Assignment 1: Text Categorization

Arsen Ignatosyan s4034538
Severin Holtmann s3624099

Introduction

Goal

In this assignment, we aim to train a model that can identify the label of a news article. The main approach will be a Naïve Bayes model with counts, td, and tf-idf as feature extractors. We will also compare this approach with decision trees and support vector machines. We may then compare the nine unique models to see which performs best on a test set. Using the best model, we will investigate the effects of tuning the hyperparameters of the *CountVectorizer* function.

Data

The dataset "Twenty Newsgroups" is a large corpus of documents from different newspapers, featuring articles about different topics. Each article is labeled according to its category. There are 20,000 articles which are mostly evenly distributed over 20 different categories. The data will be split into a train and test set with a 3:2 ratio.

Theoretical Background

Feature Extractors

In document classification, *Counts* refer to the simple occurrence count of each word or term in a document. *TF (Term Frequency)* measures the relative frequency of each word in a document, often normalized by the total number of words in the document, thereby providing a measure of the importance of a word within that document. *TF-IDF (Term Frequency-Inverse Document Frequency)* expands on TF by incorporating IDF. IDF measures the inverse document frequency to highlight words that are important in a specific document but not too common across all documents. TF-IDF is widely used for text classification to weigh terms based on their importance in a document relative to their importance in the entire corpus.

Models

Naïve Bayes

Naïve Bayes is a probabilistic algorithm. It operates on the assumption that the occurrence of words in an article is independent given its class, allowing it to efficiently calculate the probability of an article belonging to each category based on the frequencies of words in the text. The algorithm calculates these probabilities for each category and assigns the article to the category with the highest probability. This method is widely applied in natural language processing tasks due to its efficiency and ability to handle large corpora of textual data.

Decision Tree

Decision trees are a machine learning algorithm useful for feature selection and classification. These trees recursively split the dataset based on the most informative word or feature at each node, effectively creating a hierarchical structure of decisions. By evaluating the presence or absence of specific words or phrases, decision trees can classify the articles according to our labels.

SVM

Support Vector Machines (SVM) aim to find a hyperplane that best separates different classes of text documents in the high-dimensional corpus space. The core idea is to represent text documents as feature vectors, where each dimension corresponds to a term or feature's importance. SVM then seeks to find the hyperplane that maximizes the margin between the classes while minimizing classification errors.

Methodology

Model Performance

Naïve Bayes

Extractor	Counts	tf	tf-idf
Weighted F1	0.75	0.69	0.77

Judging by the overall f1 performance, the tf-idf performed best. This suggests that the model correctly identifies a substantial portion of the relevant instances while maintaining a high level of correctness among the instances it classifies as positive.

Decision Tree

Extractor	Counts	tf	tf-idf
Weighted F1	0.57	0.56	0.56

The results are effectively identical between the feature styles. However, the model generally performs much worse than Naïve Bayes.

SVM

Extractor	Counts	tf	tf-idf
Weighted F1	0.14	0.74	0.82

Here, we observe that the feature style may have a substantial impact on the performance of a text classification model. SVM features both the worst performance, using classical counts, as well as the best, using full tf-idf scoring. The f1 score of 0.82 suggests that the model manages to classify very well. Precision and recall are also nicely balanced, which proves that this is a very robust model.

Therefore, this model will be used for further investigation of the effects of hyperparameter tuning.

Hyperparameter Tuning

Lowercase

The *lowercase* argument simply converts all words to lowercase. By default, this is set to *True*. The benefit of this is that it simplifies the feature space with relatively little loss of information. We will investigate the effect of deactivating this feature, thereby making the model case-sensitive.

Setting lowercase to False gives a weighted average f1-score of 0.81. The results are almost identical to the default. While arguably within a margin of error, case sensitivity appears to make the model slightly worse. This may be attributed to the fact that including case-sensitivity

broadens the feature space, and inevitably, slightly regularises the vector density. While SVM generally excels at high-dimensional classification, the increase in dimensions likely obfuscates the hyperspace, making the specification of the optimal hyperplane more challenging.

Stop Words

Stopwords are ubiquitous in most sentences, however, they do not provide any context or useful information. Thereby, they act as a nuisance in the feature space. The *stop_words* argument tells the vectorizer to identify and omit stop words of a given language, English in our case. This can have a particular impact on more crude feature specification methods, such as counts, as the words are typically the most prevalent in any given document, and thus also in the overall corpus. As it generally simplifies the feature space, it should also enhance training of the SVM.
Removing stop words yields a weighted average f1-score of 0.81. The improvement is again negligible and does not suggest that there is an obvious difference. Nevertheless, it does make the results more robust, as was expected. We will thus proceed with keeping the feature activated.

Analyzer and n-gram range

The *analyzer* and *ngram_range* arguments specify token length and amount of surrounding n-grams considered during tokenization respectively. *analyzer* may be set to *'word', 'char', 'char_wb'*, or a callable, *'word'* being the default. With *'word'*, each word is a token. *'char'* tokenizes each character separately. *'char_wb'* adds consideration of word boundaries to character separation. This means that multiple words may be considered in one token on a character level. A callable allows for a proprietary algorithm to tokenize with. However, we will only investigate the standard options.

ngram_range takes a tuple of two integers: *(min_n, max_n)*, where *min_n* is the minimum size of the n-grams, and *max_n* is the maximum size. Larger ngram ranges allow for consideration of larger token sequences, thereby taking more context into consideration.

arguments	‘word’, (1, 2)	‘word’, (1, 3)	‘char’, (1, 1)	‘char’, (2, 2)	‘char_wb’, (1, 1)	‘char_wb’, (2, 2)
Weighted F1	0.81	0.80	0.27	0.62	0.22	0.62

First, by also allowing for bigrams, we do not improve performance. One reason for this may be that it overcomplicates the hyperspace. This is further confirmed by additionally allowing for trigrams, which causes an even greater decrease in performance. Character tokenization does not allow for stop word omission. Although character-level investigation is not really applicable

to the problem, we investigated this nonetheless. We find that evaluating each character separately has a detrimental effect on performance. While investigating characters seems inappropriate for such a large corpus of newspaper articles, using Bigrams instead is better than Unigrams, which also makes intuitive sense, as it allows for greater associations. We also investigated bounded character tokenization. While word bounding can put more emphasis on context than normal characterization, the effect is small when only using unigrams. Indeed, we receive similarly poor performance as with regular unigram character tokenization. The difference probably becomes more noticeable once more characters are considered, as can be seen by the bigram performance. All in all, we will retain the default parameters.

max_features

max_features fixes the size of the token vector, thereby only allowing for the 'n' most popular features. The default is *none*, in which case all tokens are included in the feature vector. The SVM without stop words and unigram word tokenization has 129,796 features in total. We will start at a much lower size of 1,000, and gradually increase the feature size to inspect the impact on test set performance.

max_features	1000	2000	3000	4000	8000
Weighted F1	0.69	0.74	0.77	0.78	0.81

All in all, we can see that using a fraction of the original features is still performing almost as well. This could perhaps be viewed as a useful regularisation measure, as we reach almost identical performance to the full model at around 8000 features.

Conclusion

In conclusion, we have investigated the performance of several machine-learning approaches for labeled text classification. Additionally, we evaluated the difference between the three most common extractors; ‘counts’, ‘tf’, and ‘tf-idf’. This has shown us that a Support Vector Machine with ‘tf-idf’ performs best on our test corpus. Notably, the rather simple ‘counts’ extractor also showed good performance with a Naive Bayes approach. Overall, it should be acknowledged that Naive Bayes has the most robust performance for all extractor types. This should be taken into consideration when working with a corpus that may only work properly with certain types of extractors.

Furthermore, several hyperparameters of the *CountVectorizer* function were explored to optimize the model. The only truly beneficial option was to omit stop words. This is a useful technique to simplify the model. Besides this, we investigated several combinations of tokenization procedures and ngram structures. As expected, it has become apparent that such

a corpus is more attuned to word tokenization. Allowing for larger ngram evaluation, however, did not improve the performance. Lastly, we tried to see if reducing the feature space would allow for a simpler model with not too much performance loss. Indeed, we were able to obtain a model with almost identical performance, but only ~6% of the feature size. All in all, it is recommended to explore different models and tunings, as performance can be volatile without much a priori intuition on the causes.