B(E)3M33UI — Semestral project 1: Spam filter

Petr Pošík

March 24, 2020

1 Introduction

The goal of this task is to employ the knowledge of the ML lectures and previous exercises, and apply it to a text classification task – spam filtering. The goal is to create as good spam filter as possible, and describe its principle in a report.

2 Requirements

You have to submit the python scripts, which demonstrate what you have achieved, and a report describing the methods and results, i.e. submit

1. module `filter.py` (mandatory), containing a working filter of your choice, which will be used to assess the quality of your solution,
2. PDF file containing the report (mandatory), and
3. (optionally) other Python scripts, data, etc. used to create the report, especially the code that generates the results and figures used in the report, e.g. when comparing several filters. The code shall be organized such that it is clear what parts of the code are related to what parts of the report. These files will not be evaluated, they serve as a reference: if any part of the report would rise any doubts, these files will be consulted. If they are missing, the questionable part of the report may be evaluated with 0 points.

Remember to submit your own work! Do not commit plagiarism! The plagiarism detection feature will be on for this task, and any confirmed plagiat may be a reason to assign 0 points from this task! This applies to both the PDF reports and python scripts.

You can look at an example of text processing using scikit-learn.

2.1 Code

Your code should be clean, readable, and understandable. This will form one criterion of the evaluation. Please, take the time to come up with meaningful names of variables, functions, constants, etc. Keep the functions short, spanning several lines only, if possible. Make appropriate comments.

2.2 Report

The report can be written in Czech or in English. It should have a form of a scientific article. You can assume the reader has a general background in machine learning, but does not know the individual methods. The report should contain an adequate description of the data, principles, methods used and results achieved in your work. By adequate description we mean a **concise description** sufficient to **understand and replicate** the work done by you. You should not only present the results, but also try to **interpret and discuss** them. You can use the provided templates in LaTeX or MS Word.

3 Spam filter: mandatory part

Create a spam filtering algorithm, describe it, and submit it for the quality evaluation.

3.1 Specifications

In module `filter.py`, implement functions `train_filter()` and `predict()`. Do not change arguments of these functions, as they will be imported for quality evaluation.

1. `train_filter(X,y)`, which takes the training data (email texts) X, and email labels (classes "ham" or "spam") y, trains a classifier (or a pipeline with preprocessors and a classifier), and returns the trained classifier (or rather the whole pipeline).
2. `predict(filter,X)` takes the filter (classifier, pipeline) and the email texts X, and returns the predictions of the filter for this data.

3.2 Working example

In the module `filter.py`, you can see an example of a dummy spam filter:

1. The filter first transforms the training emails into the *bag of words* representation, using the `CountVectorizer` class.
2. Then it uses the `DummyClassifier` to decide if an email is ham or spam.

4 Data

In the support archive for this task, you get two datasets, which you will use for: algorithm selection, hyper-parameter tinning, and performance estimation. Use them as you wish.

The quality evaluation of your spam filters will be done on 2 other different datasets. These datasets come from the same source, its characteristics is thus be very similar to the dataset you have.

5 Filter evaluation

The filters shall be evaluated using a modified accuracy score

$$macc = \frac{n_{TP} + n_{TN}}{n_{TP} + n_{TN} + 10 n_{FP} + n_{FN}}, \tag{1}$$

i.e. by an accuracy measure which uses 10 times larger penalty for FP than for FN.

It is already implemented in the supplied `filter.py` module as the function `modified_accuracy`. It uses the confusion matrix feature to compute TP, FP, TN, FN.

6 Model tuning

Use grid search feature to search for optimal settings of each filter. When you construct a pipeline and need to tune some parameters of transformations and model inside that pipeline, you can "address" the parameters hidden in the pipeline as `<estimator>__<parameter>`. See the documentation of pipeline with the example of `GridSearch`.

When using the grid search facility, you may also need some other information about how to construct your own scoring function and use it e.g. in grid search.

7 Comparison of several filters

Compare several types of (preferably tuned) filters using

- ROC curves, and/or
- learning curves, and/or
- validation curves, and/or
- area under the curve (AUC score), ...

You can then choose the model best one as the final filter to submit. You can choose several types of classifiers from

- k nearest neighbors,
- logistic regression,
- SVM with different kernels (linear, polynomial, RBF),
- naive Bayes with multinomial distribution,
- decision trees, adaboost, random forrests,
- neural networks, ...

8 Additional ideas

To gain additional points, you can try to elaborate the following ideas:

- Try to modify the tokenization process of the `CountVectorizer`. You can set e.g. a custum analyzer like this:

  ```
  def my_analyzer(s):
      return s.split()
  vec = CountVectorizer(analyzer=my_analyzer)
  ```

  Similarly, you can set only the preprocessor, or tokenizer, you can try e.g.

  - to prevent the vectorizer to make the tokens lowercase,
  - to extract other types of tokens than character sequences, ...

- Try to evaluate the importance of individual tokens for the discrimination of spam.

- Try to choose only a subset of words as features (feature selection).

- Or you can take a look at recent advances in NLP – word embeddings – and use it instead of bag of words, cf e.g. word2vec and fastText

9 Scoring

The final score for the task will be composed of the following components:

Component	**Regular points**
Code quality (readability, understand-ability)	0-2
Report in LaTeX	0-1
Adequate description of the final filter chosen for competition	0-2
Filter tuning via grid search + adequate description in the report	0-2 per filter type, max. 5
Filter comparison + adequate description in the report	0-4
Evaluation on dataset A	0-3
Evaluation on dataset B	0-3
Component	**Bonus points**
Additional (non-trivial) ideas tried + description in the report	0-1 per idea, max. 2
Filters contest	0-3
Total	Max. 20 regular points + max. 5 bonus points

The necessary condition for the assessment is to get at least 10 regular points from this project. It is recommended to try to improve your filter score in several ways, and describe them adequately in the report. This way, you will

1. build a better filter which will score higher in the during quality assessment and in the contests (thus bringing you more points), and
2. have the chance to get additional points for this additional effort.

9.1 Quality scoring

The modified accuracy *macc* will be used to measure the quality of your filter on certain dataset. The score will be translated to points using the following table:

macc	Points
$macc < 0.50$	0
$0.50 \leq macc < 0.85$	1
$0.85 \leq macc < 0.90$	2
$0.90 \leq macc$	3

9.2 Contests scoring

As a bonus, your filters will enter a contest. Based on the results on datasets A and B, the filters will be ranked from the best to the worst.

Quartile	Points
1st	3
2nd	2
3rd	1
4th	0

10 Good luck and have fun!