COMPARISON OF NAIVE BAYES AND RANDOM FOREST SPAM FILTERING IN PERSONAL SMS MESSAGES

Han Gao,Yanbing Gong and Nan He[1]

[1]Department of Computer Science, University of North Carolina, Chapel Hill

Abstract

With large increase in the amount of SMS messages, spams are increasing day by day so the manual handling is not a feasible solution and it has become necessary to categorize them in different classes. Naive Bayes and Random Forest are two popular classifier algorithms and are implemented and compared in this paper.

Keywords: Naive Bayes, Random Forest, Anti-spam Filter

1 INTRODUCTION

With the increasing popularity and low cost of SMS messages in recent years, it has attracted the attention of businesses. Market messages can be sent to hundreds of recipients in a blind and unsolicited way at essentially no cost. Therefore, it's common for users to receive large number of spam messages and makes it a must to use machine learning methods to filter these spam messages.
In this paper, we design two classifiers based on Naive Bayes and Random Forest method separately, fine tune the hyperparameters and compare their performance based on a Kaggle Dataset.

2 NAIVE BAYESIAN CLASSIFIER

2.1 Introduction to Naive Bayes

The Naive Bayes method works based on the assumption that the input variables are independent of each other. Thus, given an input represented by $X = (X_1, X_2, ..., X_n)$, we have:

$$P(X|Y_k) = \prod_{i=1}^{n} P(X_i|Y_k) \tag{1}$$

Given the Bayes formula, we have:

$$P(Y_k|X) = \frac{P(X|Y_k)P(Y_k)}{P(X)} \tag{2}$$

$$P(X) = \sum_{k=1}^{m} P(X|Y_k) \tag{3}$$

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

$$P(Y_k|X) = \frac{P(Y_k)\prod_{i=1}^{n} P(X_i|Y_k)}{\sum_{k=1}^{m} P(Y_k)\prod_{i=1}^{n} P(X_i|Y_k)} \tag{4}$$

2.2 Gaussian Naive Bayes

In Gaussian Naive Bayes, we assume X_i follows a Gaussian Distribution:

$$p(X_i \mid Y_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(X_i - \mu_k)^2}{2\sigma_k^2}} \tag{5}$$

When training the model, our aim is to find the μ and σ for the corresponding i and k. The optimal μ and σ are calculated by taking the derivative of the probabilities with respect to these. We will skip the detailed calculations and provide results:

$$\mu_k = \frac{\sum_{X_i \sim Y_k} X_i}{N_{X_i \sim Y_k}} \tag{6}$$

$$\sigma_k = \sqrt{\frac{\sum_{X_i \sim Y_k} (X_i - \mu_k)^2}{N_{X_i \sim Y_k} - 1}} \tag{7}$$

where $N_{X_i \sim Y_k}$ denotes the number of X_i corresponds to Y_k

3 RANDOM FOREST CLASSIFIER

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

$$Gini = 1 - \sum_{i=1}^{n} (P_i)^2 \tag{8}$$

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:

$$e(x) = b(x)^2 + var(fD(x)) + \sigma(x) \tag{9}$$

where $e(x)$ denotes the error, $b(x)$ denotes the bias and $\sigma(x)$ denotes the noise function 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.

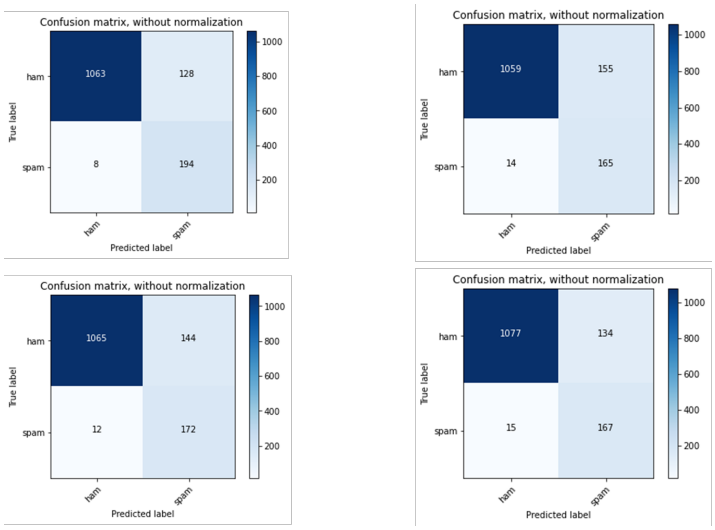

Figure 1. Confusion Matrix with Naive Bayes

4 EXPERIMENTS

4.1 Dataset Introduction

The dataset we acquired from Kaggle is the SMS Spam Collection Dataset (`https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset?resource=download`), which contains 5574 SMS messages in English, labeled either Spam or Ham.

4.2 Experiments of Naive Bayes Algorithm

With the Naïve Bayes algorithm, there is no hyperparameter to tune. However, to better access the credibility of the algorithm, we still use the K-fold validation technic. With K set to four, meaning that we are using 75% of the dataset as the training set and 25% percent of the set as validation set. We determines that the number of false predictions made in each of the four folds are: 136, 169, 156, 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 as is shown in Fig. 2.
By observation, the Naïve Bayes algorithm has a false negative rate of 6.56%, and has a false positive rate of 11.6%.

4.3 Experiments of Random Forest algorithm

With the Random Forest algorithm, we have some hyperparameters to tune. Namely, we need to decide: minimum size of set in non-terminal node, maximum depth of a single tree, number of trees, number of feature 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 feature a single tree split on(1000; 2000; 3000). At the same time, we are fixing the maximum depth of tree to 20, and number of trees to ten. The error rate we get are shown in Fig. 1.
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 continue to

investigate the influence of the number of trees on the error rate. We try different number of trees from [10, 20, 50]. The results is shown in Fig. 3.

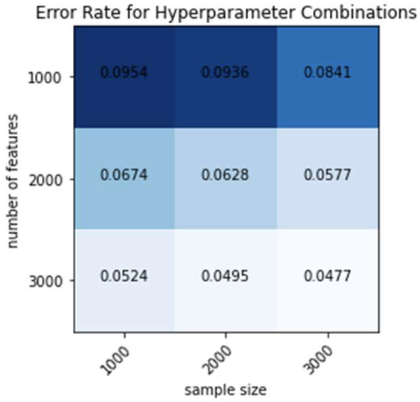

Figure 2. Error Rate with Number of Feature - Sample Size

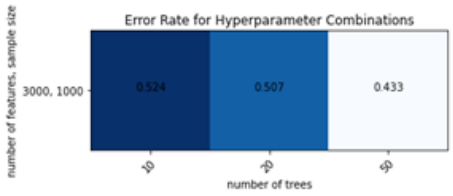

Figure 3. Error Rate with Number of Trees

Thus, we can decide that (3000, 1000, 50) is the set of hyperparameters we want to choose. Let's examine the confusion matrix from the four folds of validation in Fig. 4.

5 CONCLUSION

As mentioned above, we have achieved lower overall error rate then 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 technics generates sparce matrices, which is not ideal for tree-based methods. (2). The dataset has far more Ham messages than Spam messages, which makes it more possible for tree leaves to vote for Ham. (3). 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.

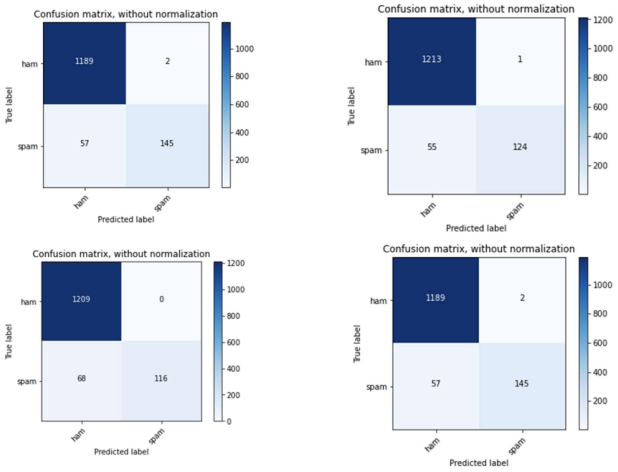

Figure 4. Confusion Matrix with Random Forest

6 REFERENCE

[1] https://machinelearningmastery.com/implement-random-forest-scratch-python/
[2] https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
[3] https://python.plainenglish.io/roc-auc-in-machine-learning-d70d4308f636