Comparing Naïve Bayes with Random Forest Spam Filtering

Naïve Bayes Introduction:

We have learned about the Naïve Bayes method for filtering spam messages in class. The Naïve Bayes method works based on the assumption that the input variables are independent of each other. Thus, we have:

(1) $P(X \mid Y_i) = \pi_n P(X_n \mid Y_i)$

Given the Bayes formula, we have:

(2) $P(Y_i \mid X) = P(X \mid Y_i) * P(Y_i) / P(X)$

(3) $P(X) = \Sigma_i P(X \wedge Y_1)$

Combining (1), (2), and (3), we have:

(4) $P(Y_i \mid X) = P(Yi) * \pi_n P(X_n \mid Y_i) / \Sigma_i P(Yi) * \pi_n P(X_n \mid Y_i)$

In our case, we also assume that Xi follows a Gaussian Distribution:

(5) $P(X_i \mid Y_j) = \exp(- (X_i - mean_{YjXi}) / (2 * stdev_{YjXi}^2)) / sqrt(2 * pi * stdev_{YjXi}^2)$

When training the model, we aim to find the mean and stdev for the corresponding i and j. We calculate the optimal mean and stdev by taking the derivative of the probabilities to these. We will skip the detailed calculations and provide the results:

(6) $mean_{YjXi} = \Sigma$ (X_i when X corresponds Y_j) / number of X corresponds to Y_j

(7) $stdev_{YjXi} = sqrt(\Sigma$ ($X_i - mean_{YjXi}$ when X corresponds Y_j) / (number of X corresponds to Y_j - 1))

We need to calculate $P(Y_i \mid X)$ for all labels and use the label corresponding to the largest probability when making predictions.

Random Forest Introduction:

Tree-based methods do not rely on any probability model. Instead, it solves a classification issue. Each non-terminal node splits the dataset into subsets based on their specific feature values. Each terminal node is associated with the dominating class in the dataset within that node. The pre-defined scoring function determines the feature to split on and the threshold. We have chosen Gini-index to be our scoring function, which has the following formula:

(1) $Gini = 1 - \Sigma_i (P_i)^2$

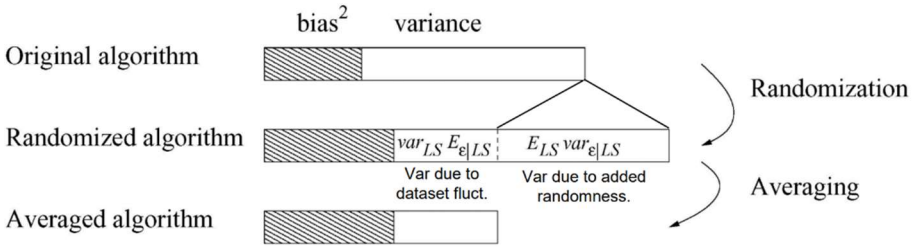
Gini index measures the probability of a particular variable being wrongly chosen. Gini index 0 means that there is only one class of variable, and Gini index of 1 means that the variables are randomly distributed. To make a prediction, we follow the tree until we reach a terminal node.

The concept of random forest is to train multiple trees on subsets of the dataset and split on subsets of the features. The trees in the forest vote to generate the final prediction. A

single tree model is very good at producing low-bias predictions but usually is terrible at producing low-variance predictions. The prediction a tree makes differs a lot if a tree is trained on different parts of the dataset. We can find a decomposition of error:

(2) Error(x) = Bias(x)^2 + Var[fD(x)] + Noise(x)

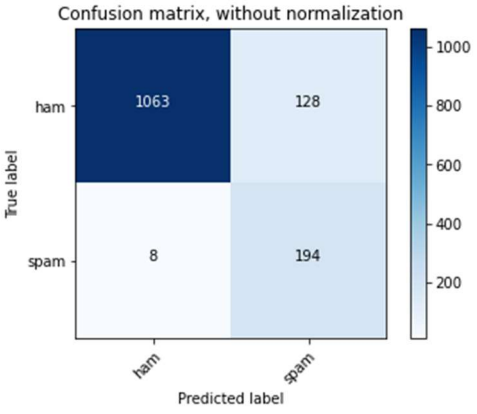
With different trees trained on different subsets of the dataset, we are transforming bias caused by data selection into bias generated by randomness. Then, with multiple trees, the bias caused by randomness is reduced.

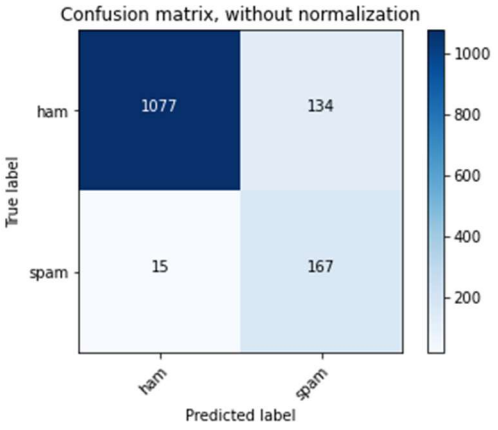


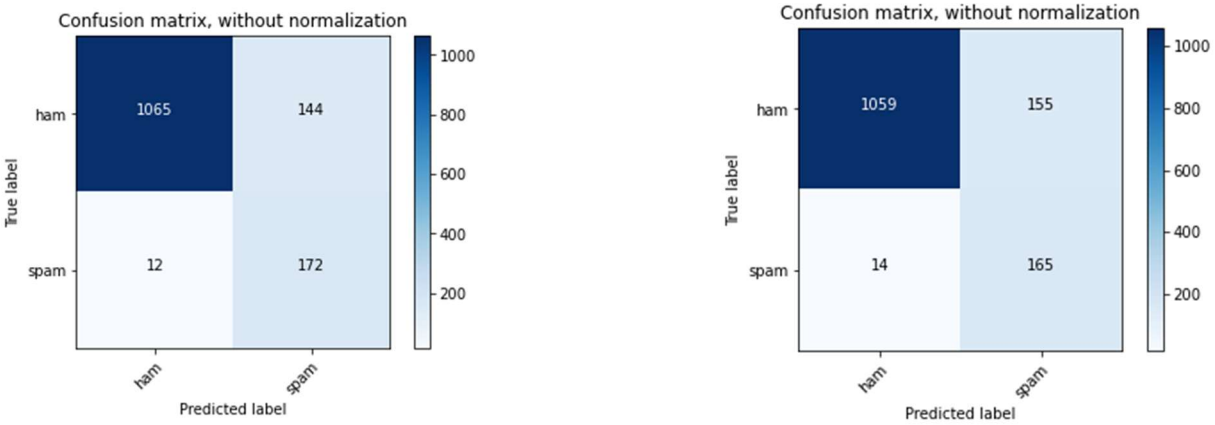
Experiments:

We acquired our dataset from the Kaggle website at https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset?resource=download. It contains 5574 SMS messages in English, labeled either Spam or Ham. We have written our code for the Naïve Bayes and Random Forest algorithms, but we relied on the Bag-Of-Word feature generation function from the Scikit-learn library.

With the Naïve Bayes algorithm, there is no hyperparameter to tune. However, to better assess the credibility of the algorithm, we still use the K-fold validation technic. With K set to four, we are using 75% of the dataset as the training set and 25% percent of the set as the validation set. We determine that the number of false predictions made in each of the four folds is: 136, 169, 156, and 149. Thus, the average accuracy of the Naïve Bayes model is about 89.05%. To better evaluate the performance of the model, we have also generated the confusion matrix for the model for the four folds:

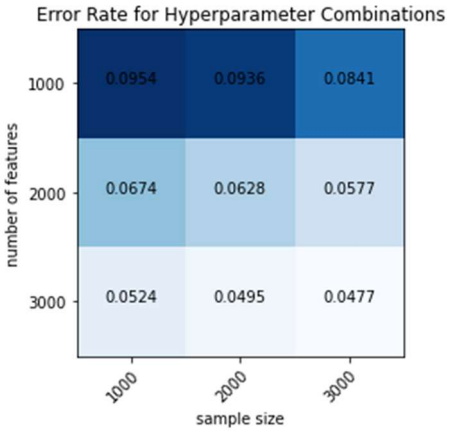




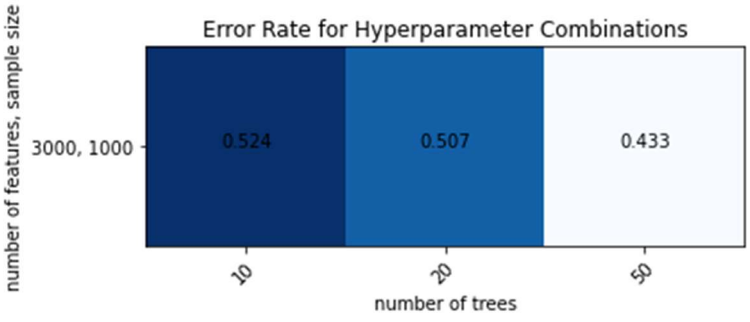


By observation, the Naïve Bayes algorithm has a false negative rate of 6.56% and a positive rate of 11.6%.

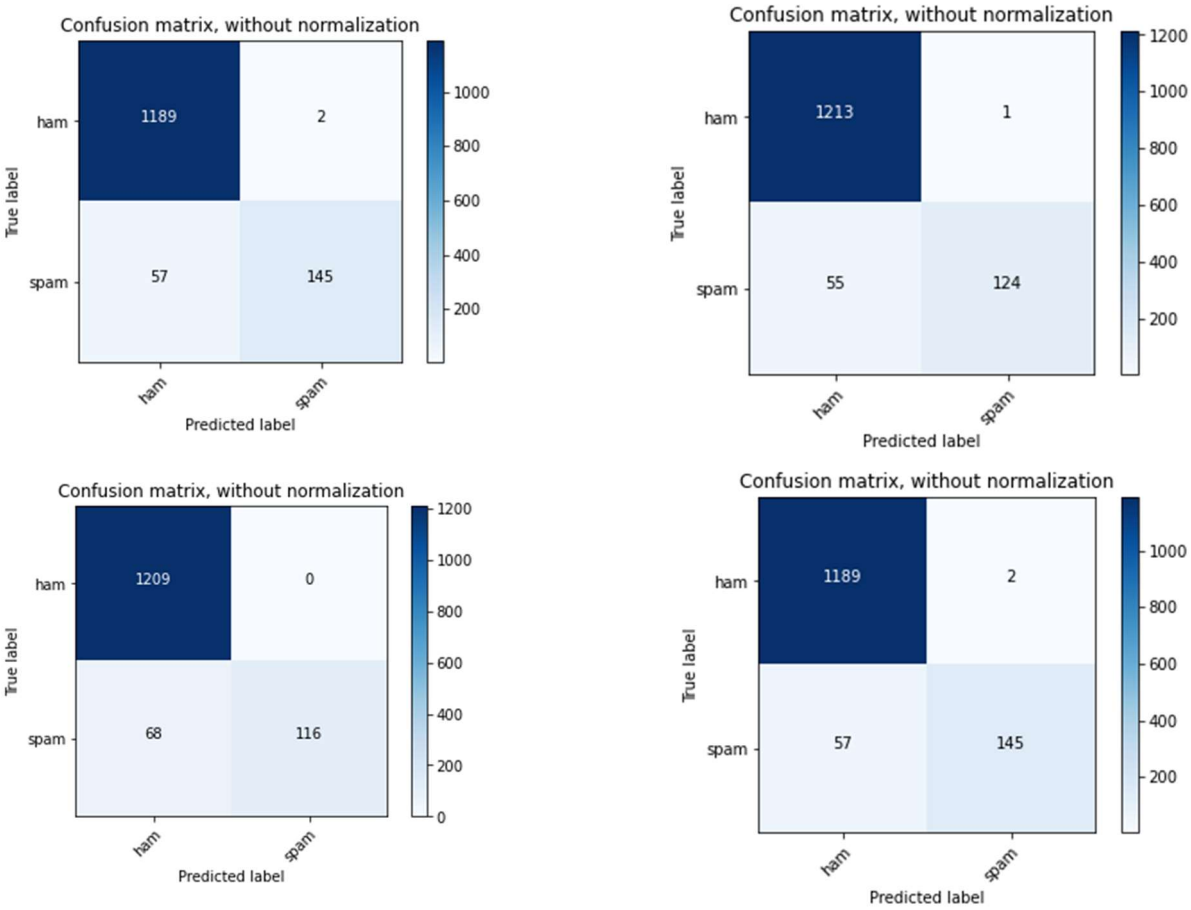
With the Random Forest algorithm, we have some hyperparameters to tune. Namely, we need to decide: the minimum number of samples in a non-terminal node, maximum depth of a single tree, number of trees, number of features a single tree split on, and sample size of a single tree. Given the complexity of training the random forest algorithm, we first try to pick the optimum number of samples for each tree(1000; 2000; 3000) and the optimal number of features a single tree split on(1000; 2000; 3000). At the same time, we are fixing the maximum depth of trees to 20, and the number of trees to ten. The error rate we get is:



Given the large training time for (number of features = 3000, sample size = 3000) is significantly large, we decide to use (number of features = 3000, sample size = 1000). Then, we investigate the influence of the number of trees on the error rate. We try a different number of trees from [10, 20, 50]. Here are the results:



Thus, we can decide that 3000, 1000, and 50 are the set of hyperparameters we want to choose. Let's examine the confusion matrix from the four folds of validation:



While The False Positive Rate is significantly lower than the Naïve Bayes model, we have a much larger False Negative Rate.

Conclusions

We have achieved a lower overall error rate than Naïve Bayes with Random Forest. However, the False Negative Rate increases significantly. We believe this is due to several factors:

(1) The Bag-Of-Words feature generation method generates sparse matrices, which is not ideal for tree-based methods.

(2) The dataset has far more Ham messages than Spam messages, making it more possible for tree leaves to vote for Ham.

If we continue to increase the depth or the number of trees in the forest, we may be able to improve the performance of the Random Forest Method. However, the computation power and memory needed to train trees are beyond our personal computers. Another direction would be to use better feature generation methods. The tree-based method will presumably work much better if the samples are represented with dense matrices.

References:

https://machinelearningmastery.com/implement-random-forest-scratch-python/

https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

https://python.plainenglish.io/roc-auc-in-machine-learning-d70d4308f636