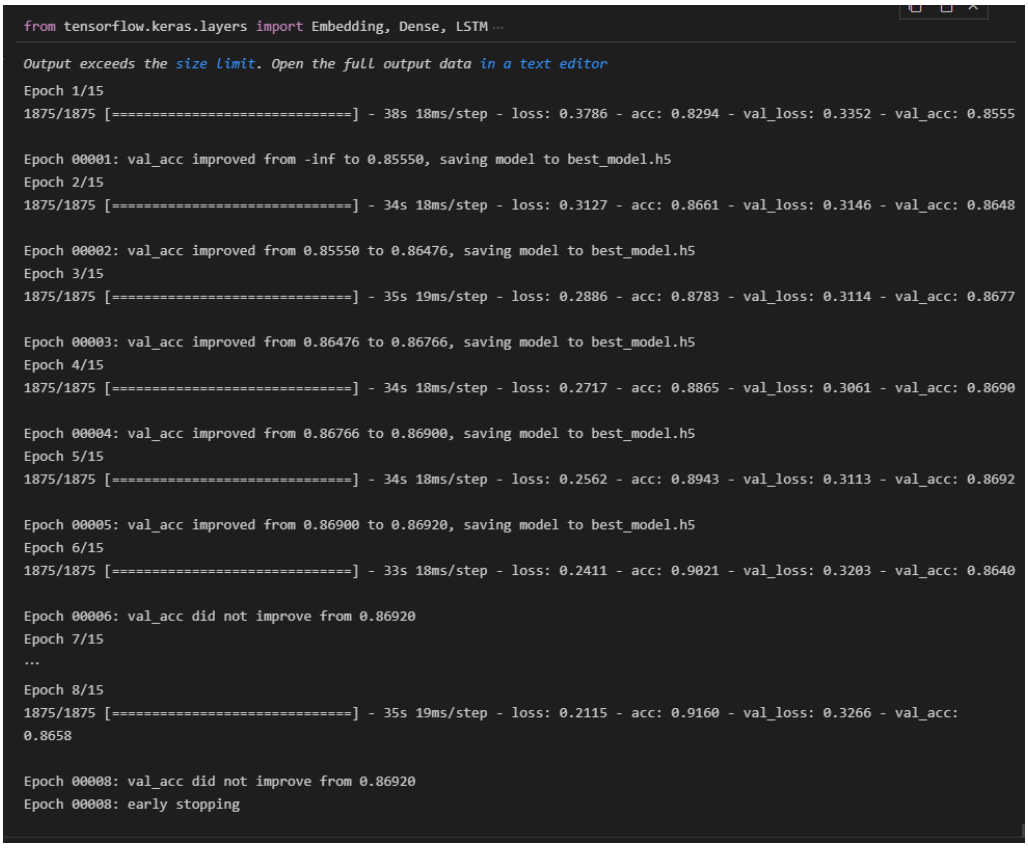


The RNN architecture of the model used is briefly explained below. Inspiration of the architecture was found (https://www.kaggle.com/nilanml/imdb-review-deep-model-94-89-accuracy)

1. Model is to start with an input layer from the tokenized dataset.
2. The model is to utilize LSTM after its embedding layer. From looking at the dataset, the chaotic sentence structure of some reviews to greatly benefit from LSTM's "context storage" capability.
3. The model would then need to be flattened so that it can be fed into the dense layer.
4. A dropout layer was added after that to reduce sample variance.
5. As this is a binary classification problem, the model would need a logistic regression. As a result, sigmoid activation function was used as the final layer of the model.

Hyperparameter tuning was conducted afterwards for accuracy optimization.

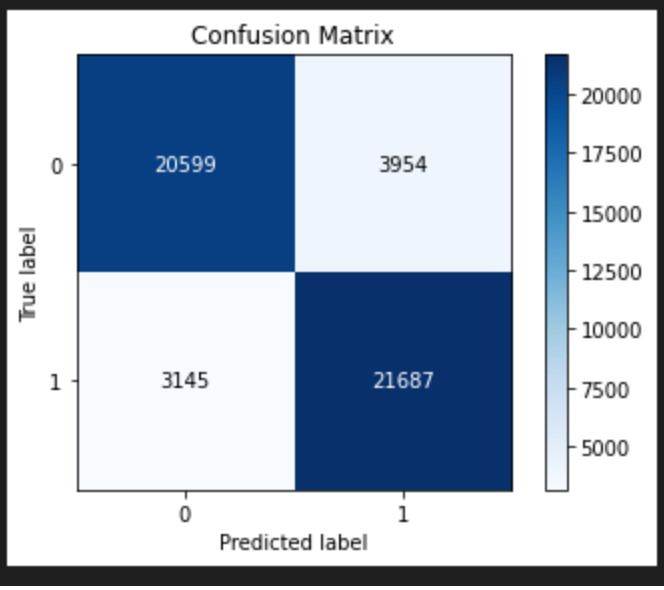
Insert model architecture



Summary of Dataset

The model gave an accuracy of ~86%. This is an improvement of similar works done in the past utilizing NLP for Korean sentiments that saw an accuracy of 84~85%. The more modern utilization of LSTM compared to regular RNN/dense model is thought to have made this improvement possible. Some English sentiment analysis models see accuracy >95% but due to differences in language/structure, Korean sentiment analysis models have yet to show such high accuracy. Also, due to the vast difference in the number of English speakers vs. Korean speakers, support tools for English NLP are much more polished with much more contributors (NLTK is much vaster and more developed than any current Korean language toolkit). However, I am optimistic that small, consistent improvements by numerous contributors will eventually allow Korean language models to also reach high levels of accuracy similar to its English counterpart.

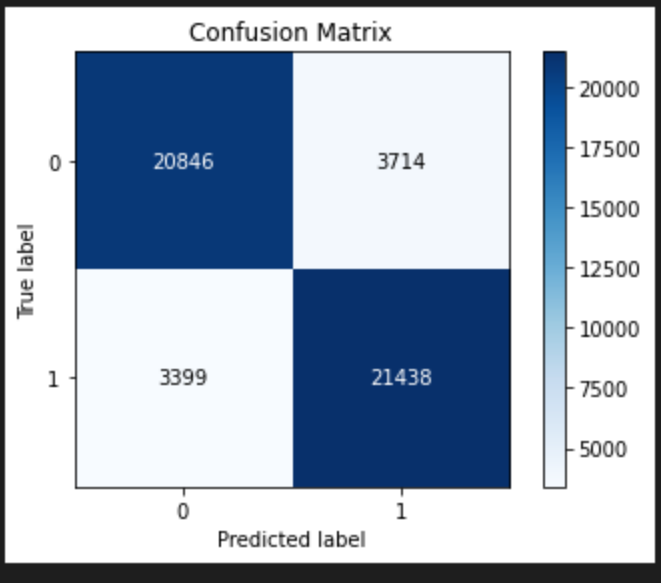
```
Accuracy_score:  0.8562518983497013
Precision_score:  0.8457938457938458
Recall_score:  0.8733489046391752
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.87      0.84      0.85     24553
           1       0.85      0.87      0.86     24832

    accuracy                           0.86     49385
   macro avg       0.86      0.86      0.86     49385
weighted avg       0.86      0.86      0.86     49385
```



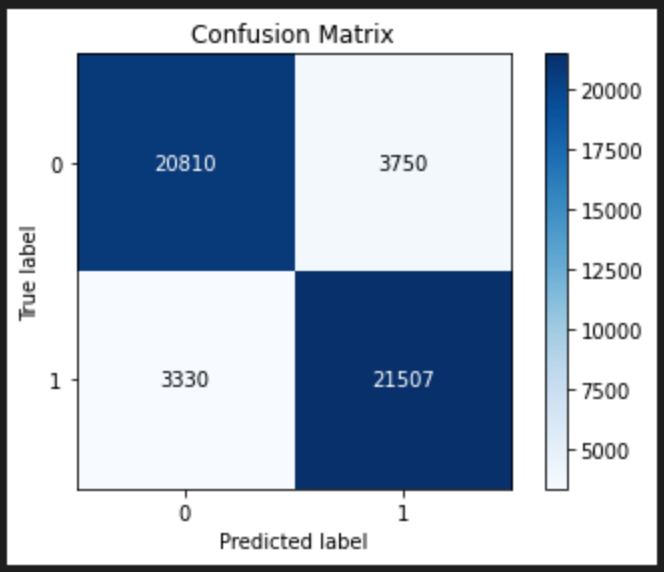
```
Accuracy_score:  0.856003401016256
Precision_score:  0.8523377862595419
Recall_score:  0.8631477231549705
```

---

	precision	recall	f1-score	support
0	0.86	0.85	0.85	24560
1	0.85	0.86	0.86	24837
accuracy			0.86	49397
macro avg	0.86	0.86	0.86	49397
weighted avg	0.86	0.86	0.86	49397



```
Accuracy_score:  0.8566714577808369
Precision_score:  0.8515263095379498
Recall_score:  0.8659258364536779
-------------------------------------------------
              precision    recall  f1-score   support

           0       0.86      0.85      0.85     24560
           1       0.85      0.87      0.86     24837

    accuracy                           0.86     49397
   macro avg       0.86      0.86      0.86     49397
weighted avg       0.86      0.86      0.86     49397
```



Model: "sequential_2"

_____

Layer (type) Output Shape Param #

==========

embedding_2 (Embedding) (None, None, 128) 1191424

_____

bidirectional_2 (Bidirection (None, None, 128) 98816

_____

global_max_pooling1d_2 (Glob (None, 128) 0

_____

dense_4 (Dense) (None, 32) 4128

_____

dropout_2 (Dropout) (None, 32) 0

_____

dense_5 (Dense) (None, 1) 33

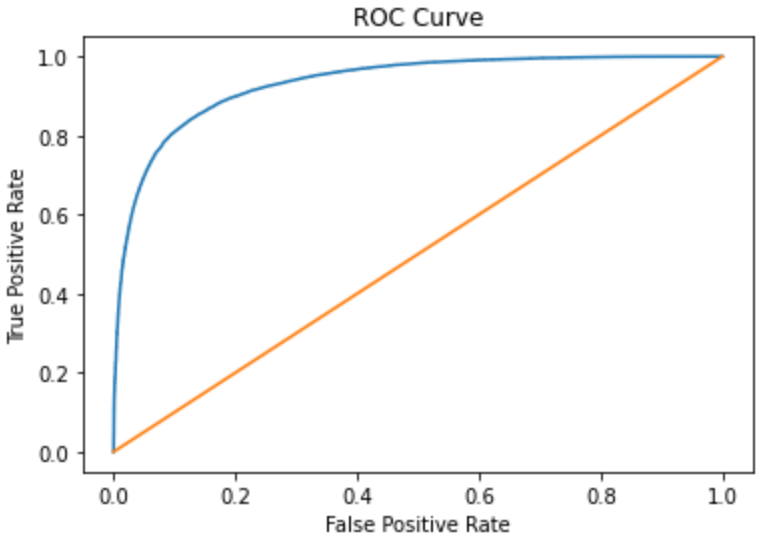
==========

Total params: 1,294,401

Trainable params: 1,294,401

Non-trainable params: 0

```
model.summary()
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 128)         1191424
_________________________________________________________________
bidirectional_2 (Bidirection (None, None, 128)         98816
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,294,401
Trainable params: 1,294,401
Non-trainable params: 0
_________________________________________________________________
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0