Naive Bayes Classifier for Sentiment Analysis

Student ID: 2022311113
Name: 이소현

1. NaiveBayesClassifier

(1) Feature Selection

In the NaiveBayesClassifer, methods for feature selection exist. First, through 'extract_feature' method, all text can be converted to lowercase. Then, the method removes special characters, "!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~", tokenizes the text into words by splitting based on the spacing, and removes stop words.[1] After making word list through the process, a frequency dictionary which represents the number of appearances of a specific word is generated. Through the dictionary, 1000 most frequently appeared words can be selected. The selected features from train data are used in 'preprocess' method for extracting key features from both train and test data.

(2) Fit and prediction

After extracting key features, fitting starts. In 'fit' method, two types of dictionaries, 'post_dict' and 'neg_dict' are generated. 'post_dict' contains feature frequency in positive sentences, and 'neg_dict' contains feature frequency in negative sentences. 'prob' method can calculate probability for each class (positive or negative) of test data based on the dictionaries. To prevent returning zero, Laplace Smoothing was implemented. If a word in test data does not exist in both dictionaries, Laplace Smoothing assumes that the word appeared once in both dictionaries. In 'predict' method, positive probability and negative probability of a specific data are compared. Those probabilities were calculated as the frequency of a specific word per the length of each dictionary. If positive probability is higher than negative probability, the method predicts the class of the data as 1. If negative probability is higher, the method predicts the class as 0.

2. Model Training

Before start training, some data preprocessing took place. In the original datasets, classes were comprised of '5' and '1'. For binary classification, the work of converting '5' to 1 and '1' to 0 was done in advance. Thus, positive sentences were labeled as 1 and negative sentences were labeled as 0. After that, NaiveBayesClassfieir class was generated, and the train data and stop words were input to the model for training. Then, the test data were input to the trained model to check the model's performance. The model returns predicted labels about the test data.

[1] Ragnar, "All English Stopwords (700+)",<Kaggle> ,https://www.kaggle.com/datasets/rowhitswami/stopwords

3. Result

4. The performance of the model can be visualized through confusion matrix.

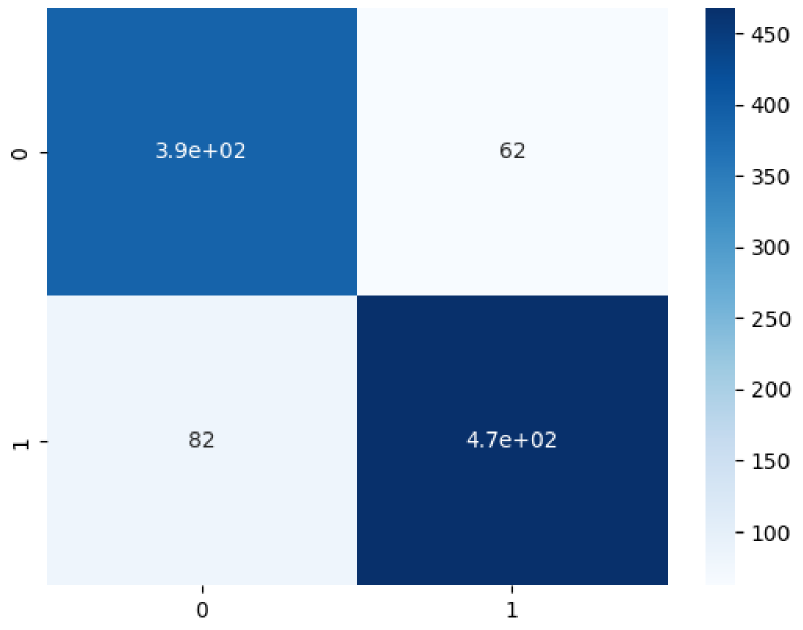

Fig 1

Its test accuracy was higher than 0.85, therefore the model might be considered as well-trained model.

When the amount of train data varies, the performance may also change. The learning curve by varying the amount of the train data used [10%, 30%, 50%, 70%, 100%] can be visualized as shown below.

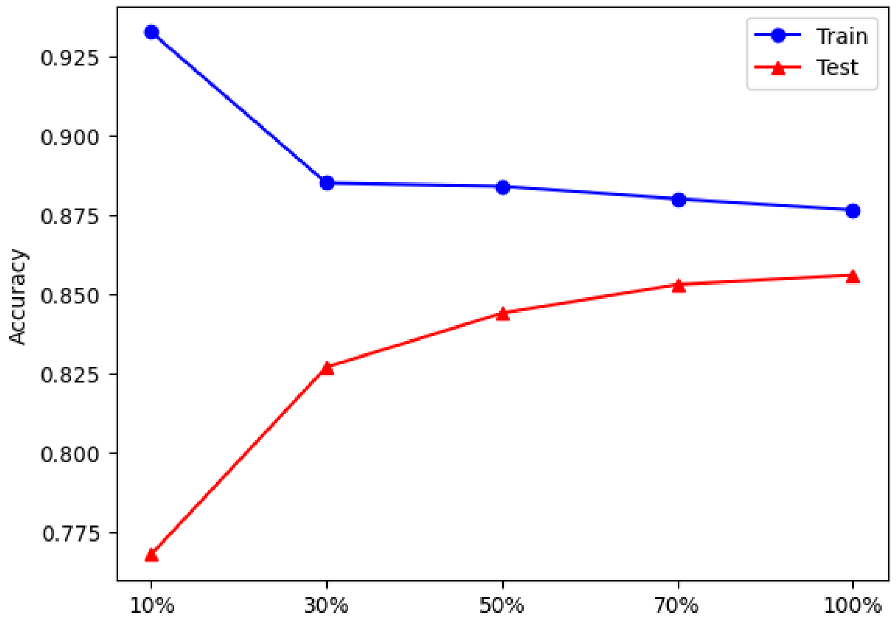

Fig 2

5. Discussion

It is important to get enough amount of train data to increase the model performance. When the amount of the train data was 10%, overfitting occurred where the score of the training data was overwhelmingly higher than that of the training data. As the amount increases, the difference in scores has narrowed. Therefore, using 100% of train data would be optimal.

However, after 70%, the increase in test accuracy was minimal It may mean that additional train data did not contribute significantly to performance improvement. To increase the model's performance, other methods would be needed to be found.

The method for tokenization was simple. Each sentence was tokenized based on spacing. Therefore, there might be some words which are not formal. It is expected that tokenizing words more properly would contribute to increasing test accuracy.