Spam Filter

Can Gundogdu

***Abstract*— Spam is unsolicited bulk messages, that is messages sent to multiple recipients who did not ask for them. Thus making it unnecessary to receive or to store. Spam filter becomes critical to conclude these issues and neglect the unwanted spam. It is one of the most useful filters by today. Spam filter can reduce the spam mail or messages by a huge efficiency. Therefore it is very anticipated technique in order to get rid of spam mails, spam SMS, or other message services. The application of spam filter could be used in different fields such as social networks , web filtering etc.**

I. Introduction

The purpose of the task is teaching various methods of classification and machine learning tools that was introduced in the lab and lecture sessions of the course. The analogy is to tune different classifiers for spam filtering problem and pick the one classifier that is most promising and apply further investigation.

The approach of spam filtering is based on constructing a vocabulary that contains words collected from the dataset for training our filter, then labels are assigned and weights are selected according to the occurrences of these words in the spam or ham (not spam) mails. By learning which words are often used in spam mails rather than ham mails, we can predict if a specific mail is spam or ham.

II. Methodology

A. Setup of experiments

There are two datasets available for the task. Which, one of them will be used to train the filter and other to test the filter. In the dataset there are set of files which are texts that represents emails and they are sorted in to ham or spam (in binary; ham = 0 , spam = 1) we will train classifiers with these datasets and we will try to predict if a text is ham (0) or spam (1) according to data that classifier learned from these texts.

Finding an optimal combination of classifiers, vectorizers and other parameters are the crucial part of this task. There are two script files created two solve the spam filtering. File "filter_template_test.py" is been used to configure classifiers and tuning their parameters easily. Both of the datasets were split in two equal pieces and both halves were concatenated and processed for training the filter (this method resulted in fluctuations in the accuracy because of random data selection) .Most of the parameters were found experimentally by trial and error however due to randomness of data caused by splitting we need to use Grid Search CV to adjust some parameters according to current training dataset. After the filter training the filter then is tested with modified accuracy to evaluate the filter.

B. Configuration of algorithms

Pipelines were used in all test cases due to it's effectiveness and every pipeline consists of Count Vectorizer, Tfidf transformer and a specific classifier. Count Vectorizer has been utilized for processing pregiven data and it builds a word counter. In the "filter_template_test" a custom word analyzer has been implemented to ensure increased filter accuracy. Tfidf transformer is used to improve the correctness of words weight's granted. Since the spam filter's approach is based on occurrences of words in a text thus making it essential to adjust the correct weights for the proper analysis.

The classifier is one the most important aspects of the pipeline. Different classifier's have different methods to classify thus it is crucial to select a robust classifier for this task. First stages of the task was spent on properly selecting a classifier. While adjusting the proper parameters for the classifier, Grid Search CV was used. This module takes a full advantage of trying parameter settings for the same classifier and prints out best score using those parameters. Which is very useful in this task.

After trying various classifiers such as SVC, kNeighbors and Decision Tree Classifier (these classifiers were eliminated) the highest accuracies were considered to be the most promising classifiers and those classifiers were selected for deeper experimentations.

III. Experiments

After selecting the candidate classifiers (MLPC , AdaBoost ,Multinomial Naïve Bayes), it is significant to tune the parameters of the classifiers. Count Vectorizer becomes very handy tool for achiving higher accuracy. It influeces the the filtering by preprocessing the words that are in the text and to improve the filtering, custom analyzer is implemented in "filter_template_test.py" Using the modified analyzer

considerably increased the performance of the filter.Custom analyzer uses English "stopwords" to filter the out the words that are shorter than two letters or the words that are in the NLTK (stopword english module) are discarded.The reason for this is teaching every word to the classifier might be misleading or decreasing the performance of the filter.NLTK stopwords are vocabulary of words that has no effect in determining a text whether spam or ham. For example words such as ; a, in, from, which, that, etc are included in the stopwords vocabulary provided by NLTK. Since it's a efficient way to filter, every classifier used this module to boost their performance.

The training data set is split uniquely everytime, this results in instability of the traning set. However this issue was solved by dynamically adjusting the parameter (max_df) which is responsible for ignoring words that are frequently used the in texts. These words can't be an indication of spam or ham that's why it is neglected.The max_df parameters should be around 0.7- 0.9 for best results.

Tfidf transformer is a great complement to Count Vectorizer.The parameter that effects the results is the "sublinear_tf" and declaring it True increased the overall accuracy of the filter by optimizing the wieghts.Changing other parameters were tried but they were not situmulating for better results.

Naïve Bayes classifier, MultinomialNB() is one the prominent classifiers for spam filtering.It usually has a high accuracy score (around %91) .And for this reason it was selected as a candidate.The parameter that needs to be adjusted is the alpha (additive smoothing parameter).During the testing stage, best values for the parameter was observed to be 0.001 .Training speed was very fast comparing it to other candidate classifiers.Filtering results were not fluctuating , it is considered to be stable.

AdaBoost classifier, AdaBoostClassifier() is the least promising classifier amongst the other candidates although it's accuracy was high (around %87) it was not enough to be the best classifier. Training speed was medium comparing to the other candidates. The parameter to be tuned is the n_estimators , the value should be around 250 or even higher for best results. Filtering results were stable during testing.

Neural Network classifier, Multilayer Perceptron Classifier (MLPC) is the best candidate amongst the other classifiers with the highest accuracy (around %96) . It has the slowest training speed, it takes significant time to select the proper parameters parameters. Although the results were the highest in accuracy the instability of the classifier is undeniable. It might give different accuracy values for the same training dataset. Nonetheless accuracy is higher than the other candidate classifiers. The reason for the slow processing is the Grid Search CV trying to find the optimal parameter to fit the data which takes time in repetitions. The neural network architecture parameters is 100,100,80,80,40 meaning that there are 5 hidden layers with 100 neurons in the first two hidden layers and 80 neurons in the next two hidden layers and finally 40 neurons in the last hidden layer. This setup seemed to present the best accuracy results. MLPClassifier was selected to be the best classifier and the final setup of the classifier can be found in the "filter_template.py". As previously mentioned the classifier consist of pipelines in order Count Vectorizer with English stopword implemented from NLTK library. Then tfid with sublinear_tf set to True , then the architecture of the classifier is MLPC((100,100,80,80,40)).This following setup gave the best results for the task.

Classifiers	Speed	Accuracy	Stability
AdaBoost	Medium	%87.1	Stable
Neural Network	Slow	%95.7	Unstable
MultinomialNB	Fast	%91.4	Stable

Figure 1. Classifiers tested for 10 repetitions and their average results

IV. Discussion

Training data is a major factor for better accuracy results. The training data has to have adequate quantity and quality for the best results. It also very important to optimally train the model to avoid overfitting or underfitting. To bypass these issues both of the datasets were split in to two and both halves were concatenated to train the model. Otherwise in the case of using full datasets without splitting caused almost %100 accuracy in the first dataset then a significant drop in the second dataset around %85. This is due to difference and inconsistency of data in the datasets and this problem was solved by mixing the datasets.

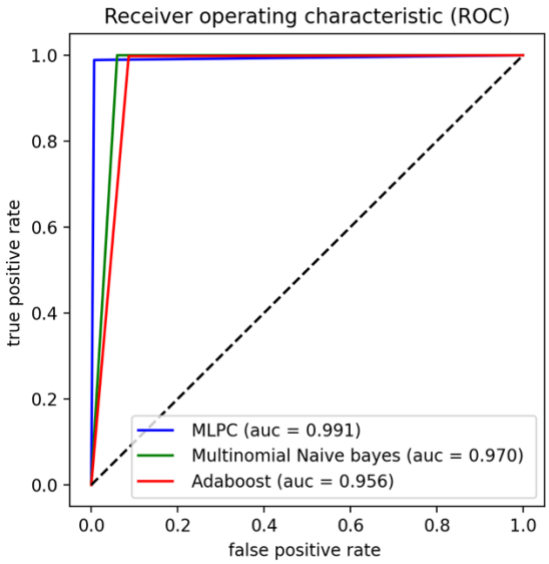

Figure 2. ROC curves of the candidate classifiers

V. Conclusion

Every classifier tried for this task differs from each other in various ways such as algorithms used , training speed , result stability and accuracy. Increasing the overall performance of the classifiers was handled by preprocessing and feature extraction of the dataset. In this paper Neural Network has achieved the highest accuracy. Although changing parameters gradually effect the results. Other two candidates Naïve Bayes and AdaBoost were more stable and faster than MLPC but could not reach the same accuracies. But with better tuning they might be considered to be better then MLPC due to robustness of these classifiers.

References

[1] Andreas C. Müller, Sarah Guido, “Introduction to Machine Learning with Python: A Guide for data Scientists” O’Reilly, 2016

[2] https://www.nltk.org/data.html (NLTK Dataset)

[3] https://scikit-learn.org/ (Classifier Information and feature extraction)

[4] Aurelien Geron , “Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow”, O’Reilly, 2017