Text Mining Hand-in Assignment 1

Elin Benja Dijkstra (s2696096)
October 20th 2019

1 Methods

Our data for this assignment consists of a collection of short newsgroup documents. These documents are nearly evenly distributed over 20 categories. The task at hand is to train a classifier which can successfully predict the category of unseen documents. Since we have a good baseline classifier from following the tutorial, we will now experiment with different classifiers and features.

The first thing we want to do is expand our training set to contain all twenty categories. When fetching the data, the tutorial uses a filter where $categories = catergories$. By removing this filter we run the tutorial on all target categories in the data set.

1.1 Classifiers

To classify the datasset, three different classifiers are implemented: Multinomial Naive Bayes, Stochastic Gradient Descent and Linear Support Vector Classifier.

1.1.1 Multinomial Naive Bayes

A Naive Bayes classifier is based on the prior probability $p(c)$ of categories. It assumes that each of the features are conditionally independent given a class. For documents, it computes the posterior distribution over the possible categories based on the Bayes' Theorem.

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{1}$$

The Multinomial Naive Bayes used in this implementation is a variant of Naive Bayes Classifier which assumes the conditional probability to be a Multinomial distribution. This classifier is known to work well for data which can easily be turned into counts. Since word counts can be computed from text, it is a successful classifier for text data.

1.1.2 Stochastic Gradient Descent

The SGDClassifier uses linear classifiers such as SVM and logistic regression with stochastic gradient descent learning. By default it fits a linear support vector machine. The model is updated with a step size of the learning rate in the direction of the negative gradient of the loss function.

1.1.3 Linear Support Vector Classifier

The linear support vector classifier determines the best decision boundary between categories. It does do by maximizing the margin between points within certain categories and the maximum margin line or hyperplane.

1.2 Gridsearch in the feature space

All above mentioned classifiers have various different feature settings which can influence the performance of the model. To enhance the performance, we do a gridsearch on a pre-defined parameter space and find the optimal settings within this search space. Our parameter space is defined as follows:

Parameter	Possible settings	Description
Tokenization	Counts; TF; TFIDF	Counts: Term Frequency TF: Term frequency in a document / Term frequency in the corpus TFIDF: TF x Inverse documents frequency ($log_2(\frac{N}{nt})$)
Lower case	True; False	Convert all characters to lowercase before tokenization
Stop words	'english'; None	Filters stop words out of text based on a collection or on frequency
Analyzer	'word'; 'char'; 'char_wb'],	Whether the feature should be made of word or character n-grams
Ngram range	(1,1); (1,2)	The lower and upper boundary of the range of n-values range for different n-grams to be extracted
Max. features	None; 100; 1000; 10000	Build a vocabulary based on the top[max_features] ordered by TF across the corpus.

Table 1: Possible setting for our classifiers with short description

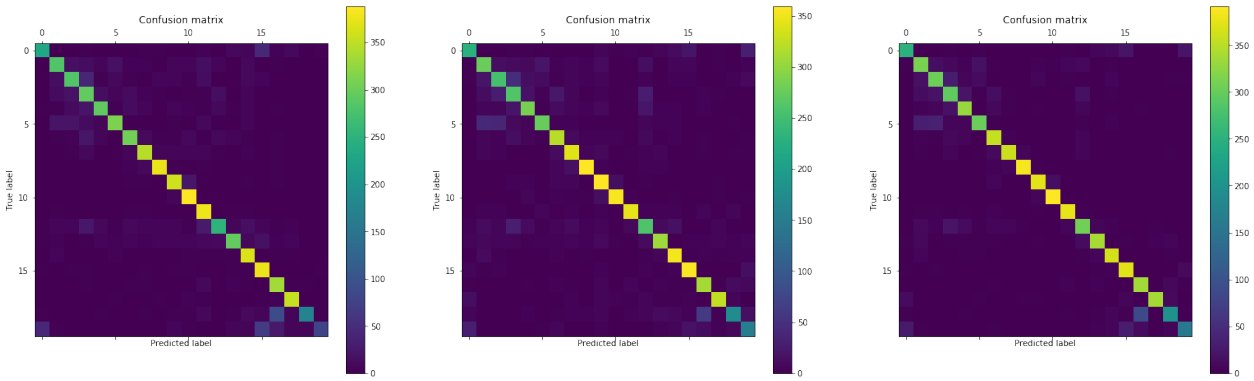
2 Results

The results of the performed gridsearch showing the optimal features in our search space are displayed below in Table 2. The optimal parameters for the three different classifiers are very similar, with the stop words setting for Naive Bayes as only exception.

Classifier	Best found feature Set	Best score on trainingset	Precision	Recall	F1 Score
Naive Bayes	tfidf___use_idf: True vect___analyzer: 'word' vect___lowercase: True vect___max_features: None vect___ngram_range: (1, 2) vect___stop_words: 'english'	0.8828	0.82	0.81	0.80
SGD	tfidf___use_idf: True vect___analyzer: 'word' vect___lowercase :True vect___max_features: None vect___ngram_range: (1, 2) vect___stop_words: None	0.9033	0.84	0.83	0.83
LinearSVC	tfidf___use_idf: True vect___analyzer: 'word' vect___lowercase: True vect___max_features: None vect___ngram_range: (1, 2) vect___stop_words: None	0.9287	0.86	0.86	0.86

Table 2: Results of the grid search for the three classifiers with weighted measures

From the table we see the weighted precision, the proportion of the assigned labels that are correct, and the weighted recall, the proportion of the relevant labels that were assigned. The F1 score is a weighted sum of the precision and recall. In all three experiments, these three values are close together. However, the values fluctuate between the twenty categories. In Figure 1 we can interpret the misclassifications by the models by looking at their respective confusion matrices.



(a) Naive Bayes (b) SGD (c) SVM

Figure 1: Confusion matrices for the best estimators for three Classifiers

In the bottom right corner we see that category 19: 'talk.politics.misc' is likely to be misclassified as catergory 17: 'talk.politics.guns'. Since these two subjects are both related to politics, the misclassification is not surprised. Another common misclassificiation is between category 6: 'comp.windows.x' and category 3: where both are related to Windows.

In both of these cases, we see that there is a miscellaneous category involved where odd documents in a subject are classified together. It seems to be the case that the model recognized the "umbrella" subject correctly, but sees terms related to a more specific categories and thus classified it in this instead of miscellaneous. These are likely to be words often used in this subject. In the plots of the confusion matrices, we see that all classifiers more or less have difficulty with the same topics but overall perform well with a strong diagonal.

3 Discussion

From the results in Table 2 we conclude that the LinearSVC works best as the classifier on our dataset in combination with the best found feature set. For the determining the most meaningful words in a document, TF-IDF is found to be the best setting. The optimized CountVectorizer settings include transforming all characters to lowercase, not using a predefined list of stop words, no limit of the number of features and that the feature should be made up out of words and not n-grams. With these settings an F1 score or 0.86, a precision of 0.86 and a recall of 0.86 is achieved. To enhance the perfomance even more, other classifiers could be tested and the grid search parameters could be expanded as well as classifier