기계학습: Term Project

스팸 필터링: 나이브 베이즈 기반 LSTM 필터링 모델

2016130927 박준영

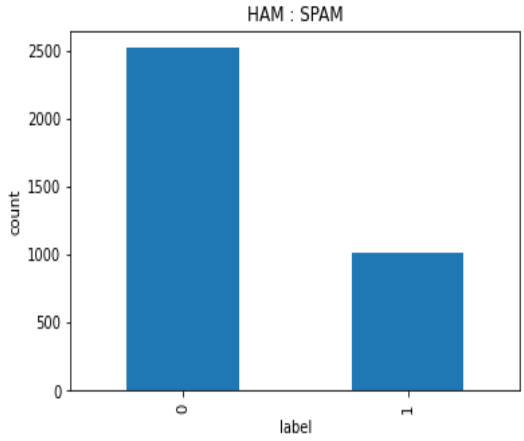
I. 실행 환경

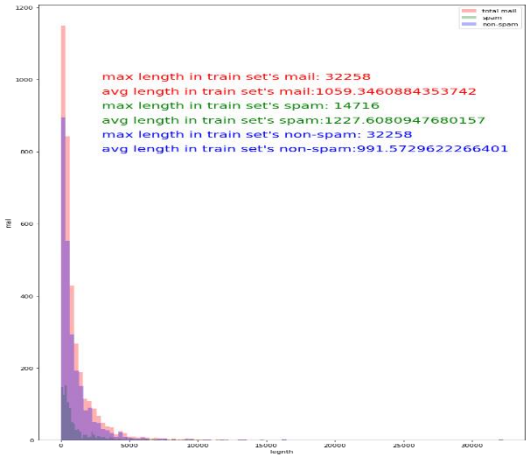
```
 !pip install contractions

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import collections
import contractions
import re
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVecto
rizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report,  p
lot_confusion_matrix

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('wordnet')

import keras
from keras.layers import Dense, Embedding, LSTM, Dropout
from keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

위 모듈을 import하고 실행하였다. 보다 상세한 사항은 함께 첨부한 requirements.text에 기록되어 있다.

II. 데이터 분석 및 전처리

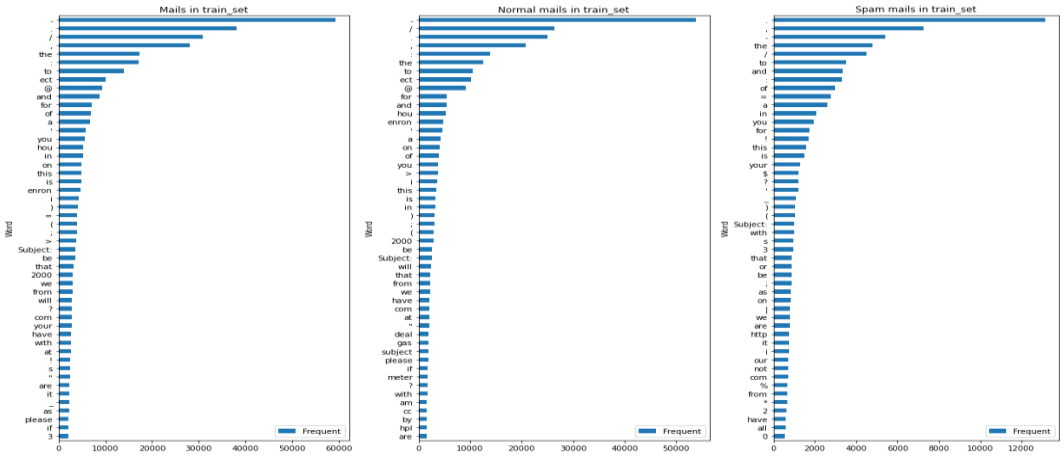
i 스팸 비율 시각화





주어진 데이터 Train_set[‘mail’] 중 스팸 대 정상 메일의 비율은 1:2.5 정도이다. 메일 역시 종류에 따라 길이 분포가 매우 다르다. 즉 데이터 분포가 고르지 못한 경우이므로, 이후 train_input과 val_input을 분리할 때 종류에 맞추어 stratify 조건에 추가할 필요가 있다. 메일의 내용 또한 종류별로 시각화할 수 있다.

ii. 단어 빈도 히스토그램



데이터 내 메일에서 가장 흔하게 등장하는 단어 50개를 도식화하였다. 전반적으로 의미를 알기 어려운 단어(‘.’, ‘/’ 등), 불용어(the, to, a), 특수문자(‘@’, ‘%’) 등이 존재한다. 정상 메일과 스팸 메일 간 주로 사용되는 단어 역시 차이가 있다. 스팸 메일에서는 ‘$’, ‘http’ 등을 정상 메일보다 빈번하게 확인할 수 있다. 하지만 위 정보만으로는 스팸 및 정상 메일을 구별하기 불충분하다고 여겨지는데, 텍스트를 기준에 따라 처리해 차이점을 다룰 필요가 있다.

iii. 데이터 클리닝

다양한 클리닝 기준 중 본 모델에서 택한 기준을 택했다.

(1). 약어 정규화: ‘I’m’과 같은 축약 표현을 ‘I am’과 같은 정규 표현으로 만든다.

(2). 소문자화: 대소문자를 구별하지 않고 모두 소문자로 표현한다.

(3). 특수문자 및 숫자 제거: '@', '$' 등 특수문자와 숫자를 제거한다.

(4). 불용어 제거: 'a', 'the' 등 의미적으로 영향력이 작은 불용어를 제거한다.

(5). 표제어 사용: 'watches' 등 변형 단어를 'watch' 등 본 품사 단어로 만든다.

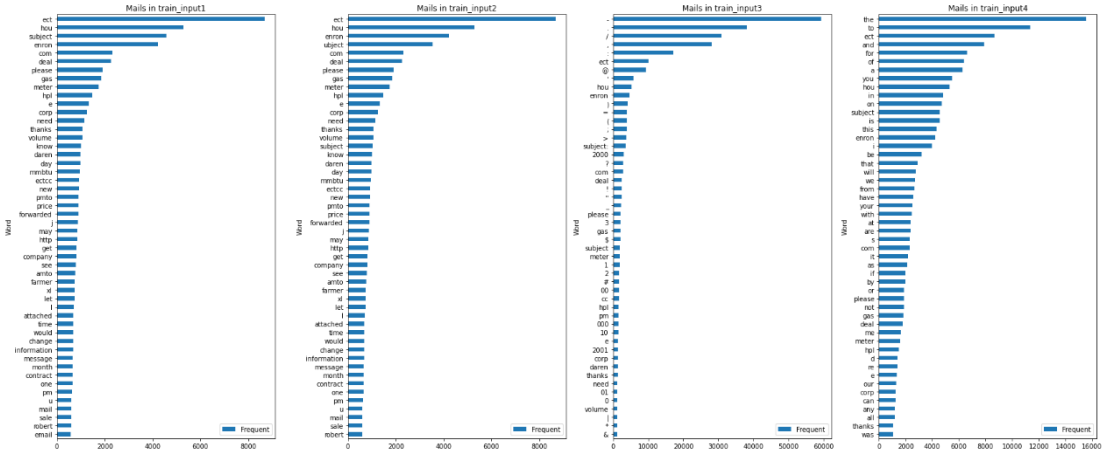
(1), (2), (5)로 동일한 단어임에도 다르게 인식되는 경우를 줄이고, (3)으로 알파벳 단어 위주 해석을 위주로 하려 했다. 특히 단어 빈도 히스토그램에서 확인한 바와 같이, 텍스트 중 절대적인 부분을 (4) 불용어가 차지한다는 점에서 기준을 추가했다. 이중 의미적으로 시험해볼 만하다고 간주한 네 가지 경우의 메일 데이터를 만들어 사용할 것이다.

(1). Input1: 위 다섯 가지 조건을 모두 적용하였다.

(2). Input2: 소문자화만 하지 않았다.

(3). Input3: 특수문자 및 숫자만 제거하지 않았다.

(4). Input4: 불용어를 제거하지 않았고, 표제어를 적용하지 않았다.



iv. 토큰화 기법

텍스트 클리닝 이후 학습에 사용하기 위해 데이터를 정제한다. 나이브 베이즈 및 로지스틱 회귀를 위해서는 각각 Count Vectorizer, TF -ID Vectorizer를 사용하고, LSTM 모델 학습을 위해서는 Tokenizer를 사용한다.

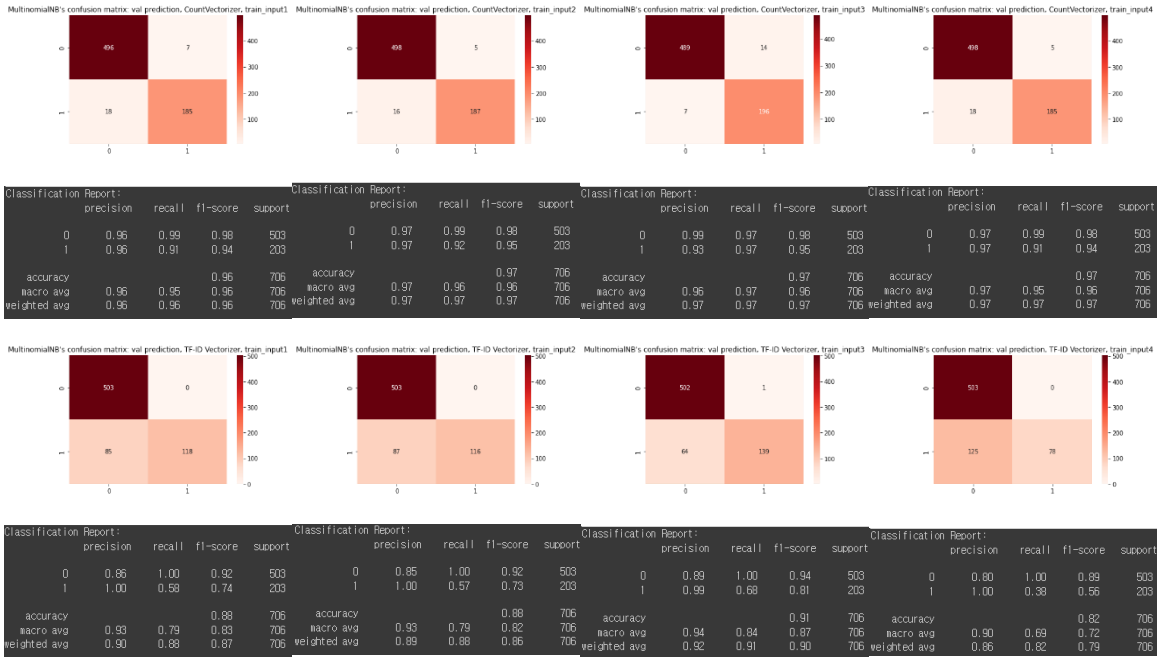
(1). Count Vectorizer: 단어 개수를 측정, 빈도에 따라 중요하게 인식한다.

(2). TF-IDF Vectorizer: 모든 라벨에서 등장하는 단어에는 가중치를 적게, 각 라벨에서만 등장하는 단어에는 가중치를 많이 준다.

(3). Tokenizer, padding: 텍스트를 시퀀스로 만들고, 가장 긴 길이를 기준으로 패딩한다.

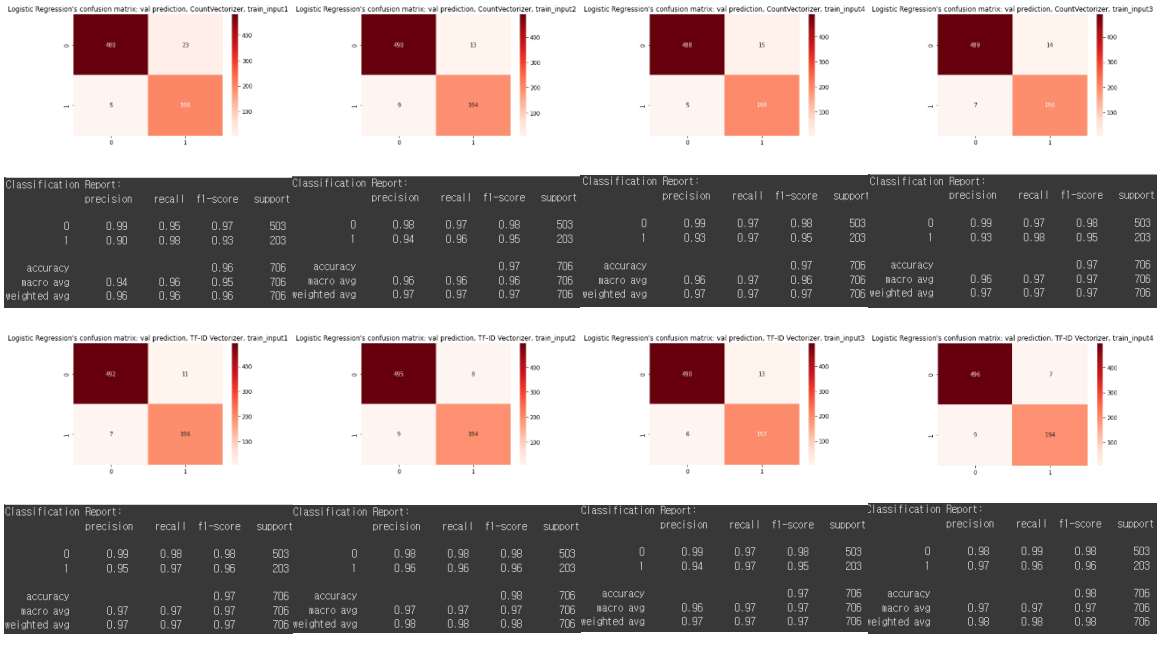
III. ML/DL 모델 및 성능 평가

i 나이브 베이즈 multinomialNB



동일한 Vectorizer를 사용한 경우 train_input 종류에 따른 성능 차는 보이지 않지만, Vectorizer 종류에 따라 성능 차이를 확인할 수 있다. 먼저 전반적 f1-score는 Count Vectorizer를 사용한 경우가 더 높다. 하지만 TF -ID Vectorizer에서 매우 높은 라벨=0, 즉 non-spam 예측 성공률을 보인다. 가령 train_input4를 TF-ID Vectorizer를 사용한 값을 나이브 베이즈로 훈련했을 때 val_input4가 분류한 정상 메일이 실제로 스팸인 개수는 0이다. 즉 나이브 베이즈 TF -ID Vectorizer를 정상 메일 분류에 사용한 경우 높은 확률로 틀리지 않을 수 있다.

ii. 로지스틱 회귀



나이브 베이즈와 마찬가지로 전반적으로 성능이 뛰어나고, 동일한 train_input을 사용한 경우 Count Vectorizer보다 TF-ID Vectorizer를 사용한 경우에서 전반적인 f1-score가 더 높다. TF-ID

Vectorizer를 사용했을 때 가장 뛰어난 성능을 보이는 모델은 train_input4(불용어, 표제어 제외)이다.

iii. LSTM

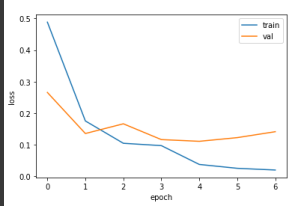
```
def create_model(max_words, max_length_sequence):

      lstm_model = Sequential()
      lstm_model.add(Embedding(max_words, 50, input_length=max_length_s
equence))
      lstm_model.add(LSTM(64))
      lstm_model.add(Dropout(0.3))
      lstm_model.add(Dense(20, activation="relu"))
      lstm_model.add(Dropout(0.3))
      lstm_model.add(Dense(1, activation = "sigmoid"))
      lstm_model.compile(loss = "binary_crossentropy", optimizer = "ada
m", metrics = ["accuracy"])
      lstm_model.summary()
      return lstm_model
```

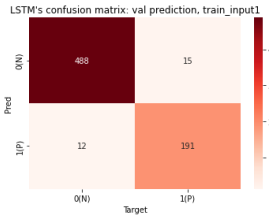
LSTM 모델을 통해 train_input을 종류별로 딥러닝한다. 중간 Dropout을 두 번 추가해 특정 인자에 가중치가 쏠리는 것을 방지했고, 활성화 함수는 relu와 sigmoid를 모두 사용했다. 이진 분류 작업이므로 바이너리 크로스엔트로피를 손실 함수로, Adam을 최적화 알고리즘으로 사용했다.

```
def lstm_run(lstm_model, train_padded, train_target, val_padded, val_ta
rget, test_padded):
    checkpoint_cb = keras.callbacks.ModelCheckpoint('best-lstm-
model.h5')
    early_stopping_cb = keras.callbacks.EarlyStopping(patience=2, resto
re_best_weights=True)
    history = lstm_model.fit(train_padded, train_target, epochs=100, ba
tch_size=64, validation_data=(val_padded, val_target), callbacks=[check
point_cb, early_stopping_cb])

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['train', 'val'])
    plt.show()

    return ((lstm_model.predict(val_padded) > 0.5).astype("int32"), (ls
tm_model.predict(test_padded) > 0.5).astype("int32"))
```

각 train_input을 인자로 받아 형성한 LSTM 모델을 실행할 때 체크포인트와 조기 종료를 추가했다. 이 경우 patience=2로 두 번 이상 validation_loss가 유효하게 감소하지 않을 경우 에포크를 중지한다. 따라서 전체 진행 에포크는 100으로 두었고, 유효한 성능을 보이지 않을 때 자동으로 종료할 것이다.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1472, 50)          2114900
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                29440
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                1300
_________________________________________________________________
dropout_3 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 21
=================================================================
Total params: 2,145,661
Trainable params: 2,145,661
Non-trainable params: 0
_________________________________________________________________
```

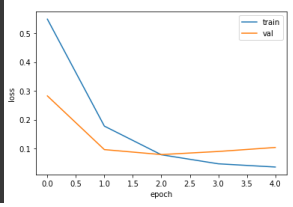


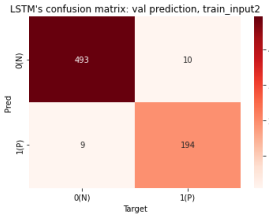



```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       503
           1       0.93      0.94      0.93       203

    accuracy                           0.96       706
   macro avg       0.95      0.96      0.95       706
weighted avg       0.96      0.96      0.96       706
```

```
Epoch 1/100
45/45 [==============================] - 12s 215ms/step - loss: 0.4885 - accuracy: 0.7682 - val_loss: 0.2657 - val_accuracy: 0.8938
Epoch 2/100
45/45 [==============================] - 9s 194ms/step - loss: 0.1753 - accuracy: 0.9575 - val_loss: 0.1355 - val_accuracy: 0.9547
Epoch 3/100
45/45 [==============================] - 9s 193ms/step - loss: 0.1046 - accuracy: 0.9738 - val_loss: 0.1663 - val_accuracy: 0.9178
Epoch 4/100
45/45 [==============================] - 9s 191ms/step - loss: 0.0972 - accuracy: 0.9752 - val_loss: 0.1163 - val_accuracy: 0.9603
Epoch 5/100
45/45 [==============================] - 9s 191ms/step - loss: 0.0375 - accuracy: 0.9904 - val_loss: 0.1109 - val_accuracy: 0.9618
Epoch 6/100
45/45 [==============================] - 9s 203ms/step - loss: 0.0253 - accuracy: 0.9954 - val_loss: 0.1225 - val_accuracy: 0.9632
Epoch 7/100
45/45 [==============================] - 9s 191ms/step - loss: 0.0198 - accuracy: 0.9947 - val_loss: 0.1413 - val_accuracy: 0.9589
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1474, 50)          2124050
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                29440
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 20)                1300
_________________________________________________________________
dropout_5 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 21
=================================================================
Total params: 2,154,811
Trainable params: 2,154,811
Non-trainable params: 0
_________________________________________________________________
```

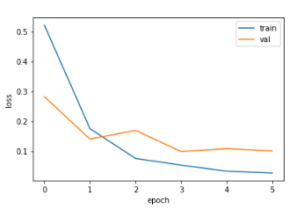


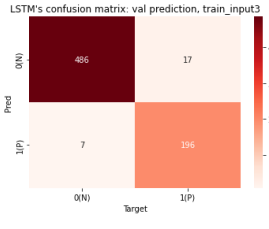



```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       503
           1       0.95      0.96      0.95       203

    accuracy                           0.97       706
   macro avg       0.97      0.97      0.97       706
weighted avg       0.97      0.97      0.97       706
```

```
Epoch 1/100
45/45 [==============================] - 12s 208ms/step - loss: 0.5492 - accuracy: 0.7434 - val_loss: 0.2825 - val_accuracy: 0.8768
Epoch 2/100
45/45 [==============================] - 9s 192ms/step - loss: 0.1776 - accuracy: 0.9497 - val_loss: 0.0960 - val_accuracy: 0.9632
Epoch 3/100
45/45 [==============================] - 9s 192ms/step - loss: 0.0781 - accuracy: 0.9770 - val_loss: 0.0787 - val_accuracy: 0.9731
Epoch 4/100
45/45 [==============================] - 9s 191ms/step - loss: 0.0467 - accuracy: 0.9883 - val_loss: 0.0895 - val_accuracy: 0.9688
Epoch 5/100
45/45 [==============================] - 9s 190ms/step - loss: 0.0354 - accuracy: 0.9911 - val_loss: 0.1034 - val_accuracy: 0.9632
```

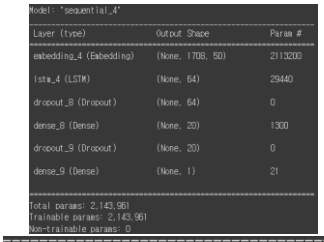
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 2018, 50)          1607150
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                29440
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 20)                1300
_________________________________________________________________
dropout_7 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 21
=================================================================
Total params: 1,637,911
Trainable params: 1,637,911
Non-trainable params: 0
_________________________________________________________________
```

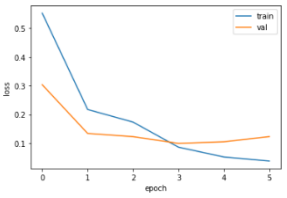


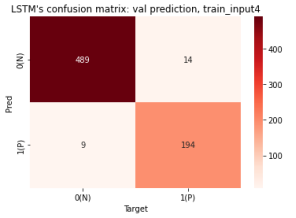


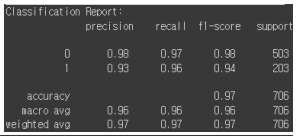

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       503
           1       0.92      0.97      0.94       203

    accuracy                           0.97       706
   macro avg       0.95      0.97      0.96       706
weighted avg       0.97      0.97      0.97       706
```

```
Epoch 1/100
45/45 [==============================] - 14s 265ms/step - loss: 0.5209 - accuracy: 0.7466 - val_loss: 0.2818 - val_accuracy: 0.9249
Epoch 2/100
45/45 [==============================] - 11s 249ms/step - loss: 0.1749 - accuracy: 0.9600 - val_loss: 0.1400 - val_accuracy: 0.9419
Epoch 3/100
45/45 [==============================] - 11s 248ms/step - loss: 0.0754 - accuracy: 0.9766 - val_loss: 0.1691 - val_accuracy: 0.9504
Epoch 4/100
45/45 [==============================] - 11s 248ms/step - loss: 0.0529 - accuracy: 0.9865 - val_loss: 0.0984 - val_accuracy: 0.9660
Epoch 5/100
45/45 [==============================] - 11s 247ms/step - loss: 0.0328 - accuracy: 0.9915 - val_loss: 0.1082 - val_accuracy: 0.9703
Epoch 6/100
45/45 [==============================] - 11s 248ms/step - loss: 0.0266 - accuracy: 0.9936 - val_loss: 0.1003 - val_accuracy: 0.9674
```










```
Epoch 1/100
45/45 [==============================] - 13s 232ms/step - loss: 0.5524 - accuracy: 0.7286 - val_loss: 0.3033 - val_accuracy: 0.8853
Epoch 2/100
45/45 [==============================] - 10s 227ms/step - loss: 0.2171 - accuracy: 0.9437 - val_loss: 0.1337 - val_accuracy: 0.9561
Epoch 3/100
45/45 [==============================] - 10s 217ms/step - loss: 0.1732 - accuracy: 0.9476 - val_loss: 0.1225 - val_accuracy: 0.9490
Epoch 4/100
45/45 [==============================] - 10s 216ms/step - loss: 0.0859 - accuracy: 0.9717 - val_loss: 0.0991 - val_accuracy: 0.9674
Epoch 5/100
45/45 [==============================] - 10s 216ms/step - loss: 0.0518 - accuracy: 0.9833 - val_loss: 0.1046 - val_accuracy: 0.9688
Epoch 6/100
45/45 [==============================] - 10s 216ms/step - loss: 0.0377 - accuracy: 0.9897 - val_loss: 0.1227 - val_accuracy: 0.9589
```

LSTM 모델은 train_input 종류에 관계없이 뛰어난 f1-score을 보이는데, 이중 train_input1을 제외한 경우가 특히 그렇다. 전체적인 train_loss는 유사하지만, val_loss를 기준으로 볼 때에는 train_input2, train_input4를 기준으로 모델링한 LSTM이 val_loss가 더 적다.

iv. 나이브 베이즈 기반 LSTM 필터링 모델(선정 모델)

앞서 나이브 베이즈, 로지스틱 회귀, LSTM 모델의 성능을 확인했다. 전체 f1-score 상으로서는 TF -ID Vectorizer을 사용한 로지스틱 회귀가 뛰어난 성능을 보였다. 하지만 TF -ID Vectorizer를 사용한 나이브 베이즈가 정상 메일을 분류하는 데에는 뛰어난 정확도를 보임을 확인할 수 있었기에, 이를 기반으로 다른 모델을 추가해 필터링할 수 있으리라 추측했다. 나이브 베이즈 mutlinomialNB이 사용할 TF-ID Vectorizer 기반 데이터는 train_input1, train_input2, train_input4 중 하나를 택하기로 결정했다(Confusion matrix를 확인했을 때 train_input3에서 나이브 베이즈가 스팸 메일 1개를 정상 메일로 간주했기 때문에 train_input3를 제외했다).

2차적으로 LSTM 모델이 예측한 라벨을 사용했다. 나이브 베이즈가 train_input1, train_input2, train_input4 중 골랐고, LSTM 모델이 train_input2, train_input4를 사용했을 때 성능이 뛰어난다.

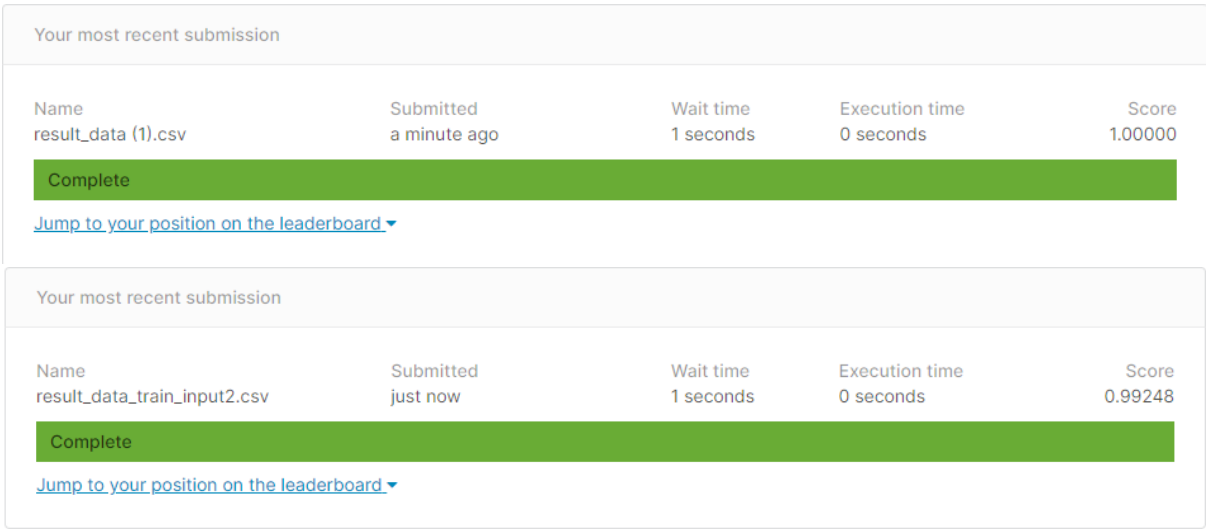
필터링을 위한 알고리즘은 간단히 앞서 구한 test_pred 값을 사용했다. 먼저 예측값이 나이브 베이즈 모델에서 정상 메일(label=0)이면 그대로 넘기고, 스팸 메일(label=1)일 경우에만 LSTM 모델을 예측한 라벨을 사용한다. 다음과 같이 train_input4를 기반으로 필터링, test_input4의 예측 라벨을 받아낸다.

```
nb_lstm_pred = np.array([])

for i in range(len(test_pred4_tf_nb)):
    if test_pred4_tf_nb[i] == 0:
        nb_lstm_pred = np.append(nb_lstm_pred, 0)
    else:
        nb_lstm_pred = np.append(nb_lstm_pred, test_pred4_lstm[i])
```

train_input2 또한 같은 원리이다.

```
nb_lstm_pred = np.array([])

for i in range(len(test_pred2_tf_nb)):
    if test_pred2_tf_nb[i] == 0:
        nb_lstm_pred = np.append(nb_lstm_pred, 0)
    else:
        nb_lstm_pred = np.append(nb_lstm_pred, test_pred2_lstm[i])
```

그런데 이를 통해 Kaggle score를 확인한 결과 f1 score에서 차이가 컸는데, 다음은 각각 train_input4, train_input2를 기반으로 한 필터링 모델의 score로 0.99248, 1.0으로 차이가 컸고, train_input4가 가장 준수한 성적을 거두었다. 이를 통해 필터링 모델의 train_input을 불용어 제거 및 표제어 처리를 하지 않은 네 번째 데이터를 기준으로 삼았다.



IV. 결과 분석

나이브 베이즈 기반 LSTM필터링 모델(이하 '필터링 모델')은 나이브 베이즈 모델 또는 LSTM 모델을 각각 사용한 경우보다 더 좋은 f1 score을 받았고, 선정 모델을 통해 Kaggle public score=1.0이라는 준수한 성적을 거뒀다. 이는 다음과 같은 요소에 기인하는 것으로 보인다.

i. 데이터 클리닝: train_input4(불용어, 표제어 제외)를 고른 기준은 나이브 베이즈와 LSTM에서 각각 뛰어난 성능을 보였기 때문이다. 나이브 베이즈는 train_input3에서, LSTM은 train_input1에서 다소 좋지 않은 성능을 보였기에 train_input2, train_input4 중 하나를 택했다. 전자는 소문자화를 하지 않았을 경우, 후자는 불용어 제거 및 표제어 적용을 하지 않은 경우인데, 이중 train_input4를 사용했다. Kaggle score를 통해 확인한 test score의 경우 train_input4가 train_input2를 사용한 score보다 현저히 높은데, 이는 표제어 처리가 본 텍스트 클리닝에서 유효한 결과를 거두지 못했기 때문이라 생각한다. 텍스트 클리닝에 필요한 품사 정보를 사전에 전달하지 못했고, 불용어를 제거하던 중 스팸 메일을 분류하는 데 있어 중요할 수 있는 데이터를 한꺼번에 제거한 가능성도 있다.

ii. TF-ID 벡터화 기반 나이브 베이즈 머신러닝: 1차 필터링(나이브 베이즈 multinomialNB)에서 가장 중요한 역할을 하는 벡터화 방법이다. TF-ID Vectorizer가 정상 메일, 스팸 메일 따로 나타나는

단어에 보다 가중치를 주었고, 나이브 베이즈를 통해 정상 메일을 정확하게 분류할 수 있었다. 이때 정상 메일을 스팸 메일로 간주할 수도 있지만, 이는 필터링을 통해 정확도를 높였다.

iii. LSTM 딥러닝: LSTM 모델 자체가 뛰어난 정확도 및 성능을 보였다. 1차 필터링 이후 그대로 LSTM 모델이 예측한 라벨을 사용한 까닭이다.

이처럼 나이브 베이즈 또는 LSTM 모델은 자체만으로도 뛰어난 성능을 보였지만, 특히 나이브 베이즈 모델이 TF -IF Vectorizer를 기반으로 사용한 경우 뛰어난 정상 메일 예측을 보인다는 점을 근거로 필터링 모델을 고안하였고, 이후 2차 필터링은 LSTM 딥러닝의 높은 성능을 통해 구현되었다.