Spam Filter

Mikhail Ivanov

***Abstract*— Spam filter is one of the most useful filters in the world today – amount of information generated is always increasing and without a tool that can sort the information by means of ham or spam, we would be spending a lot of our time sorting the income mails. Most of the people are facing on daily basis spam (=unwanted) types of messages they receive in their electronic and traditional mail boxes, via SMS and other message services. The same approach we use for e-mail spam filtering could be also used in many other fields like social network and SMS spam filtering, web-filtering etc.**

I. Assignment

This semestral task is aimed to teach us of using of different classification and machine learning tools we met at lectures and seminars. The idea is to try and tune different classifiers for spam filtering problem and pick the most appropriate one for further investigation and competition. Most of the tools are already presented in Numpy and Sklearn libraries for Python, we just need to test some of them and try different settings.

II. Introduction

The idea of spam filtering is based on making a vocabulary that contains words we gather from dataset available for training where correct labels assigned and weighting those words based on occurrence of them in spam or ham (not spam) mails. Knowing which words are more often found in a spam mails rather than ham mails, we can assume if a mail under test is spam or not.

III. Methodology

A. Setup of experiments

For the finding of an optimal combination of classifiers, vectorizer and other parameters, file filter_testing.py has been used. It represents an easy configurable file that allows us to tune testing parameters. Both datasets have been split in two equal parts, both halves have been concatenated and processed for filter training – this approach resulted some accuracy fluctuation due to dataset randomness. Most of the settings has been found empirically, however due to random nature of data splitting we need to keep Grid Search CV to tune some parameters based on current training dataset (especcialy max_df). After the training, the full datasets are tested and modified accuracy is used for spam filtering evaluation.

B. Configuration of algorithms

In all testing cases pipeline has been used, it always consists of CountVectorizer, TfidfTransformer and Classifier.

Count Vectorizer has been used for processing datasets we have and building of a word counter. Custom word analyzer as well as stop_words list has been tested in terms of improving filter accuracy.

TfidfTransformer is used for performance improving in a way of word weighting. The spam classification decision is mostly based on specific words occurrence in a mail, hence weighting tuning is quite important in our case.

Classifier, as well as pipeline parameters, are main variables we change during the testing stage. Different classifiers use different approach of data classification. Since our task is classify mails by Spam and Not Spam (ham), we need to choose quite robust classifier. During pre-testing stage different classifiers has been tested, three of them with the highest accuracy has been picked for further investigation and tuning. During the tuning stage, we defined hyper parameters that mainly influence the accuracy by using Grid Search CV module of sklearn. This module allows us to test the same tools but with different settings / parameters and print out the best score as well as specific parameter that accompany that. Neural Networks, Naïve Bayes and Adaboost were chosen as most promising solutions.

IV. Experiments

Count Vectorizer is a key tool in higher accuracy achieving. Word and vocabulary preprocessing have significant influence on filtering accuracy. Custom analyzer (can be found at filter_testing.py file) has been tested but didn't demonstrate significant improvement (it takes only alphabetic words with length more than 2 letters that are not an English stop word given in nltk dictionary). Using of English stop words has proven its efficacy, and this setting has been chosen as common parameter for all tests. Due to training set instability (it has been uniquely split every time), parameter max_df is needed to be dynamic and modified in accordance with the dataset properties (usually 0.7 – 0.9). This parameter let us ignore words with frequency higher than set – it means that we can rid of too frequently used words since it doesn't really tell us regarding presence of this word in a specific type of mail (spam or ham), hence it can't be used for classification.

TfidfTransformer in combination with CountVectorizer showed some accuracy increase. The parameter that we can play with is sublinear_tf, and by setting it to True we can slightly change the weighting algorithm. Changing of other parameters didn't demonstrate any positive performance change.

TABLE I
Filter comparison

Classifier	Training speed	Training stability	Accuracy, %
Adaboost	Moderate	Stable	84 - 88
Naïve Bayes	Fast	Stable	91 - 94
Neural Network	Slow	Unstable	95 - 99

Adaboost classifier, AdaBoostClassifier(), for some reasons showed the least filtering accuracy. Usually with higher n_estimators we can achieve better performance (equal to 250 or even more). The result of Adaboost classifier was stable for the same dataset, the training process speed was moderate, and accuracy is relatively low – around 84-88%.

Naïve Bayes classifier, MultinomialNB(), is one of the most popular classifier for spam filtering. Depending on features, it usually let us achieve high filtering accuracy. The main parameter we used to set during the test was additive smoothing parameter (the best result was achieved with 0.001). Training procedure was fast, filtering results were stable for the same dataset and accurate (91-94%).

Neural Networks classifier, MLPClassifier(), demonstrated the highest accuracy, however it required the largest amount of time to define the settings. It's also complicated by instability of training results – the same dataset can give different model and hence accuracy with absolutely the same settings. The training process is slow, accuracy is not stable and require Grid Search for repetitions in purpose to find the best fit, however its filtering capability is usually the highest among chosen classifiers (accuracy vary in range of 95-99% for concatenated halves of given datasets with (120, 120, 80, 80, 40) neural network architecture).

The final filter is presented in filter.py file and has been based on pipeline that includes CountVectorizer that consider English stop-words, TfidTransformer with sublinear_tf set and MLPClassifier() with architecture (120, 120, 80, 80, 40). MLPClassifier() has been chosen due to highest median (i.e. the most likeable) result. It's preferable to use all available data for training. In case we can't use all the data or it occurs overfitting and we need to split it, Grid Search CV is recommended to define max_df parameter among 0.7, 0.75, 0.8 due to random nature of split process. In case it's not possible to use Grid Search, value of 0.75 for max_df should be chosen, 30/70 ratio of test/training data is recommended. The result of 12 filter performance measurements and assesment results are presented in Fig.1, Table 1 and 2.

TABLE II
ACCURACY MEASUREMENTS, %

# of test	Neural Network	Naïve Bayes	Adaboost
1	0.9522	0.9371	0.8795
2	0.9308	0.9293	0.8884
3	0.9803	0.9224	0.8173
4	0.9688	0.9308	0.8341
5	0.9967	0.9522	0.8664
6	0.9466	0.9308	0.8691
7	0.9933	0.9371	0.8522
8	0.9590	0.9787	0.8991
9	0.9967	0.9052	0.8173
10	0.9482	0.9116	0.8824
11	0.9676	0.8944	0.8383
12	0.9688	0.9490	0.9006
Median	0.9682	0.9308	0.8664
St.Dev	0.0204	0.0215	0.0285

V. DISCUSSION

Amount and quality of training data is a key parameter of the training results. It's always a challenge to find an optimal dataset and avoid model underfitting and overfitting. In case of testing and classifiers properties learning, concatenated halves were chosen to represent data from both datasets. The problem is that taking one of two datasets in full, for some reasons we always achieve 100% accuracy on this dataset on most of classifiers and lower accuracy on second (from 80% for Adaboost to 95% for Neural Network). The reasons of that is possible inconsistency of the datasets. In case of assessment of the filter using different datasets, it will be reasonable to use all available data for the training (i.e. both datasets). In case it's not possible or cause overfitting, filter.py file includes code for taking only 70% or 50% of available data – please comment/uncomment appropriate sections.

VI. CONCLUSION

All the classifiers differ by means of classification algorithm, training speed, stability and overall accuracy. Data pre-processing and feature extraction is a crucial part of filter development as well as choosing appropriate dataset for training and testing. Neural Networks are promising solution for spam filtering, however, we need to be careful with setting its parameters. Naïve Bayes classifier use is usually faster in terms of training and accurate solution. Adaboost is also a recommended solution, however in this research we couldn't find appropriate settings to achieve sufficient accuracy.

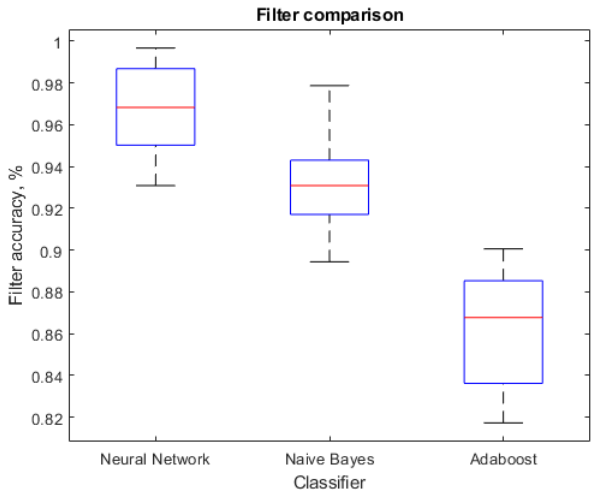

Fig. 1. Filter comparison.

REFERENCES

[1] *https://scikit-learn.org/*
[2] *Thanaki, J.* , (2017). Python Natural Language Processing. Birmingham, England : Packt Publishing..
[3] *A. Burkov*, (2018) "The hundred-page machine learning book" self-published.