Project Report 1

Lidia Freitas, Gokula Krishnan Santhanam and Ankit Srivastava

November 15, 2016

1 General Strategy

1.1 Common Preprocessing Steps

1. We consider only those codes that are present in the training files and not all codes as we do not have any positive examples for these labels. This reduced the number of *codes* to 696.
2. **Stemming** - We use the Porter Stemmer algorithm to reduce words to their word stems. This helps in greatly reducing the vocabulary size and in Naive Bayes, it helps reducing the error induced by the independence assumption by reducing all the different forms of a word into its stem form.
3. **Stopword removal**

1.2 Approach

We considered an *one vs. all* strategy which can be calculated for a given test document as:

$$\bigcup_{c \in codes} \{c : P(c|d) > P(\bar{c}|d)\}$$

. Since we don't have a probabilistic interpretation[1] in the case of SVMs and Logistic Regression, this approach can be reformulated as finding if a document belongs to a class or not.

2 Naive Bayes

For implementing naive Bayes with the one vs. all approach we needed to calculate the following estimators:

$$a)\ \widehat{P}(c) = \frac{\{d : d \in c\}}{\#\{d\}} \quad and \quad b)\ \widehat{P}(w|c) = \frac{\sum_{d \in c} tf(w;d) + 1}{\sum_{d \in c} len(d) + |v|}$$

$$c)\ \widehat{P}(\bar{c}) = 1 - \widehat{P}(c) \quad and \quad d)\ \widehat{P}(w|\bar{c}) = \frac{\sum_{d \in \bar{c}} tf(w;d) + 1}{\sum_{d \in \bar{c}} len(d) + |v|}$$

$|v| :=$ *vocabulary size after preprocessing of the training documents*

Computing the values like this proved out not to be efficient since it would involve looping a lot and repeating the same calculations.

2.1 Pre-computing

For $P(c|d)$ we precompute and we store each of the following results in a map:

$$\widehat{P}(c) \quad ; \quad \sum_{d \in c} tf(w;d) \quad ; \quad \sum_{d \in c} len(d)$$

For $P(\bar{c}|d)$ instead of calculating d) as shown above which would make us go through all codes and check if they belonged to the document we precomputed the collection frequency and calculated the total length of all documents. With all this we can calculate d) in the following manner:

$$\widehat{P}(w|\bar{c}) = \frac{\sum_d tf(w;d) - \sum_{d \in c} tf(w;d) + 1}{(\sum_d len(d) - \sum_{d \in c} len(d)) + |v|}$$

2.2 Performance

The F1 average score of all validation documents was 0.66 and the total time of training and testing was 12085 s, which is roughly 3 hours and 20 min.

[1] Some probabilistic formulations include: http://www.csie.ntu.edu.tw/ cjlin/papers/svmprob/svmprob.pdf

3 Logistic Regression

3.1 Feature Selection

For feature selection we used a frequency based method: we took the K(=100) most frequent words belonging to each class which have high inverse document frequency in all the other classes, and then we took their union. By using this combination we reduced the vocabulary size from 150k to 1k words.

3.2 Algorithm

We used SGD and for each iteration of the algorithm we used matrix multiplication instead of going through all the codes and checking if they belonged to the document which improved a lot the run time of the algorithm. We tried several different values of the η and K and we realized that with increasing η the F1-average score increased, the same happened with K but it increased the memory footprint in a way that above K=100 it was taking around 3 hours to train. We ended up using ($\eta = 50$).

Table 1: LR - hyper-parameters changes and percentage of files with perfect precision and recall

η	2	1	5	10	50
10k files	0.085	0.015	0.106	0.135	0.175

3.3 Performance

The F1 average score of all validation documents was 0.74 and the total time of training and testing was roughly 40 min.

4 Support Vector Machine

4.1 Feature Selection

Apart from the preprocessing mentioned earlier, we used feature hashing [2], as these result in dense document representations which have nice theoretical guarantees akin to those provided by simhash and minhash for Jaccard Similarity. We chose to create a 10,000 dimensional feature vector for our experiments. We also tried with the same feature selection that we used in logistic regression and we ended up using it.

4.2 Algorithm

Instead of using the Pegasos algorithm as is, we used a modified version which used mini-batches of 50 files, we could see a threefold increase in training speed using this approach. Additionally, we converted all for-loops and map operations into matrix operations. To do this, we have a vector of labels for each class which had $+1$s and -1s as well as a mask vector to decide which SVM update rule to use. This enabled us to make full use of the Breeze library and exploit multiple cores for our computations, greatly speeding up the training and testing steps. We used a fixed λ and η instead of varying it with the number of iterations as we didn't see any improvements over this approach. Unfortunately, we didn't have time to tune the hyperparameters for this approach and we ended up with the version of the algorithm that we had before.

4.3 Performance

The F1 average score of all validation documents was 0.68 and the total time of training and testing was roughly 30 min.

Table 2: SVM-hyperparameters

λ	F1-avg
0.01	0.378828
0.0001	0.397135
0.000001	0.6667595

[2] http://www.machinelearning.org/archive/icml2009/papers/407.pdf

5 Conclusion

We had some troubles implementing the algorithms not because we didn't fully understand them, but because our lack of previous knowledge in scala. We also had a lot of troubles with Breeze, and most of the times it wasn't because we implemented something incorrectly but because Breeze has many open issues which we encountered (some date back to 2014![3]).

From our experience we realized that for SVM and LR feature selection was of paramount importance and the performance of our algorithm depended greatly on the extracted features.Since the training corpus was quite large, it was difficult to try out many feature extraction strategies solely due to the its computationally intense nature. We also realized how to speed up our code by using matrix formulations wherever possible and exploiting BLAS libraries like Breeze or numpy.

Performance wise we expected SVMs to outperform Logistic Regression and Naive Bayes by a large margin. We quickly realized that this depended on choosing the right hyperparameters which was a time consuming task.

[3]See: https://recordnotfound.com/breeze-scalanlp-4557/issues