Text Mining Assignment 1

Niels Witte & Julia Wąsala

October 15, 2021

1 Introduction

This report is a summary of our findings for the first assignment of Text Mining course. In this report we compare three different classifiers from the scikit-learn package[1]; Naive Bayes which is a probabilistic model that uses previous observations to determine the probability that the next observation is of a specific class, Support Vector Machine which continuously tries to fit a linear regression model such that it splits the classes whilst maximizing the distance between data points and the models. Lastly, we compare with a Random Forest model which creates an ensemble of multiple decision trees. We then experiment with different pre-processing methods to optimize the performance of these classifiers. We then take this one step further by performing Hyper Parameter Optimization on the pre-processing steps in order to further boost the performance of our classifiers.

2 Experiments

This section briefly describes the experiments that we carried out in order to solve the classification problem for our given dataset. The dataset contains a total of 1800 samples and was split 63/37 for the training and test set. Fitting and optimization was done on the training set where as evaluation presented in section 3 was done on the test set.

Experiment 1: Comparing methods In this experiment we compare three different classifiers; Naive Bayes (NB), Support Vector Machine (SVM) and Random Forest (RF). We then tested these classifiers with different pre-processing configurations namely; Count Vectorizer, Term Frequency (TF) and Term Frequency - Inverse Document Frequency (TF-IDF) which gives us a total of nine configurations to test. We initially set the parameters of these models to their default values in the `sklearn` library.

Experiment 2: Optimizing pre-processing steps In order to further optimize the performance of our classifiers we optimized the parameters of the pre-processing steps. This process, also known as Hyper Parameter Optimization can be completely automated using Hyper Parameter Optimization Algorithms, in our case Grid Search (with 5-fold crossvalidation). The different vectorizer parameter values we tested can be found in Table 1. The different numbers of max features were chosen to investigate whether using less features than the document count (which is 18000) would perform better.

3 Results & Discussion

First we will discuss the results of the different classifiers with default parameters as can be found in Table 2. There is little difference between the scores of the different classifiers. However, both Naive Bayes and SVM have the highest scores with tf-idf.

Parameter	Values
Lowercase	`True, False`
Stop words	english, `None`
Analyzer	word, char, char_wb
N-gram range	[(1, 1), (1, 2)]
Max features	500, 1000, 1500, 3000, `None`

Table 1: Vectorizer parameter space for Grid Search

For almost all classifier, feature combinations the following parameters were returned by grid search: analyzer "word"; Lowercase `True`; Max features `Null`; N-gram range [1, 2], and stopwords english. The only exceptions were SVM with only the count and SVM with tf-idf, the optimal value for Lowercase was `False` for these classifiers. It's notable that the results are so similar, but then again the parameter space was also not very large. Another observation one can make is that apparently there is a preference for a large number of features, i.e. `None` does not constrain the number of features considered by the classifier.

Now when looking at the performance using the optimised parameters, the scores of the classifiers show some improvement. Overall the precision scores are higher than recall. The results of the classifier are still very similar, with this time SVM with tf scoring the highest in terms of precision, recall and F1 score.

	Default parameters				Optimised parameters		
	Feature	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
NB	**count**	0.762	0.764	0.745	0.817	0.797	0.793
NB	**tf**	0.792	0.682	0.673	0.82	0.795	0.794
NB	**tf-idf**	0.826	0.757	0.756	0.806	0.773	0.769
SVM	**count**	0.755	0.746	0.745	0.778	0.775	0.774
SVM	**tf**	0.779	0.758	0.755	0.837	0.825	0.824
SVM	**tf-idf**	0.828	0.813	0.812	0.803	0.79	0.788
RF	**count**	0.777	0.754	0.753	0.81	0.788	0.789
RF	**tf**	0.764	0.741	0.739	0.803	0.782	0.782
RF	**tf-idf**	0.775	0.753	0.751	0.801	0.781	0.782

Table 2: Performance of our selected models with different pre-processing steps, both with default parameters and with Grid Search optimised parameters.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.