Spam SMS Message Classification: A Naïve Bayes Approach

Junda Li, Tongkai Zhang

1. Introduction

Nowadays, the burst in the volume of unwanted text messages has caused big problems in oud daily life. These unwanted SMS are classified as junk or spam messages. One of the negative impacts of spam messages is that they cause disruption and prevents the user from making full and good use of time. As a result, there is always a demand for the development of effective tool to classify received SMS. In this project, an attempt was made to evaluate how naïve bayes, a classical machine learning technique can handle this task.

2. Dataset and evaluation metrics

We use the SMS Spam Collection from Kaggle.com. The collection is a dataset of 5574 labeled SMS messages collected for SMS Spam study. All the SMS messages are in English and are labeled as ham (legitimate) or spam.

Evaluation metrics
To evaluate the performance of our machine-learning based classifier, the following evaluation metrices were used.

Accuracy: the ratio of the number of correct predictions to the total number of predictions.

False Positive Rate: the ratio of the number of non-spam SMS misclassified to the total number of non-SMS messages.

3. Data-preprocessing

The main goal of data-preprocessing is to extract the feature set of the data set. There are three tasks in data preprocessing: splitting the text of SMS messages into tokens and get useful information from the tokens by filtering and lemmatization.

3.1 Tokenize the text

Tokenizing is a quite important step in many NLP methods. In this project, we implement a tokenizer to parse the raw text. The easiest way in Python to parse a sentence is using the built-in split function splitting the string into tokens by whitespace characters, but it is difficult to separate the punctuation from its adjacent words.

And in our project, punctuation might be an informative feature, so it’s necessary to divide the punctuation from the word.

To solve this problem, we use regular expressions to design the tokenizer. Using the Python’s built-in re.findall function and create our own pattern for each token.

3.2 Filter the stop words

The purpose of filtering is to get the key information from the text and filter the useless tokens.

One problem for the tokenized text is that it contains many non-informative words, such as “the”, “it” and “to”. These words don’t provide any information about the message, which is called “stop words”. In this project, we want our text features to identify words that provide context(good or bad). Thus, we can remove the stop words from the tokens with the built-in stopwords data set provided by nltk.

3.3 Lemmatization

Lemmatization is the process of converting a word to its base form. Based on the context the word is used, it may have multiple different base forms. Therefore, identifying the POS tag can help us get some knowledge of the context and extract the appropriate lemma.

In this project, we use Wordnet to help us lemmatize each word. Wordnet is one of the earliest and most commonly used lemmatizers. We can pass in the Wordnet tag as an argument to tell the lemmatizer which context this word belongs to. Pos_tag function in nltk assigns each word with POS tags, so here we implement a tag converter to convert between these two tags. And then use WordNetLemmatizer to lemmatize the tokens

4. Exploratory Trials

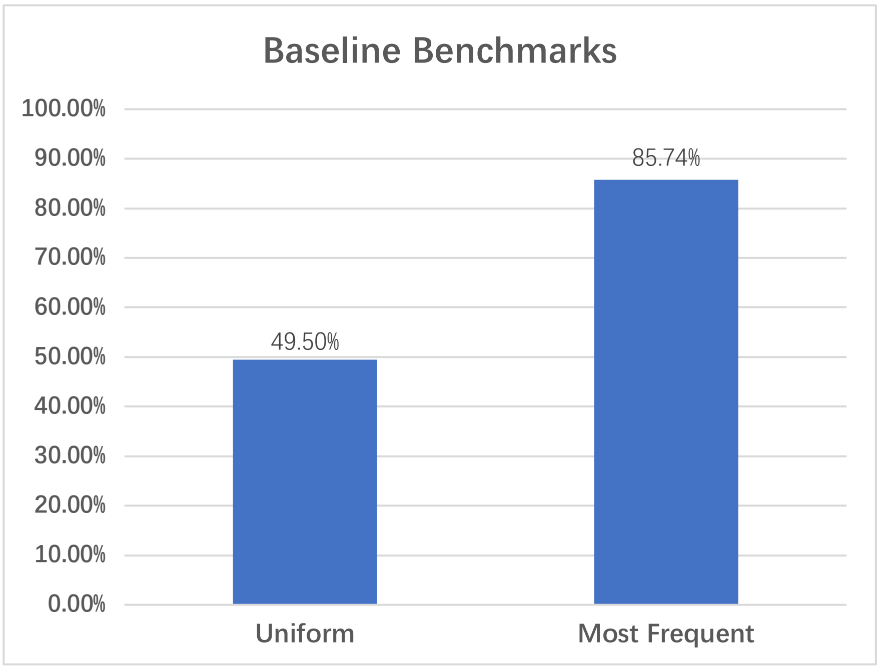
4.1 Baseline

A baseline is the method that use some simple and heuristics way to predict the label of SMS messages. It is a rule-based approach and does not involve complex machine learning. The purpose of baseline benchmark is to set up a lower boundary that we can compare the performance of our machine learning methods against. Two baseline methods were first explored.

Uniform: this is a way to predict the SMS according to uniform probability distribution of all possible types. In our dataset, each message has an equal probability to be judged as “spam” or “ham”.

Most frequent: this is a way that predict all SMS to be the most frequent category in the given datasets.

The benchmarks of these two baselines were shown below.



Comparison of baseline accuracy benchmarks

From the above graph, we can see that uniform baseline only give an accuracy of 49.50%. It is consistent with the expectation of a random guess outcome as it is closed to 50%. To our surprise, the most-frequent method yielded an accuracy of 85.74%, which is not bad. This suggests the dataset is not balanced.

As a result, we decided to use most-frequent baseline as the baseline for comparison in this project due to the unbalanced nature of the datasets.

4.2 Naïve Bayes Classifier

Naïve Bayes classifier is an effective way in text classification. It uses the bag-of-words model to identify text. According to Wikipedia, Naïve Bayes spam filtering is a baseline technique for dealing with spam that can tailor itself to the email needs of individual users and give low false positive spam detection rates that are generally acceptable to users. Therefore, we use Naïve Bayes classifier as one of the key methods to identify the spam messages.

The baseline Naïve Bayes method we used in this project is training the classifier using the tf-idf value of every word processed in part3 as feature set. The main libraries we use is the CountVectorizer and MultinomialNB from sklearn.

First create a train-test-split in order to verify the accuracy of the classifier when finished training. Then we use the CountVectorizer() to change the tokens in the training and test data set into vectors(bag of words). TfidTransformer in sklearn is used to turn the word count feature set into tf-idf feature set.

The tf-idf feature set used for training the classifier contains 3228 words tf-idf values from 4457 messages. The accuracy score for the classifier is 0.961, which is quite optimistic.

5. Optimization

5.1 Feature Engineering for Naïve Bayes Classifier

In part4, we use the bag-of-words model enhanced by using tf-idf to train the classifier and get such optimistic accuracy score. Here we experiment two optimizations based on the previous feature set:

1. Using bigrams to extract feature set
2. Using bigrams with positive pointwise mutual information to extract feature set

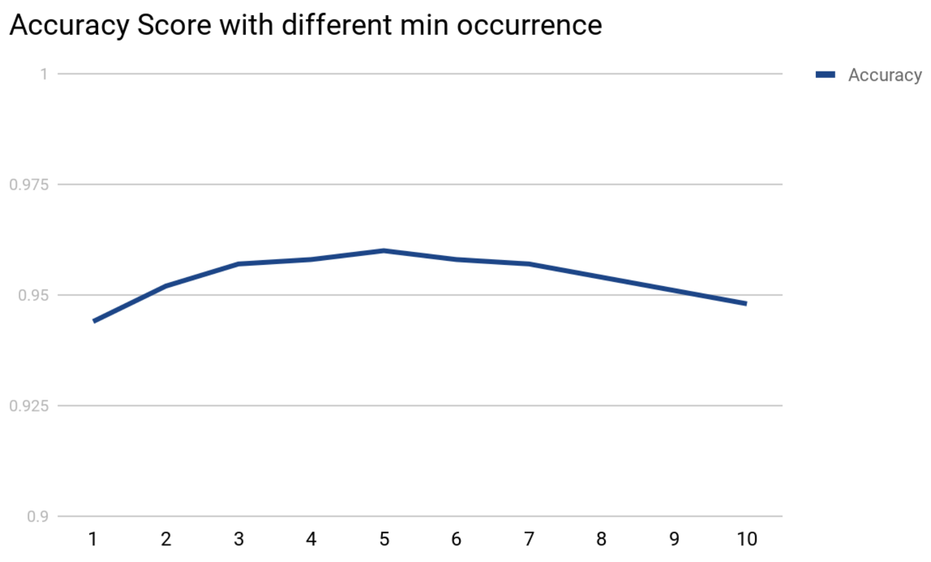
5.1.1 Bigrams as feature set

The basic idea for using bigrams is that bigrams is more informative than unigram feature set because they contain more context of the sentence. The assumption is that Naïve Bayes Classifier is more accurate using bigram than unigram.

We create the bigram feature set similarly as the unigram: extracting bigram list from the filtered token list, converting them into a vector and train the classifier using the bigram feature set.

However, the bigram feature set is much larger and sparser than unigram. Without any extra limitation(minimum occurrence allowed is the same as the unigram, which is 2), the accuracy score of using bigram feature set in our project from 4457 messages containing 6782 features is 0.952, which is lower than the unigram score.

To solve this problem, we try to find the minimum occurrence bar for a highest score, and the data is shown below. Given a minimum bound for each feature can shrink the size of the feature set effectively and also get rid of some noise. The result shows that during the training process, if we raise the minimum occurrence for each feature to 5, which means we discard all the bigrams with occurrence less than 5, we will get the highest accuracy score.



With the minimum occurrence limitation, the size of our bigram feature set is trimmed to 1213 and accuracy score is 0.960, which is quite close to the unigram score.

5.1.2 Positive PMI Bigrams as feature set

Pointwise mutual information(PMI) is a way of measuring the importance of the bigrams. It is a tool to find the collocations in the text and It can be calculated using the formula below.

$$\operatorname{pmi}(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)}.$$

Mathematical formula to get PMI(from Wikipedia)

According to the formula, we calculate the PMI for every bigram extracted in the above section and filter those bigram with negative or zero PMI values. Because non-positive PMI suggests that two words are not related. Therefore the main goal for this optimization is to get rid of the noise from the bigram feature set.

In our project, we use two dictionaries to keep the frequency of each word and bigram. Then we can calculate the probability of them to implement the PMI formula and filter the non-positive bigrams.

The size of PMI bigram feature set is 1104, which is 9% less than the original bigram feature set. The accuracy score of PMI bigram feature set is 0.961.

5.1.3 Feature engineering summary

These two approaches mainly deal with the feature set and as is shown above, the optimization in the final score is not obvious:

1. Bigram model with minimum occurrence of 5 scores 0.001 than unigram
2. Positive PMI bigram model with minimum occurrence of 5 scores the same as the unigram

However, the size of the feature set is decreasing when we process the optimization in feature set, which means we captured more informative information by relating to the context. After the feature engineering, the feature set is smaller and less sparse.

	Feature Size	Accuracy
Filtered Unigram	3228	0.961
Filtered Bigram	1213	0.96
Related Bigram	1104	0.961

Feature Size and accuracy of each model

5.2 Hyperparameter tuning for Naïve Bayes Classifier

Our work so far has shown that naïve bayes is very effective in dealing with spam classification problem. Nevertheless, has naïve bayes given its best performance in our previous trial? The answer may be not. In our previous trial, we haven't done any model tuning. In addition, we can also performance some feature engineering in our current feature set to obtain better result.

In the remaining part of project, we make attempts to optimize our naïve bayes classifier through model tuning and feature engineering.

5.2.1 Model tuning

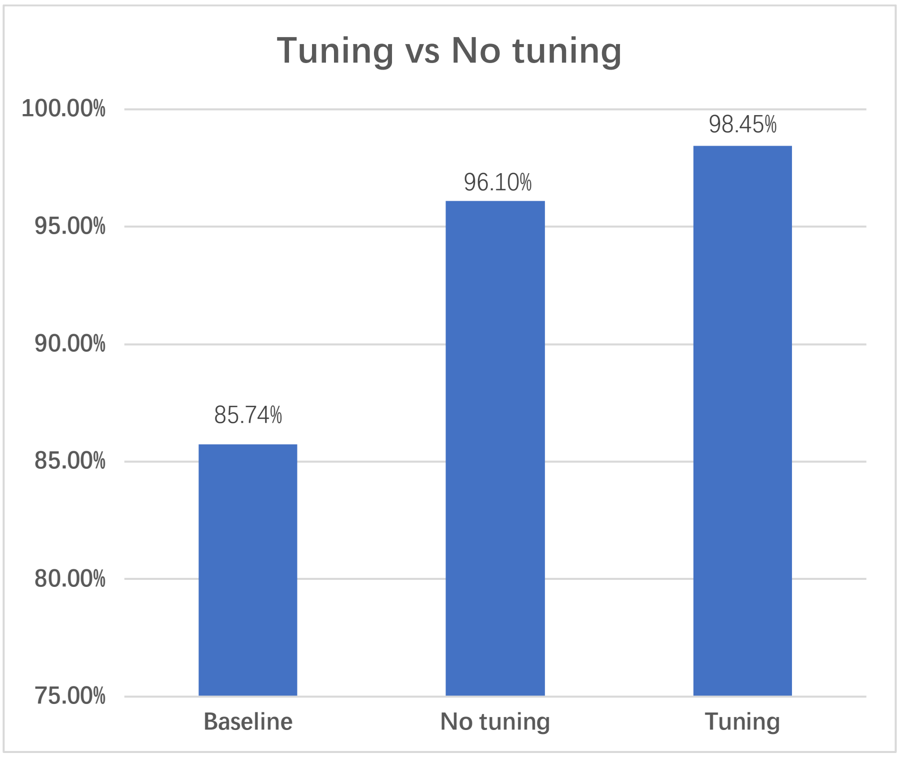
Model tuning is a process aiming to achieve the best performance of the machine learning method via searching for the best hyperparameters. Hyperparameters are the parameters that is external to the model. It is different from model parameter as it is about how to build up the model and must be manually set up by practitioner.

In naïve bayes method, one of the important hyperparameter is Alpha. Alpha is the parameter related to smoothing. Smoothing is a process to deal with the situation if a word/token in test set is not part of trainset. In such case, P(word) is zero and hence P(spam | word) or P(ham | word) will be undefined. To prevent this, an addition value called alpha is added to the numerator and denominator of P(word) so that it will not be zero but a tiny value.

In scikit-learn, the alpha value is set to 1 by default. We use grid search to iterate all possible values from 0.001 to 20 with a step of 0.001 to find out the best alpha value within that range.

The best alpha value we find is 0.12.

5.2.2 Accuracy Comparison



Comparison of accuracy of Naïve bayes with tuning or not

Using alpha = 0.12. The accuracy of the prediction is shown above. As we can see from the above graph, the accuracy increases from 96.10 % to 98.45% after tuning of the alpha parameter. Tuning of just alpha parameter gives an increase of 2.35% accuracy.

5.2.3 False Rate Comparison

Nevertheless, in the context of spam classification, we are also interested in one more metric, false positive rate. False positive has practical meaning as classifying non-spam message as spam will cause more serious problem than misclassify spam SMS as non-spam. A perfect classifier would be a one with highest accuracy and lowest false positive rate.

The average of false positive rate after 10-time cross validation is shown below.

Model	False Positive(%)
Naïve Bayes with tuning	0.21%
Naïve Bayes without tuning	0.00 %

Comparison of false positive rate of Naïve bayes with tuning or not

In contrast to the increase of accuracy after tuning, the false positive rate increases after tuning. While this may sound disappointed, 0.21 % is not a very large number and it is acceptable in real life. Nevertheless, the implication of this result is that accuracy should not be the only thing we should pay attention while evaluating machine learning methods.

5.3 Result with PMI bigram feature and model tuning

As is discussed above, we combine these two optimizations together. The score with PMI bigram feature set and model tuning is 0.963, which is lower than the tuning result of the unigram feature set(0.984) and higher than PMI bigram feature set(0.961).

6. Summary

Our project showed that naïve bayes works well in handling the problem of spam SMS messages. By appropriately data pre-processing(removing stop words, lemmatization), it can reach the accuracy score 98.45%(tuning). And our project also shows that due to the optimization result, bigram feature set can’t boost the performance for the classifier in our project. However, PMI filter can increase the bigram feature set accuracy.

In the future, we can try to explore more ways of extracting feature set like n-gram for naïve bayes model as well as more data filtering methods to get the most informative feature set. On the other hand, this project’s feature set is calculated from the training set. We can provide more realistic prior probabilities to the algorithm based on knowledge from daily practice to reduce the variance.