Information Retrieval (Fall'16)
Assignment 1
Text Categorization

Andreas Georgiadis
15-926-710

Georgios Touloupas
16-932-550

November 16, 2016

1 Implementation Details

1.1 Data Preprocessing

In order for our system to be able to produce decent results, some text preprocessing steps are essential. The first preprocessing step covers stripping the words from irrelevant alphanumeric symbols, namely everything that is not a latin letter or an apostrophe. Subsequently, in order to build the word vocabulary, very frequent and thus non-informative words (stop words) are removed. In addition, a stemming step is applied on top of the remaining words, so that similar words which are equally informative when it comes to the topics expressed in the document are treated as the same.

Furthermore, we decided two employ the following two pruning steps. The former regards removing very rare words from the vocabulary. This way we achieved to significantly reduce the memory footprint of our algorithm and the training runtime, while not observing any notable performance degradation. The latter step covered the removal of very infrequent codes, because we observed that those codes led to the generation of many false positives. As a result, their removal reduces this problem, while it is not expected to generate many false negatives, since the codes are rare anyway.

1.2 Text Classification Approach

The general approach we followed in order to be able to assign topic codes to documents was to train one binary classifier for each code, so that when the classifier predicts a positive label for the document, we predict that the corresponding topic is expressed in the document, while when a negative label is predicted it does not. We have employed three classification algorithms, namely Naive Bayes, Logistic Regression and Support Vector Machine. For the latter two classification algorithms the feature vector we have used consists of the tfidf measure for each term in the form of a sparse vector. All terms are subject to the preprocessing strategy that was mentioned in the previous section. The terms of each document that do not belong to the vocabulary that was build using the aformentioned approach are discarded.

1.3 Naive Bayes

The fundamental challenge that we had to face in order to train a Naive Bayes classifier regarded the large memory footprint required in order to keep all the calculated probabilities in memory so that they can be accessed fastly. Apart the vocabulary and topic code pruning steps that were used in all classification algorithms, in the Naive Bayes classifier we also chose to subsample

the negative documents for each label, more specifically for each code we used a sample of negative documents, whose size was 15 times bigger than the number of positive samples, after experimenting also with other values for the multiple.

1.4 Logistic Regression

In order to train a logistic regression model, we used a variant of the stochastic gradient descent approach presented in the slides, which conducts update steps for multiple epochs, while keeping the learning rate constant for each epoch. The initial learning rate we used was equal to 1. In addition, we kept the learning rate constant in the first 10 epochs and after that point we reduced it with a rate $\propto 1/t$. Furthermore, for computational reasons we did not proceed beyond the 15 epoch, even if the F1 score was still slightly improving. Furthermore, we have employed class balancing so that positive and negative labels will have the same influence during the update steps. Finally, we used the `breeze` library provided by scala and vectorized our code so that in each update step all classifiers are updated simultaneously.

1.5 SVM

In order to train the SVM classifier, we implemented the Pegasos algorithm as provided in the lecture slides. In this case, the learning rate is decreased per training step $\propto 1/t$ and as a result the training of our model is terminated after one epoch. In addition, a similar vectorization technique as described in the Logistic Regression case was employed also here.

2 Experiments

2.1 Runtime and Memory

Several design choices had to be made throughout this project, so as to keep the runtime of the algorithms as well as their memory footprint under a reasonable limit. As far as the memory is concerned, we achieved in running all of our algorithms with a 5 GB limit on the Java heap size. The runtime of Naive Bayes and SVM were also less than 30 minutes, while logistic regression needed almost 3 hours, because of our choice to repeat the training for many epochs.

2.2 Varying the hyperparameters

The following section outlines our experimentation with the various classification algorithms. Our experimental results regard the averaged P/R/F1 per document, measured on the validation set.

2.2.1 Naive Bayes

When it comes to the Naive Bayes Classifier, we selected $\alpha = 10^{-8}$ for the Laplace smoothing parameter and set the vocabulary and codes pruning threshold to 40 and 400 equivalently. The following plots depict the tuning process, where one parameter at the time was varied and the remaining ones were selected according to the default configuration $\{\alpha, vocabularyThreshold, codesThreshold\} = \{10^{-6}, 40, 100\}$. The performance we achieved using the optimal configuration $\{\alpha, vocabularyThreshold, codesThreshold\} = \{10^{-8}, 40, 400\}$ is **(P,R,F1) = (0.54,0.57, 0.52)**.

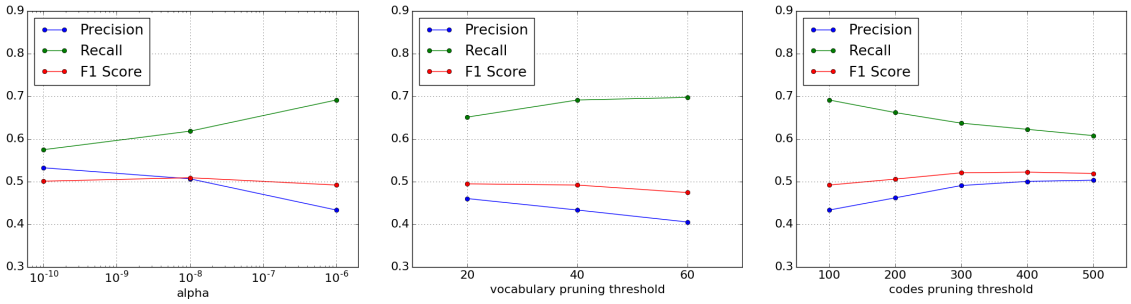

Figure 1: Naive Bayes Parameter Tuning

2.2.2 Logistic Regression

In Logistic Regression, the parameters we varied were the vocabulary and codes threshold. The default configuration is $\{vocabularyThreshold, codesThreshold\} = \{40, 100\}$ and one parameter is varied at a time. The performance we achieved using the optimal configuration $\{vocabularyThreshold, codesThreshold\} = \{60, 300\}$ is **(P,R,F1) = (0.69,0.77,0.70)**.

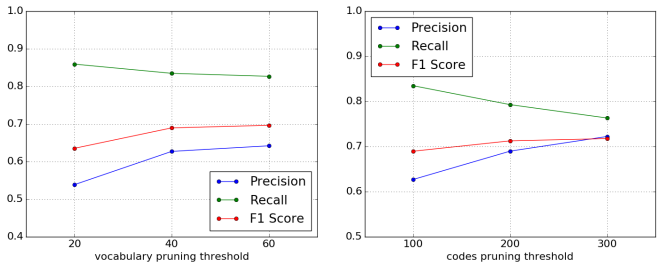

Figure 2: Logistic Regression Parameter Tuning

2.2.3 Support Vector Machine

In order to tune SVM, the parameters we varied were the regularization constant λ, and the vocabulary and codes thresholds. The default configuration is $\{\lambda, vocabularyThreshold, codes\ Threshold\} = \{0.1, 40, 100\}$ and one parameter is varied at a time. The performance we achieved using the optimal configuration $\{\lambda, vocabularyThreshold, codesThreshold\} = \{0.1, 40, 100\}$ is **(P,R,F1) = (0.91,0.72,0.78)**.

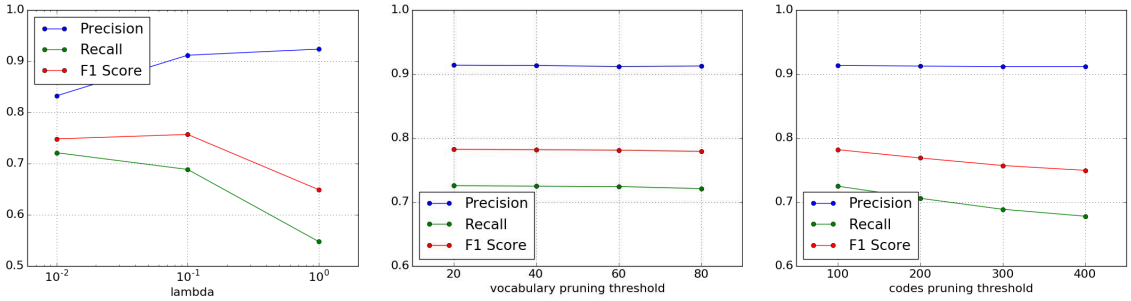

Figure 3: SVM Parameter Tuning